MASTER THESIS

ECONOMETRICS AND MANAGEMENT SCIENCE

The Multi-Compartment Vehicle Routing Problem with Time Windows and Split Deliveries

---

Author:
Liana van der Hagen
409165

Supervisors:
Dr. R. Spliet (EUR)
S. Rieß MSc (ORTEC)

Second assessor
Dr. W. van den Heuvel (EUR)

Erasmus School of Economics
ERASMUS UNIVERSITY ROTTERDAM

December 13, 2018

Abstract

This thesis considers an extension of the Vehicle Routing Problem with Time Windows and Split Deliveries, in which customers demand different types of products. These products must be transported within the right temperature regime and without contaminating other products. To do so, the vehicles can be equipped with bulkheads to divide the vehicle in several compartments, specified by one of the pre-defined vehicle configurations. To solve this problem, we propose an Adaptive Large Neighborhood Search framework, in which the ruin and recreate principle is used for creating the routes, with an order related removal heuristic. This heuristic is specifically designed for this problem and often outperforms the other removal heuristics by improving solutions towards the end of the search. The feasibility of each insertion by one of the insertion heuristics, is checked based on the product related requirements with our developed algorithm. We propose a heuristic method to check per possible vehicle configuration and per temperature regime present in the compartments of that configuration whether or not a feasible assignment to the compartments exists. Computational experiments on both real-life and generated instances show that the heuristic is able to provide an optimal solution for many instances in reasonable time, compared to the much longer runtime which is required to solve the problem with CPLEX. The experiments also show that the overall routing solution can be better when an optimal solution is not always found, compared to when an optimal solution is found for each of these feasibility checks. Finally, we present a method that consolidates the smaller parts of which the customer demand consists into larger groups each of which is transported in one vehicle. This method consolidates in such a way that the routing algorithm can find routes for which vehicles are more efficiently filled, which is demonstrated by the experiments.

Contents

List of Symbols

Customers

N	customer nodes
$[a_i, b_i]$	time window of customer $i \in N$
s_i	service time at customer $i \in N$
X_i	suborders requested by customer $i \in N$
X	$\bigcup_{i \in N} X_i$
l_x	size of suborder $x \in X$
P	product types
p_x	product type of suborder $x \in X$
$P(\bar{X})$	product types of suborders $\bar{X} \subseteq X$
Z	temperatures
z_p	temperature required for product $p \in P$
$h_{p\tilde{p}}$	parameter equal to one if $p \in P$ and $\tilde{p} \in P$ contaminate and zero otherwise
O_i	orders of customer $i \in N$
O	$\bigcup_{i \in N} O_i$
X_o	suborders in order $o \in O$
l_o	total size of suborders X_o, $o \in O$
Z_o	temperatures required for products in suborders X_o, $o \in O$
X_o^z	suborders in order $o \in O$ with required temperature z

Routes

K	available vehicles
$\hat{d}$	depot node
$[a_{\hat{d}}, b_{\hat{d}}]$	time window at depot $\hat{d}$
$s_{\hat{d}}$	service time at depot $\hat{d}$
Q	vehicle capacity without bulkheads
M	configurations
C_m	compartments specified in configuration $m \in M$
C_m^z	compartments in C_m with temperature regime z
q_c	capacity of compartment $c \in C_m$, $m \in M$
z_c	temperature regime of compartment $c \in C_m$, $m \in M$
X_k	suborders assigned to vehicle $k \in K$
Z_k	temperatures required for suborders X_k, $k \in K$
X_k^z	suborders assigned to vehicle $k \in K$ with required temperature z

Adaptive Large neighborhood Search

d_{ij}	distance between nodes i,$j \in N \cup \{d\}$
t_{ij}	travel time between nodes i,$j \in N \cup \{d\}$
n	number of iterations
U	order bank
q	number of orders to remove
p	randomness parameter in distance and time related removal heuristic
τ	number of closest orders considered in time related removal heuristic
p_{worst}	randomness parameter in worst removal heuristic
T	temperature simulated annealing
c	cooling rate simulated annealing
w	parameter to determine start temperature simulated annealing
ρ	reaction factor weight adaptation for the heuristics
σ_1	score increment if new global best solution
σ_2	score increment if better, not yet accepted solution
σ_3	score increment if worse, not yet accepted solution
η	parameter for determining the amount of noise

Load assignment

V	$X_o^z \cup X_k^z$
C	C_m^z
D	products in conflict clique
D_p	products in non-conflict clique containing $p \in P$
D'_p	non-conflict clique of suborders corresponding to D_p
$o(D'_p)$	overlapping suborders in D'_p
κ	size of segment for configuration sorting
α	influence parameter configuration sorting

Consolidation of suborders into orders

$Q(o)$	maximum capacity of a vehicle containing order o due to bulkheads
$a(o)$	smaller approximate vehicle capacity of a vehicle containing o
A	orders with size larger than $a(o)$
$b(o)$	larger approximate vehicle capacity of a vehicle containing o
B	orders with size larger than $b(o)$
γ	parameter to determine $a(o)$
β	parameter to determine $b(o)$
O_i^n	neighboring orders of customer $i \in N$
$\delta(i,j)$	maximum allowed distance from customer i to a neighboring customer j
σ_i	number of times still allowed to split an order of customer i

1 Introduction

Due to the importance of efficient transportation of goods, the Vehicle Routing Problem (VRP) and several extensions of the problem have already received much attention. Many companies face the problem of having to transport products in an efficient way to various different stores in order to be able to compete with competitors and save total costs involved with the transportation. Therefore, routes must be constructed in such a way that vehicles cover the least distance possible, less vehicles are needed and many more objectives might be taken into account.
Additionally, customer satisfaction must be increased. Customers might choose a specific time window in which the goods must be delivered, and goods might be restricted to be delivered by a maximum number of vehicles. The latter restriction can be seen as a restriction on the relaxed VRP which allows for split deliveries, the Vehicle Routing Problem with Split Deliveries (VRPSD). The reason behind the introduction of the VRPSD is that allowing for multiple visits at a customer location might lead to more efficient filling of the vehicles and to overall cost savings.
In real life applications, many more attributes of the routing problem must be taken into account. In this thesis, we consider a multinational retail company, with a large number of customers. The customers require different types of products which must be transported within different temperatures. To do so, each vehicle might be devoted to transport goods with one specific required temperature. The vehicles of the company can, however, be equipped with bulkheads that divide the trailer of the vehicle into several compartments, which might have different temperatures. By considering these bulkheads, more routing possibilities are created, which results in increased flexibility and complexity. Additionally, the retail company restricts that some types of products may not be transported in the same compartment due to contamination, which contributes to an even more complex problem.
The purpose of this research is to develop a suitable method for this rich VRP, which takes into account the extra restrictions and additional flexibility of the retail company, such as time windows, split deliveries and vehicles with multiple compartments.
The research is carried out at ORTEC, one of the largest providers of optimization software in the world. ORTEC provides planning and scheduling software and operates in various branches, such as retail and oil industry. The company provides, among other things, vehicle routing, loading and workforce scheduling solutions.
The structure of this thesis is as follows. In Section 2 we provide a literature review on the problem. The problem and its separate elements are precisely defined in Section 3. Thereafter, we discuss the routing algorithm in Section 4, without distinguishing between different product types and without considering vehicles with multiple compartments. In Section 5 and Section 6 we present the extension of the methodology, that is required to handle these additional elements. The experiments that have been designed to test these methods and their results are discussed in Section 7. Finally, Section 8 contains a conclusion and a discussion.

2 Literature Review

2.1 Vehicle Routing Problem

The Vehicle Routing Problem and several of its variants have been studied extensively in the literature. Variants of the general problem include the Capacitated Vehicle Routing Problem (CVRP), in which the amount of loaded products cannot exceed vehicle capacity,

and the Vehicle Routing Problem with Time Windows (VRPTW), in which customers must be visited within their specified time windows (Baldacci et al., 2012).
Due to the complexity of these problems, mainly heuristic methods have been proposed to solve them. A survey on several constructive heuristics, improvement heuristics, population mechanisms and learning mechanisms is given by Cordeau et al. (2005). Additionally, some good performing heuristics are covered more extensively and compared.
One widely used solution method to avoid local minima, Large Neighborhood Search (LNS), is introduced by Shaw (1997). In LNS different removal and insertion heuristics are used to destroy and repair solutions. Ropke and Pisinger (2006) add an adaptive layer to the LNS, introducing Adaptive Large Neighborhood Search (ALNS) for the Pickup and Delivery Problem with Time Windows. This heuristic selects the removal and insertion heuristics based on adaptive selection probabilities. Pisinger and Ropke (2007) consider five different variants of the VRP, including the VRPTW, which they transform into a Rich Pickup and Delivery Problem with Time Windows and solve with ALNS. They show that the heuristic can be used for a variety of different problems, providing good results.
In real applications of Vehicle Routing Problems, many additional constraints and characteristics should be taken into account. A survey on heuristics for these so-called Multi-Attribute Vehicle Routing Problems is given by Vidal, Crainic, Gendreau, and Prins (2013). The authors argue that successful metaheuristics result from a balance between several elements, such as a good balance between diversification and intensification.

2.2 Vehicle Routing Problem with Split Deliveries

The Vehicle Routing Problem with Split Deliveries (VRPSD) is introduced by Dror and Trudeau (1990). This is a relaxation of the VRP, which is still NP-hard, and could result in savings in total distance travelled and in the number of used vehicles. Since then, many methods have been proposed to solve the VRPSD. An extensive survey on the solution approaches for the VRPSD can be found in Archetti and Speranza (2012).
Ho and Haugland (2004) develop a tabu search algorithm for the Vehicle Routing Problem with Time Windows and Split Deliveries, in which vehicles may deliver a fraction of the total demand size for one customer. The method is applied to the Solomon benchmark instances (Solomon, 1987) both when allowing for split deliveries, as well as when not allowing for split deliveries. The method improved for some instances the best solution known until then, even without split deliveries. The experiments also show that due to the small ratio of order size to vehicle capacity and short scheduling horizon, no split deliveries are actually made. They also prove that when the costs on the arcs satisfy the triangle inequality, the problem has an optimal solution where no two routes have more than one customer in common, which was already proven for the problem without time windows (Dror & Trudeau, 1990).
Archetti, Speranza, and Hertz (2006) propose a simple tabu search algorithm for the problem in which the demand sizes are integers and each vehicle must deliver an integer amount to each customer. The advantage of this tabu search is that only few parameters have to be tuned: the length of the tabu list and the maximum number of iterations without improvement. An initial feasible solution is obtained with a construction heuristic that first creates as many direct trips as possible for each customer. A reduced instance of the problem then consists of the remaining demands of the customers. A large Traveling Salesman Problem (TSP) tour is obtained for this reduced instance with the GENIUS algorithm (Gendreau et al., 1992) and cut into pieces such that capacity of the vehicles are satisfied.
This tabu search algorithm is used in combination with an integer program by Archetti,

Speranza, and Savelsbergh (2008). Archetti et al. (2008) present a route-based formulation. The key idea of their solution approach is that the tabu search identifies parts of the solution space with high quality solutions. When demand of a customer is not or rarely split during the tabu search, it is imposed in the integer program that the customer is only visited once. Also, the size of the graph is reduced by disregarding edges that are rarely used and a set of promising routes is made. As it is impossible to solve the problem with all promising routes, a good subset of these routes is selected, based on several criteria. For all instances, except for one instance, the results were better than the ones obtained by Archetti et al. (2006).
Derigs, Li, and Vogel (2010) have modified different local search moves that have been proposed in the literature for standard vehicle routing problems, in order to support the case in which split deliveries are allowed. Different metaheuristics, such as Simulated Annealing and Attribute Based Hill Climber heuristic (ABHC), are compared. The methods are tested on benchmark instances from Archetti et al. (2008) and ABHC performed best on these instances.
The VRPSD and variants of the problem can be considered in different applications. The VRP with time windows and split deliveries is considered in a retail context by Yoshizaki et al. (2009). The authors propose a scatter search method and apply it on data from a large Brazilian retail group. The method provided better solutions than the actual company solution.
Another case study is considered for an Italian company in the food industry which delivers fresh/dry or frozen products (Ambrosino & Sciomachen, 2007). Ambrosino and Sciomachen (2007) present a Mixed Integer Program (MIP) formulation for the Multi-Product VRPSD, in which the number of stops is restricted when transporting frozen goods in order to avoid temperature variation. It is assumed that due to the availability of bulkheads the different kinds of products can be loaded on the same vehicle, but this is not included explicitly in the model. That is, the information on different kinds of products is only used for the restriction on the number of stops and it is simply assumed that compartments can be adjusted in such a way that the vehicle can hold all products. The authors suggest a cluster-first, route-second approach to find an initial solution. A local-search improvement algorithm, which allows for the splitting of demand, is then applied to the initial solution. The benefit from allowing split deliveries was mainly visible in a reduction of the number of vehicles used.
Archetti, Campbell, and Speranza (2014) consider four different problems and perform a worst-case analysis. The authors distinguish the cases in which vehicles can transport any set of commodities or only one commodity and the cases in which demand may or may not be split over different vehicles. Solving the Split Delivery Mixed Routing problem, which allows splitting of demand over multiple vehicles and allows vehicles to transport different commodities, corresponds to solving the VRPSD (Archetti et al., 2014). Here it is assumed that the commodities can be transported in any combination in the vehicles. A general algorithmic framework for the four types of problems is proposed based on the work of Archetti et al. (2008).
A customer-oriented Multi-Commodity VRPSD, in which total waiting time of customers is minimized and multiple commodities can be required per customer, is proposed by Moshref-Javadi and Lee (2016). The problem consists of determining vehicle routes and the quantities of different commodities to load and unload, where each vehicle can transport any combination of commodities. One of the relevant applications of this problem is in disaster response.
The VRPSD with extra restrictions for the possible splitting of demand has also been studied in the literature. Gulczynski, Golden, and Wasil (2010) studied the Split Delivery Vehicle

Routing Problem, with the restriction that the split suborders must have a minimum size, which is a pre-defined percentage of the total order size. This extra restriction limits the inconvenience for the customer of having very small orders delivered many times. They formulated an endpoint MIP with minimum delivery amounts, in which only the delivery at endpoints of a route may be split. The idea behind considering only the endpoints, is that the endpoints are closest to the depot and closer together which leads to more efficient splits. Their proposed heuristic combines this endpoint mixed integer program with an enhanced record-to-record travel algorithm.
Another variant of the VRPSD is the problem in which the demand of a customer may only be split over different vehicles when multiple commodities are required by that customer (Archetti et al., 2015). This means that all products of one type have to be transported to the customer by the same vehicle. This is imposed such that the inconvenience for the customer is limited. The vehicles are allowed to transport any combination of the commodities. The authors propose a set partitioning formulation and use a branch-price-and-cut approach to solve the problem.
The restrictions on the split deliveries for the customers considered in this thesis are different from what is studied in literature. Instead of allowing to split the demand size arbitrarily over different vehicles, we assume that the demand of each customer is already divided into smaller parts of different sizes each of which cannot be split further and must be transported in one vehicle. For the explicit problem description we refer to Section 3.

2.3 Multi-Compartment Vehicle Routing Problem

The Multi-Compartment Vehicle Routing Problem (MCVRP) is an extension of the standard VRP, in which the vehicles have multiple compartments. The benefits of co-distribution by vehicles with multiple compartments are examined by Muyldermans and Pang (2010). When more commodities are transported, the customer locations are more spread, or the vehicles have a larger capacity, the benefits of having vehicles with multiple compartments are shown to be higher.
Derigs et al. (2011) describe a problem formulation for the MCVRP, where all vehicles have the same set of compartments. Incompatibilities between products that may not be transported together in the same compartment, or restrictions that products are not allowed in some compartments, are also taken into account. Both the vehicle itself and all compartments have a specified capacity. When the total capacity of the compartments is larger than the vehicle capacity, the compartments can be seen as flexible. The authors also created some test instances in which either incompatibility between products exists, which occurs in the petrol industry, or products are not allowed in every compartment, which is a restriction in the food industry. Several heuristic methods that are widely used in the literature for the standard VRP, are adapted for the case of multiple compartments and compared. The choice of metaheuristic was shown not to have a very large impact on the solution quality.
The MCVRP and variants of the problem have a wide variety of applications.
In the milk collection problem only one type of milk can be assigned to each compartment in a vehicle (Caramia & Guerriero, 2010). In the application considered by Caramia and Guerriero (2010) some locations are only accessible by trucks without a trailer and therefore uncoupling of the trailer is possible at a parking place. The authors developed a two-phase approach. In the first phase the farmers are assigned to vehicles and in the second phase the routes are determined. For both parts a mathematical programming formulation is proposed. This is combined with a local search and a restart mechanism. In the case study

158 customers of an Italian company are considered, for which the solutions have proven to be better than how the company first operated.
Cornillier, Boctor, Laporte, and Renaud (2008) developed an exact algorithm for the problem that determines the allocation of petroleum products to compartments and determines routes of the trucks. The compartments have to be emptied in their entirety due to absence of flow meters and can only hold one product. The routing problem is NP-hard but when the number of customers per route is at most two, the problem reduces to a Matching Problem.
Multiple compartments also have their application in ship scheduling. Fagerholt and Christiansen (2000) propose a two-phase set partitioning approach for the Ship Scheduling and Allocation Problem.
Hvattum, Fagerholt, and Armentano (2009) discuss different variants of the Tank Allocation Problem, by describing several constraints that might be required in real world problems. The authors show that the problem is NP-complete, but find that for many of their considered instances a feasible solution could be found within reasonable time. However, when the problem arises as a subproblem for evaluating feasibility frequently in a local search based metaheuristic, it might be better to resort to heuristic methods.
Mendoza, Castanier, Guéret, Medaglia, and Velasco (2010) formulate the MCVRP with stochastic demands as a stochastic program with recourse for the case of collection routes and vehicles with compartments of fixed size for each product type. The recourse action corresponds to a return trip to the depot to unload all compartments and after that complete the service at the customer location. To solve the problem Mendoza et al. (2010) propose a memetic algorithm, which is an algorithm that uses local search procedures to intensify a genetic search.
Eglese, Mercer, and Sohrabi (2005) consider the MCVRP with flexible compartments in which loading feasibility plays an important role. For example, frozen products have to be in the front compartment and can only be delivered after the other compartments are emptied. The authors propose a heuristic based on simulated annealing to solve the problem.
Henke, Speranza, and Wäscher (2015) consider the Vehicle Routing Problem in which vehicles have compartments of flexible sizes, but with discrete possibilities for the sizes, by defining a fixed step size. All demand for one product type of a customer must be transported in the same vehicle. The application which they consider is the collection of glass, where different colours should be separated. The authors present a problem formulation and a variable neighborhood search. Different neighborhood structures are considered, some related to the kind of products loaded and some more location-related. The method is applied both to randomly generated instances and to instances based on data from practice. In reasonable time good quality solutions are found.
Henke, Speranza, and Wäscher (2017) propose a branch-and-cut algorithm for the same problem, which they tested on instances with up to 50 customer locations. All instances were solved to optimality in two hours.
A slightly different variant of the problem is considered by Koch, Henke, and Wäscher (2016), where the compartment sizes can be arbitrary. They propose a genetic algorithm to solve the problem which showed to give good results with a small optimality gap.

2.4 Multi-Compartment Vehicle Routing Problem with Split Deliveries

The VRP in which the vehicles have multiple flexible compartments and customers are allowed to be visited multiple times, has not yet been studied extensively.
Recently, Coelho and Laporte (2015) made a classification of four different types of Multi-Compartment Delivery Problems in the context of fuel distribution. In these problems

vehicles have multiple compartments of fixed sizes and each compartment can only hold one product type. The customers have several tanks to which different products should be delivered. Tanks at customer locations can either be split or not. When the tanks are considered to be split, the customers may receive that product from different vehicles. Another classification results from the ability to split the content of a compartment over multiple customers or not. Both single-period as well as multi-period routing problems are considered. The authors present formulations and propose a branch-and-cut algorithm to solve the problems, which they tested on randomly generated instances with up to 50 customers. For the single-period problem with split compartments and split tanks, all instances were solved to optimality within reasonable time.
To the best of our knowledge, there is no literature on the Multi-Compartment Vehicle Routing Problem with Split Deliveries (MCVRPSD) with all the additional attributes that we consider in this work. Namely, we study the MCVRPSD in which there is a limited set of possibilities for the compartments in each vehicle and each vehicle may consist of different compartments. The assignment of demand to compartments should be explicitly modeled to ensure feasibility, which is not yet done for the case of limited flexible compartments and split deliveries. Moreover, the splitting possibilities are more restricted in our problem, on which we elaborate in the next section. This holds for both the splitting over the multiple customer visits as the splitting of customer demand over multiple compartments.

3 Problem Description

In this section we elaborate on the specific problem for the retail company we consider in this thesis, which we call the Multi-Compartment Vehicle Routing Problem with Time Windows and Split Deliveries (MCVRPTWSD).

3.1 Vehicle Routing Problem

The general VRP involves the design of routes visiting customers in the most efficient way. The vehicles that are available for the execution of the routes form the set K and have equal capacity Q. The problem is defined on a directed graph $G = (V, A)$, where V denotes the set of vertices and $A = V \times V$ the set of arcs. The vertices consist of customer nodes N and depot node $\hat{d}$, thus $V = N \cup \{\hat{d}\}$. The distance and travel time on arc $(i, j) \in A$ with $i, j \in V$ are denoted by d_{ij} and t_{ij}, respectively. Each customer $i \in N$ requests several suborders, which together form the set X_i. The suborders of all customers form the set $X = \bigcup_{i \in N} X_i$. The size of suborder $x \in X$ is denoted by l_x.
All the suborders must be assigned to vehicles and a sequence of assigned suborders per vehicle must be obtained which specifies the route the vehicle travels. This must be done in such a way that the vehicle capacity Q is not exceeded and all suborders of a customer are in the same route. The relaxation of this last restriction, additional restrictions on the assignment of suborders to the vehicles and other attributes of the problem will be discussed in the next subsections.
The primary objective, as defined by the retail company, is to minimize the total distance travelled by all vehicles. The objective function in this thesis, therefore, minimizes this total distance.

3.2 Time windows

In this thesis we consider the extension of the VRP in which the customer locations have time windows, the VRPTW. Each customer $i \in N$ is assigned a time window $[a_i, b_i]$, which

means that the start time of service at customer i must be after time a_i and before time b_i. If a truck arrives before a_i at the location of customer i, it has to wait until a_i to start service. Each customer location i has a fixed duration of service s_i. Let t_i be the time at which service starts at location i. If customer j is serviced after customer i in a route, this means that $t_i + s_i + t_{ij} \leq b_j$. Furthermore, for customer i it must hold that $t_i \leq b_i$.
Also for the depot, $\hat{d}$, a time window, $[a_{\hat{d}}, b_{\hat{d}}]$ is provided. The vehicles are allowed to leave the depot after time $a_{\hat{d}}$ and must have returned before time $b_{\hat{d}}$. The service time at the depot is denoted by $s_{\hat{d}}$.

3.3 Different products and vehicles with compartments

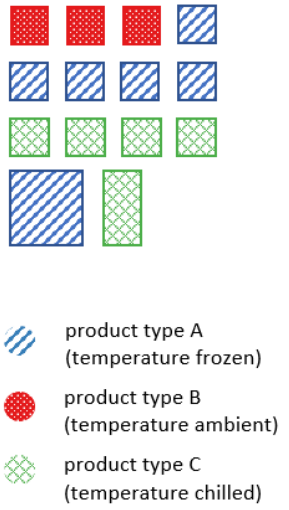
We consider the problem in which different types of products are transported to the customers. Each suborder consists of one product type and a customer may request different suborders of the same product type. In this thesis, we use the term product and product type interchangeably, where we mean a combined group of products such as meat. Let P be the set of different products and p_x the product associated with suborder $x \in X$. Each product $p \in P$ is connected to a temperature z_p from the set of all temperature regimes Z. A suborder containing products of type p, must be transported within temperature regime z_p. For any $\bar{X} \subseteq X$ the set $P(\bar{X})$ consists of all products present in the suborders of $\bar{X}$.
In order to be able to transport products with different temperature regimes, each vehicle $k \in K$ can be equipped with bulkheads, which are used to divide the vehicle into compartments. In the vehicles, the bulkheads can only be placed in some pre-defined positions. This means that the compartments are flexible, but with limited possibilities. We define a set of all possible configurations of compartments in a vehicle, M. Each configuration $m \in M$ specifies a set of compartments C_m and the capacity of the compartments, $q_c, \forall c \in C_m$. Furthermore for each of these compartments a temperature regime is provided, $z_c \in Z, \forall c \in C_m, m \in M$. Note that the same symbol is used to denote the temperature regime of a product and of a compartment, but that from the context can be derived which one is considered. Placement of bulkheads might lead to a decreased total capacity of the vehicle and therefore using configurations with less compartments typically results in larger total capacity of the vehicles. Each route must be assigned a feasible configuration, that is, a configuration in which all products are in a compartment with the right temperature regime and the capacities of the compartments are not exceeded.

3.4 Contamination

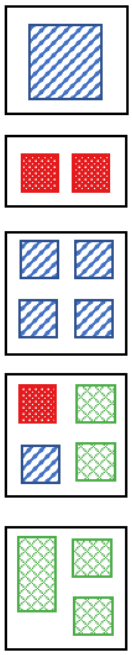
Some products are not allowed in the same compartment because they could contaminate each other, which gives rise to another level of difficulty of the problem. Throughout this thesis, we also refer to contamination as incompatibility between products.
The retail company considered in this thesis, specifies a symmetric contamination matrix. This is a matrix which specifies for each pair of product types, whether they can be transported in the same compartment of a vehicle. The parameter $h_{p\tilde{p}}$ equals one if product types p and $\tilde{p}$ are incompatible, so the products cannot be transported within the same compartment, and zero otherwise.
For the assignment of suborders to a vehicle to be feasible, there should be no contamination for all suborders delivered by that vehicle.

3.5 Orders and multiple customer visits

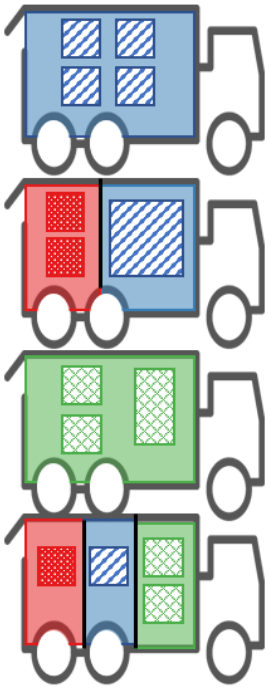
The customers of the retail company are allowed to be visited multiple times for delivering the different suborders. Each customer can request many suborders X_i, which we consolidate into different groups called orders. The set O_i consists of all orders of customer i and the orders of all customers are in the set $O = \bigcup_{i \in N} O_i$. Each order $o \in O$ should be delivered by one vehicle and can be planned independently of the other orders in one of the routes. Because each order is planned in one route, the number of visits to customer i is at most $|O_i|$. The reason that we do not plan all suborders separately, is that the number of suborders that have to be routed is then so large that the routing algorithm, presented in Section 4, would take much longer to obtain good solutions. Moreover, planning a group of suborders together in one route, ensures that the number of visits to the customer is limited.
In Section 6 we specify how the suborders are grouped into orders, but for now it can be assumed that the suborders are somehow combined.
Let X_o be the set of suborders present in order o and let l_o be the corresponding total size of the suborders in the order. The suborders in X_o must be delivered by the same vehicle, but may be transported in different compartments of that vehicle. X_o may, for example, contain suborders consisting of products with different required temperatures, which must be placed in different compartments.
However, it is not allowed to split one suborder over different compartments. A reason for this could be that a suborder corresponds to a pallet or a sealed box, which cannot be taken apart.
In contrast to most split delivery problems studied in the literature, in our problem the total demand of a customer cannot be split in parts of arbitrary size. In fact, each suborder must be transported in its entirety in one vehicle, which makes it more difficult to efficiently fill the vehicles. Additionally, each suborder must be in its entirety in one vehicle compartment.
A schematic overview of the discussed elements can be found in Figure 1 by means of a simplified example. In Figure 1a the 14 suborders requested by a customer are shown. It can be seen that each suborder consists of one product type and multiple suborders of the same product type are requested. The suborders can have different sizes and for each product type a required temperature is specified.
The suborders of this customer are consolidated into five orders in Figure 1b, on which we elaborate in Section 6.
These orders are placed in routes by the routing algorithm, which we present in Section 4, in Figure 1c. Each order is placed in one vehicle only and two different orders may be placed in the same route. The vehicles can be equipped with bulkheads to obtain different compartments of different temperatures. Furthermore, it can be seen that the suborders of one order may be in different compartments in one vehicle, but each suborder is placed in one compartment only.



(a) Input: suborders requested by the customer



(b) Suborders consolidated in orders



(c) Possible assignment to vehicles

Figure 1: Schematic overview of the problem

3.6 Motivation for this study

The problem considered in this work is inspired by a real-life problem tackled by ORTEC for a large retail company. We point out the differences between the problem considered in this thesis and the problem currently tackled by ORTEC.
The major difference is that the methods at ORTEC are not suited for suborders of different sizes that may not be split over multiple compartments. Their methods would not restrict a suborder to be in one compartment only, except if all suborders have equal size and the capacity of each compartment is an integer multiple of this size. If we would want to solve the problem presented in this thesis using the currently implemented methods at ORTEC, this should be corrected manually at the end of the optimization. That is, the suborders that have been split by the algorithm over multiple compartments, should be unassigned. In this thesis we present a solution approach which can handle the different requirements discussed previously.
Additionally, we propose a method that consolidates suborders into orders, which has not yet been developed. The details are discussed in the next sections.

4 Adaptive Large Neighborhood Search

We have separated our solution approach into different parts. In this section, we discuss the elements of our proposed routing methodology framework. At the basis of this framework is the ruin and recreate principle. Ruin and recreate diversifies the search by destroying and then reconstructing a part of the solution. Inspired by Ropke and Pisinger (2006), we apply Adaptive Large Neighborhood Search (ALNS). In ALNS we choose the removal and insertion methods of the ruin and recreate phase in an adaptive manner, based on their previous performance. Pisinger and Ropke (2007) argue that ALNS is particularly appropriate when the problem is tightly constrained, as small neighborhoods might not be sufficient to escape local optima. Therefore, the heuristic is suitable for our problem.
In the remainder of this section we elaborate on a problem specific ALNS. Thereafter, in Section 5, we present the so-called load assignment problem and propose an algorithm to solve it. This load assignment method is used as a feasibility check for assigning an order to a vehicle at any step of modifying a route.

4.1 Overview of the ALNS framework

The ALNS heuristic which is discussed in this section is mainly based on the work of Ropke and Pisinger (2006) and Pisinger and Ropke (2007), with some changes and additions. Algorithm 1 shows the outline of our ALNS algorithm in pseudocode. This algorithm is applied to the orders of customers, which are combinations of suborders. Each order must be inserted in one route, but multiple orders of one customer can be inserted independently of each other. The ALNS algorithm keeps track of the best solution and as a stopping criterion it applies n iterations in total. In line 10 one removal method and one insertion method are selected from the sets of removal and insertion heuristics, R and I, respectively. This selection is based on the weights w assigned to each method. The selected removal heuristic x removes q orders from the solution, which is represented by sequences of orders each of which corresponds to a route. This number, q, is determined uniformly at random from a pre-defined interval. Pisinger and Ropke (2007) argue that a good interval is $\min\{0.1|O|, 30\} \leq q \leq \min\{0.4|O|, 40\}$, with $|O|$ the number of orders. The removed orders are placed in the order bank U and the selected insertion heuristic y tries to insert the orders of U back into the solution. For an order to be placed into a route, there must exist

a feasible load assignment for that route. Therefore, the load assignment algorithm, which is discussed in Section 5, should return a feasible assignment. If this is not the case, that specific insertion is not considered again and we continue with the next possible insertion, until the order bank U is empty or no insertions are feasible. To determine whether or not the obtained new solution is accepted, simulated annealing is used. Given a temperature T, which is updated every iteration, and the current solution s, simulated annealing determines an acceptance probability. A new solution s' with lower distance $f(s')$ is always accepted. The adaptive layer of the ALNS heuristic originates from the fact that the weights are updated after each segment, which consists of a pre-defined number of iterations. The weights are changed based on scores that are assigned to the heuristics during the iterations in the segment and based on the previous weights.
In the next sections, we elaborate further on our specific design of the ALNS heuristic. In Section 4.2 and Section 4.3 we discuss the considered removal heuristics and insertion heuristics, respectively. Thereafter, in Section 4.4 we discuss the method to obtain an initial solution. The acceptance criteria based on simulated annealing and the adaptive layer of the search are discussed in Section 4.5 and Section 4.6, respectively. Finally, in Section 4.7 another form of randomization, namely noise in the objective function, is explained.

Algorithm 1 Adaptive Large Neighborhood Search

```
1:  Input: initial solution s_start
2:  s_best ← s_start
3:  s ← s_start
4:  initialize weights
5:  while stopping criteria not met do
6:      i ← 0
7:      set scores to 0
8:      while i < segment size do
9:          determine q
10:         select x ∈ R and y ∈ I according to w
11:         s' ← x(s)
12:         while orders in order bank that can be inserted do
13:             if feasible load assignment found for y(s') then
14:                 s' ← y(s')
15:         if accept(s,s',T) then
16:             s ← s'
17:         if f(s') ≤ f(s_best) then
18:             s_best ← s'
19:         update scores
20:         update temperature T
21:         i ← i + 1
22:     update
23: return s_best
```

4.2 Removal heuristics

A current solution s is destroyed by removing q orders using one of the removal heuristics in the set R. In total we consider six different removal methods, of which the four are also applied by Pisinger and Ropke (2007) and two are designed specifically for our problem.

Namely, we developed the order related removal heuristic and included a customer removal heuristic.

Distance related removal heuristic The general related removal heuristic first randomly selects a seed order to be removed and then removes orders that are in some way related to this seed order or another already removed order. Algorithm 2 shows the pseudocode of a general related removal heuristic. The set Z contains the removed orders and is expanded in each iteration, until q orders have been removed. L contains the orders that have not yet been removed and is sorted based on the relatedness measure between orders o and $\hat{o}$, $R(o, \hat{o})$. At the front of L are the most related orders, with a smaller relatedness measure. To randomize the search, the algorithm does not always select the most related order to add to Z. Parameter $p \geq 1$ controls the amount of randomness. A higher value for p contributes to selecting the more related orders at the front of the list, whereas lower values for p direct the algorithm to selecting less related orders.

Algorithm 2 Related removal heuristic

Input: solution s, parameter q, parameter p
$Z \leftarrow$ random order from s
while $|Z| < q$ **do**
    select order $o \in Z$ uniformly at random
    array $L \leftarrow$ orders from s not in Z
    sort L according to relatedness to o:
        $i < j \iff R(L[i], o) < R(L[j], o)$
    $y \leftarrow$ random number in interval $[0, 1)$
    index $l \leftarrow \lfloor y^p |L| \rfloor$
    $Z \leftarrow Z \cup L[l]$
remove orders in Z from their routes in s and put in U

In the distance related removal heuristic, orders are said to be related when the distance between the customer locations is small. The distance relatedness measure $R^d(o, \hat{o})$ between orders $o \in O_i$ and $\hat{o} \in O_j$ is as follows.

$$R^d(o, \hat{o}) = d_{ij} \tag{1}$$

where d_{ij} is the distance between customer i and customer j.
We apply a distance related removal heuristic, because for customers located close to each other it is more likely that the orders can be interchanged to obtain a different solution.

Time related removal heuristic The time related removal heuristic is similar to the distance related removal heuristic. However, instead of measuring relatedness with the distance between customers, relatedness is defined in terms of difference in start time of the service at the customer locations. The idea behind this heuristic, is that orders served at a similar time are more likely to be interchangeable. As Pisinger and Ropke (2007) have found that it is beneficial to also take distance into account, only the τ orders closest to the seed order are considered. Of those τ closest orders, the $q - 1$ orders most related to the seed order in terms of time are then also removed from the solution. As for the distance related removal heuristic, we also apply randomization with the parameter p. Let t_o denote the start time of service for order $o \in O_i$ at customer i.

The time relatedness measure $R^t(o, \hat{o})$ is the following.

$$R^t(o, \hat{o}) = |t_o - t_{\hat{o}}| \tag{2}$$

Pseudocode for the time related removal heuristic is shown in Algorithm 3.

Algorithm 3 Time related removal heuristic

Input: solution s, parameter q, parameter p, parameter τ
$o \leftarrow$ random order from s
$Z \leftarrow o$
array $D \leftarrow$ orders from s except o
sort D according to distance relatedness to o:
    $i < j \iff R^d(D[i], o) < R^d(D[j], o)$
array $L \leftarrow D[0] \cdots D[\tau - 1]$
sort L according to time relatedness to o:
    $i < j \iff R^t(L[i], o) < R^t(L[j], o)$
while $|Z| < q$ **do**
    $y \leftarrow$ random number in interval $[0, 1)$
    index $l \leftarrow \lfloor y^p |L| \rfloor$
    $Z \leftarrow Z \cup L[l]$
    reduce L by removing $L[l]$
remove orders in Z from their routes in s and put in U

Random removal heuristic The random removal heuristic is a simple removal heuristic which removes q orders from the solution uniformly at random.

Worst removal heuristic The worst removal heuristic removes expensive orders from their routes. The cost $c(o, s)$ of an order o is defined as the difference between the objective value of the current solution s and the objective value of the solution without the order o. An expensive order is an order for which removing it from the solution results in a much lower objective value. Until q orders have been removed, the heuristics removes the order o with largest cost $c(o, s)$ from the solution in each iteration. Again, we apply randomization which is controlled by the parameter p_{worst}. The pseudocode for the worst removal heuristic can be found in Algorithm 4. The orders with the highest costs are at the front of array L and are, thus, more likely to be removed from the solution.

Algorithm 4 Worst removal heuristic

Input: solution s, parameter q, parameter p_{worst}
$Z \leftarrow \emptyset$
while $|Z| < q$ **do**
    array $L \leftarrow$ orders from s not in Z
    sort L:
        $i < j \iff c(L[i], s) > c(L[j], s)$
    $y \leftarrow$ random number in interval $[0, 1)$
    index $l \leftarrow \lfloor y^{p_{\text{worst}}} |L| \rfloor$
    $Z \leftarrow Z \cup L[l]$
    remove order $L[l]$ from s

Order related removal heuristic The above mentioned removal heuristics do not take into account the fact that orders might not be interchangeable if they consist of different products. For example, two orders of which one mainly consists of frozen goods and the other mainly consists of ambient goods are not likely to fit in each others vehicles if the vehicles do not have the right compartments. The order related removal heuristic is a variant of the related removal heuristic in which relatedness is based on the required temperatures for the orders. Furthermore, it takes into account the sizes of the suborders for which the required temperatures match. The orders are less related if one of the temperatures of order o is not required for any of the suborders of order $\hat{o}$ and when the total amount of order o that requires that specific temperature is larger. Let Z_o be the set with the required temperatures for at least one of the suborders in o. The order relatedness measure is the following.

$$R^o(o,\hat{o}) = \frac{1}{|Z_o \cup Z_{\hat{o}}|} \sum_{z \in Z_o \cup Z_{\hat{o}}} \frac{|\sum_{x \in X_o} l_x \mathbb{1}_{\{z_{p_x}=z\}} - \sum_{\hat{x} \in X_{\hat{o}}} l_{\hat{x}} \mathbb{1}_{\{z_{p_{\hat{x}}}=z\}}|}{\max\{\sum_{x \in X_o} l_x \mathbb{1}_{\{z_{p_x}=z\}}, \sum_{\hat{x} \in X_{\hat{o}}} l_{\hat{x}} \mathbb{1}_{\{z_{p_{\hat{x}}}=z\}}\}} \tag{3}$$

For each temperature which is required for at least one of the orders o or $\hat{o}$, a value between 0 and 1 is added to the relatedness measure. The value 1 is added when a temperature regime is not required by one of the orders and the value 0 is added when both orders require exactly the same space with a specific temperature regime. The sum is then scaled by the number of considered temperature regimes.
For the same reason as with the time related removal heuristic, we take into account the distance between the orders. We therefore apply Algorithm 3 where we replace $R^t(o,\hat{o})$ by $R^o(o,\hat{o})$.

Customer removal heuristic We also include a removal heuristic which removes all orders of one customer. In each iteration a customer is chosen randomly, for which all orders are removed from their routes, until in total q orders have been removed.

4.3 Insertion heuristics

Basic greedy heuristic The basis greedy heuristic iteratively inserts the order that is cheapest to insert in its cheapest insertion place. The cheapest insertion corresponds to the insertion for which the objective value increases the least. For each route k, the cheapest insertion position for each order o is determined and the corresponding costs are denoted by Δf_{ok}. When insertion in the route is not feasible for that order based on the capacity or time window restrictions, we set $\Delta f_{ok} = \infty$. The algorithm inserts order o in vehicle k in the cheapest position for which

$$(o,k) = \underset{o \in U, k \in K}{\arg\min} \Delta f_{ok}, \tag{4}$$

where U is the set of unplanned orders and K the set of vehicles.

Regret heuristic With the basic greedy heuristic, it is possible that the more difficult orders are saved for last, making it very costly or impossible to insert them. The regret heuristic anticipates on this problem by taking into account future insertion costs when the cheapest insertion routes are not possible anymore, due to previous insertions of other orders.
Let Δf_o^r be the cost of inserting order o in the r-th best route in its best position. Note

that the basic greedy heuristic inserts the order o which minimizes $\Delta f_o^1 = \min_{k \in K} \Delta f_{ok}$, where the superscript 1 indicates that it corresponds to the cheapest vehicle for order o. The regret-r heuristic inserts the order for which the total regret, the costs of inserting the order in the second best route up to the r-th best route instead of the best route, is the largest. The order o, for which the regret is the largest, is inserted it the cheapest route at its best position. The order o with the largest regret is, thus, determined as follows

$$o = \arg\max_{o \in U} \left(\sum_{i=2}^{r} \left(\Delta f_o^i - \Delta f_o^1 \right) \right). \tag{5}$$

Ties are broken by inserting the order with lowest insertion costs.
We consider the regret-r heuristics with $r = \{2, 3, 4, |K|\}$.

4.4 Initial solution

We use the basic greedy heuristic to obtain an initial feasible solution. This method starts with all customer orders in the order bank U and stops when the order bank U is empty or no more order can be placed in any of the routes.

4.5 Acceptance criteria

Derigs et al. (2011) have investigated various metaheuristics for the MCVRP and have shown that the difference in solution quality between the metaheuristics is not large. However, the considered metaheuristics performed better than when simply accepting all solutions or accepting all improved solutions. Therefore, following Ropke and Pisinger (2006) and Pisinger and Ropke (2007), we use Simulated Annealing (SA) to govern the search process. In SA a better solution is always accepted and a worse solution is accepted with a certain probability. This probability depends on the increase in objective value of the new solution, s', compared to the previous one, s, and on a so-called temperature, $T > 0$. This temperature decreases over the iterations, leading to smaller probabilities of accepting worse solutions. Solution s' is accepted with probability

$$e^{\frac{-(f(s')-f(s))}{T}}, \tag{6}$$

where $f(s')$ and $f(s)$ denote the objective values of the new and current solution, respectively. The algorithm starts with temperature T_{start} based on the initial solution and the temperature is updated in each iteration according to $T = T \times c$, where $0 < c < 1$ is the cooling rate. We choose T_{start} such that a solution in which $w\%$ more distance must be travelled than in the initial solution, is accepted with probability 0.5.
Pisinger and Ropke (2007) propose to divide T_{start} by the number of orders in the instance, such that the method is able to cope better with instances of different sizes. We follow this approach.

4.6 Adaptive layer

Ropke and Pisinger (2006) presented ALNS, as variant of LNS in which the removal an insertion methods are chosen with adaptive probabilities. These probabilities are modified based on the performance in previous iterations. As this method has shown to give good results, we follow their idea and apply this so-called roulette wheel principle. Pisinger and Ropke (2007) argue that an additional advantage of this method is that the calibration of the algorithm is limited due to the roulette wheel which determines automatically the

influence of each removal and insertion method.
The selection of an insertion heuristic is independent of the selection of a removal heuristic. Therefore, the two types of heuristics have a separate roulette wheel. The iterations of the ALNS heuristic are divided into segments of 100 iterations. The algorithm uses the same weights, which determine the probability of selecting a heuristic, for the iterations inside one segment. Let H be the set of heuristics, either insertion or removal heuristics. Then heuristic $\hat{h}$ is chosen in segment i with probability

$$\frac{w_{\hat{h},i}}{\sum_{h \in H} w_{h,i}}, \tag{7}$$

where $w_{h,i}$ denotes the weight of heuristic h in segment i. The weights are initially equal for all heuristics in the set H, summing up to 1. At the end of each segment, the weights are updated, based on the scores collected in the segment and on the previous weight as follows

$$w_{h,i+1} = (1 - \rho) w_{h,i} + \rho \frac{\zeta_{h,i}}{\tau_{h,i}}. \tag{8}$$

Here, $\tau_{h,i}$ is the number of times heuristic h is selected in segment i and $\zeta_{h,i}$ is the corresponding score. The parameter ρ is called the reaction factor and controls how much the scores influence the new weights. The scores are initially 0 at the start of the segment and are updated in iterations where the heuristic is used. The increase in score depends on the solution s' obtained in that iteration, as follows

$$\zeta_h = \zeta_h + \begin{cases} \sigma_1 & \text{if new global best solution} \\ \sigma_2 & \text{if better than current solution, not accepted before} \\ \sigma_3 & \text{if worse than current solution, accepted now but not accepted before} \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

The value for σ_1 is much higher than the values for σ_2 and σ_3, because new global best solutions should be rewarded the most. Only when solutions have not been accepted in previous iterations, the scores are incremented, in order to diversify the search. As in every iteration an insertion and a removal heuristic are applied, both of their scores are updated equally.
The algorithm keeps track of already accepted solutions by assigning a hash code representing the solution.

4.7 Noise term in the objective function

We also apply the roulette wheel principle for deciding whether or not noise is added to the objective value, to randomize the search even more. The same method as in Section 4.6 is used, where the two choices are either to apply noise or not, independent of the selected insertion heuristic. The scores for applying noise are, therefore, updated whenever one of the insertion methods uses the version with noise. For not applying noise, this works similarly. The amount of noise X^n that is added is determined uniformly at random from the interval $[-X, X]$, with

$$X = \eta \, max_{i,j \in V} d_{ij}. \tag{10}$$

Here, η is the parameter that controls the amount of noise. Now the algorithm changes the insertion costs C to the modified insertion costs

$$C' = \max\{0, C + X^n\}. \tag{11}$$

5 Load assignment

In the previous section, we discussed several insertion heuristics that are applied in the ALNS framework. Each insertion heuristic does not only take into account the increase in distance when inserting an order inside a vehicle, but also checks whether or not the insertion is feasible in terms of capacity and time window violation.
In the problem considered in this thesis, each order consists of several suborders. Each suborder consists of one product type, but the suborders in one order could consist of different product types. Due to different required temperatures for these products and due to incompatibility between some of the products, inserting an order inside a vehicle requires checking the restrictions that are attached to these requirements.
The load assignment problem is defined as the problem of finding a feasible configuration $m \in M$ of compartments for the vehicle $k \in K$ in which the insertion heuristic tries to insert an order which is put in the order bank U, $o \in U$. Finding such a feasible configuration m requires finding an assignment of both the suborders X_o as well as the suborders that were inserted previously in k, X_k, in one of the compartments in C_m with the right temperature regime and with no contaminating products in the same compartment.
Because the load assignment problem itself is more difficult than simply checking the capacity or time windows restrictions, the load assignment check is only applied after the insertion heuristic has determined which order o is best to insert in which vehicle k. Figure 2 shows schematically when the load assignment algorithm is applied inside the ALNS framework. In the figure, the initial solution is assumed to be a solution for which a feasible load assignment exists. The initial solution is, in fact, constructed by applying the basic greedy insertion heuristic and solving the load assignment problem for each insertion to check the feasibility.

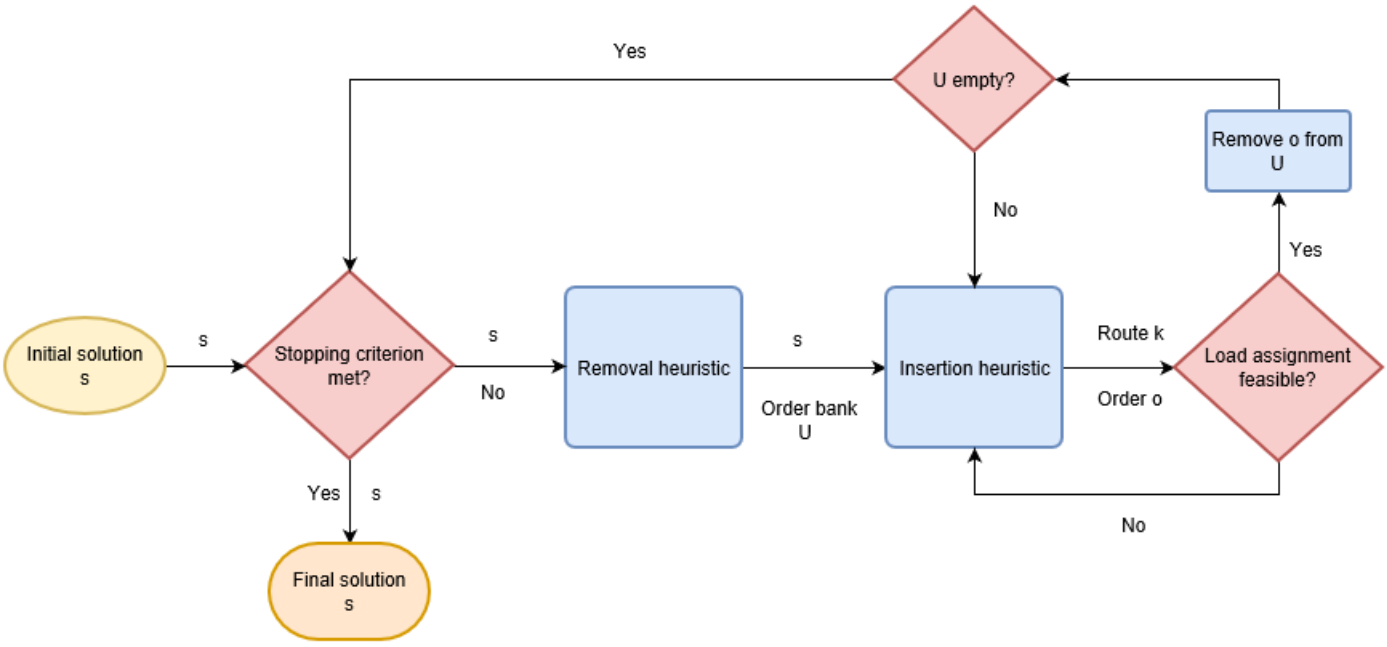


Figure 2: Schematic overview of the load assignment inside the ALNS framework

5.1 Division of the load assignment problem into subproblems

First of all, we notice that the problem can be split per temperature regime z, as each compartment can only accommodate suborders with one required temperature. Moreover, note that the sub-problem for a temperature z is simple when the configuration only contains one compartment with temperature regime z. We elaborate on these statements.
Let Z_o and Z_k be the sets with the required temperatures for the suborders of order o that has to be inserted and the suborders X_k that are already in the vehicle, respectively. Furthermore, let the suborders of X_o and X_k with required temperature z be denoted by X_o^z and X_k^z, respectively, and the compartments from C_m with temperature regime z by C_m^z. If for all temperatures z in $Z_o \cup Z_k$ a feasible assignment of the suborders in $X_o^z \cup X_k^z$ is found to one of the compartments of the considered configuration, the configuration is feasible for this insertion.
When the configuration only contains one compartment with temperature regime $z \in Z_o \cup Z_k$, all suborders with required temperature z should be inserted in the same compartment. To check if this is feasible, we need to check if the capacity of this compartment is large enough to accommodate the suborders $X_o^z \cup X_k^z$ and whether or not any of these suborders are incompatible with each other.
Additionally, we notice that we can apply some prechecks for the considered configuration before running the actual load assignment algorithm. We check whether or not the total capacity available per temperature regime is sufficient to accommodate all suborders with that required temperature. The algorithm also keeps track of the minimum number of compartments needed, due to contamination restrictions that were violated for previously considered configurations. So we also include a precheck which checks for each temperature regime if the configuration contains at least this minimum number of compartments.
The configurations are inspected in a route-specific order, on which we will elaborate in Section 5.3. Before considering all possible configurations, we start by checking feasibility for the configuration which was assigned to vehicle k in a previous ALNS iteration. We do so, because this configuration was the latest feasible configuration for this vehicle and we know that X_k can all be accommodated using this configuration. We do not want to limit ourselves by fixing the previous assignment of the suborders in X_k to their compartments, so we allow these suborders to be assigned to a different compartment in this iteration.
We define the load assignment sub-problem as the problem of finding an assignment of $X_o^z \cup X_k^z$ to the multiple compartments in C_m^z for one specific configuration m and temperature z. In Figure 3, we have placed the sub-problem in the context of the so-far discussed elements of the load assignment methodology. In the next section we refer to the suborders $X_o^z \cup X_k^z$ in this sub-problem as V and to the compartments C_m^z as C.

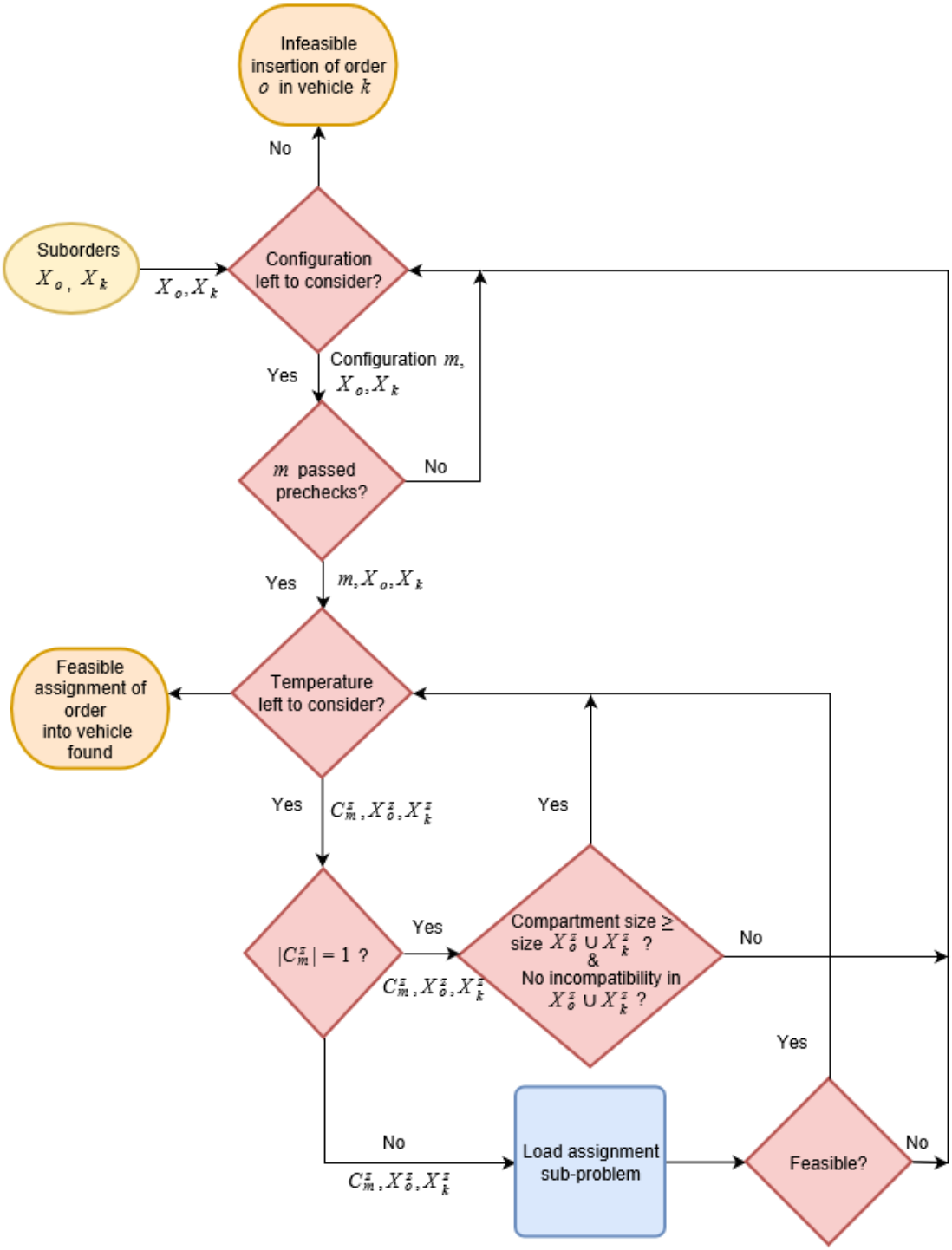


Figure 3: General outline of the load assignment methodology

5.2 Solution approach for the load assignment sub-problem

5.2.1 Motivation for the heuristic

Solving the load assignment sub-problem comes down to checking for the considered configuration whether or not a feasible assignment of the suborders can be made with the

compartments specified by that configuration. This problem has a lot of similarity with well known problems in the literature and is, in fact, a generalization of the classical Bin-Packing Problem (BPP) (see for example Johnson and Garey (1985)). The decision version of the BPP, in which must be decided if the items fit into a specified number of bins, is NP-complete, as the known NP-complete Partition Problem can be reduced to the BPP and its solutions can be verified in polynomial time. The BPP is a special case of the load assignment sub-problem in which the bins are compartments with equal capacity. The items are suborders that are all compatible with each other.

The BPP is also very much related to the extensively studied Cutting Stock Problem, in which stock material should be cut into pieces of specified sizes while minimizing the waste of material (see Sweeney and Paternoster (1992) for a review).

Another variant of the BPP which is closely related to our load assignment sub-problem, is the Variable Sized Bin-Packing Problem (VSBPP) (Haouari & Serairi, 2009). In this problem, we are given a set of different types of bins which different capacities and costs and the objective is to pack all items in a set of bins with minimum costs. However, there is no incompatibility between the items.

Gendreau, Laporte, and Semet (2004) present lower-bounds and heuristics for the Bin-Packing Problem with Conflicts (BPPC), which is a variant of the BPP in which items can be conflicting. In fact, the BPPC can be seen as a combination of the BPP and of the Vertex Coloring Problem (VCP) (Muritiba et al., 2010). In the VCP each vertex in the graph must be assigned a color, such that adjacent vertices do not have the same color and the number of colors used is minimized (Malaguti et al., 2008). Due to the complexity of coloring problems, these problems are often studied on specific graph classes.

Jansen (1999) proposes an asymptotic approximation scheme for the BPPC which is only suited for specific conflict graph types, such as trees and grid graphs. Their algorithm combines the approximation scheme of Karmarkar and Karp (1982) to obtain a solution for BPP without conflicts and an approximation algorithm for the coloring problem, providing a limited number of additional bins required to remove the conflicts.

Epstein, Favrholdt, and Levin (2011) present the Online Variable Sized Bin-Packing Problem with Conflicts, in which the bins of different sizes arrive one by one and in each arriving bin an item has to be placed. This problem has its application in scheduling jobs on different processors, where due to security reasons some pairs of jobs cannot be processed by a common processor. The fact that it is an online problem, makes that it is different from the load assignment sub-problem. The authors focus on the analysis of the asymptotic competitive ratio, but show that no competitive algorithm can be found for the online problem. In contrast to their problem, in our offline problem we can make use of the fact that we can take into account the different sizes of the compartments in advance.

More recently, the offline version of the Variable Sized Bin-Packing Problem with Conflicts is addressed in Maiza, Radjef, and Sais (2016). The authors propose lower bounds for the problem, which can be obtained by solving a proposed mathematical programming formulation. However, the authors consider infinite available bins for each bin type, in contrast to the fixed number of compartments in our problem. We believe that the presented lower bound does, therefore, not give us so much insight. Furthermore, we focus on developing an algorithm to actually find feasible solutions.

Gendreau et al. (2004) present six algorithms for the BPPC, of which the one which makes use of conflict and non-conflict cliques provides good results. Therefore, inspired by this method, we develop a heuristic which is specifically suited for the load assignment sub-problem and makes use of this concept of cliques. In the remainder of Section 5.2, we explain this concept and propose our heuristic.

5.2.2 Conflict graphs, non-conflict graphs and cliques

Throughout the heuristic, which we present in Section 5.2.3, we make use of conflict graphs and non-conflict graphs. In both types of graphs the vertices represent the product types. The conflict graph has edges between incompatible product types, whereas in the non-conflict graph the edges connect compatible product types. In these conflict and non-conflict graphs, we make use of the concept of cliques. A clique is a subset of vertices in an undirected graph, such that the sub-graph induced by the clique is a complete graph. That is, every two distinct nodes in a clique are connected.
We explain now the general idea of the usage of these cliques, and we elaborate more on this in the next paragraphs. The algorithm developed in this work uses a large conflict clique that contains products that require each their own compartment in the vehicle and can be seen as the most difficult products, because they contaminate with many other products. For each of these difficult products, one non-conflict clique is used to determine which products could be inserted in the same compartment. In each iteration of the heuristic, the suborders corresponding to the products of one of these non-conflict cliques can be inserted at once in empty compartments, without having to take into account the contamination.

Conflict clique We use this concept of cliques first to identify mutually incompatible suborders in the set of all suborders considered in the sub-problem, V. In other words, V contains the suborders from order o with a required temperature z and the already present suborders in vehicle k with required temperature z. Let $P(V)$ be the set of product types present in V. To identify these incompatible suborders, we solve the maximum clique problem on the conflict graph of the sub-problem. None of the product types that are present in the so-called conflict clique, obtained by solving the maximum clique problem, can be placed in the same compartment due to contamination. Therefore, this clique, the largest group of product types that are mutually incompatible, also provides us with a lower-bound on the number of compartments that are needed with temperature regime z.
The maximum clique problem is computationally equivalent to the maximum independent set problem and the minimum vertex cover problem, problems that are all known to be NP-complete (Bomze et al., 1999). The maximum clique problem on specific classes of graphs, such as perfect graphs, is polynomially solvable. However, we do not want to restrict ourselves to graphs with a special structure.
Johnson (1974) proposes a heuristic and evaluates its worst case behaviour, the ratio of the optimal value to the worst solution value. The algorithm can be implemented in time $\mathcal{O}(n \log n)$, but it is shown with an example that the heuristic performs bad in worst case. Nevertheless, following Gendreau et al. (2004), we apply an adaptation of this heuristic as it provides very good results in less extreme examples. As we need to make many cliques for each insertion in each ALNS iteration, we choose for a faster approach by using a heuristic. Moreover, as will become clear after the overall load assignment heuristic is presented, not finding the maximum clique does not necessarily result in failure nor in larger probability of failure of the load assignment algorithm.

Algorithm 5 Heuristic to obtain conflict clique

```
1: function obtainConflictClique (V)
2:     conflict clique D ← ∅
3:     R ← P(V)
4:     while |R| > 0 do
5:         let R' ⊆ R be the maximum degree products in the conflict graph induced by R
6:         let y ∈ R' be the product with the largest number of suborders in the set V
7:         D ← D ∪ {y}
8:         remove y and the products that are not connected to y from R
9:     return D
```

The heuristic is shown in Algorithm 5. The algorithm starts with an empty conflict clique D and all product types that are present in the graph are placed in a set R. In each iteration of the algorithm, a product type y is transferred from R to D. Product type y is chosen based on that it has the largest degree in the conflict graph induced by R, in order to have more possible product types to transfer from R to D in the next iterations. After adding product type y to the clique, all product types that are not connected to y should be removed from R. For the selection of product type y, ties are broken by selecting the product for which there are the most suborders in V. This is because we believe it is beneficial for the conflict clique to contain the most difficult product types, which is not only defined by the product types which have the most contamination with the other products. As a measure of this extra difficulty of a product type we take the number of suborders for this type, because there could be less possibilities for inserting more suborders together. Note that the products are added to the conflict clique in order of difficulty.

Non-conflict cliques For each product p in the conflict clique D, Algorithm 6 is used to detect compatible product types in a non-conflict clique. These non-conflict cliques should together contain as many products as possible. If some products could be in multiple cliques, these cliques could contain partially the same products. This would be the case if there are products that do not contaminate many other products. More products in the conflict cliques could result in more flexibility for the insertion of the cliques. We make this flexibility concrete, by separating the overlapping and non-overlapping products in the non-conflict cliques. The overlapping products are products that can be found in multiple non-conflict cliques. We use this classification to prioritize the different suborders for insertion, on which we elaborate in the next paragraph.

Algorithm 6 Heuristic to obtain non-conflict clique

```
1: function obtainNonConflictClique (V, product p ∈ D, products W )
2:     non-conflict clique D_p ← {p}
3:     R ← P(V)
4:     remove p and the products that are not connected to p from R
5:     while |R| > 0 do
6:         let R' ⊆ R be maximum degree products in the non-conflict graph induced by R
7:         let R'' ← R' \ W
8:         if |R''| > 0 then
9:             let y ∈ R'' be the product with the largest number of suborders in the set V
10:        else
11:            let y ∈ R' be the product with the largest number of suborders in the set V
12:        D_p ← D_p ∪ {y}
13:        remove y and the products that are not connected to y from R
14:        add y to W
15:    return D_p
```

The heuristic, which can be found in Algorithm 6, is similar to Algorithm 5, but with minor differences. First of all, the algorithm starts by adding p itself to the non-conflict clique. Furthermore, the set W contains the products that have already been assigned to a previously constructed non-conflict clique. We illustrate this with an example. Suppose that D contains two products p_1 and p_2, where p_1 is added first and p_2 second. First, the algorithm considers $W = \emptyset$ and creates non-conflict clique D_{p_1}. Then for constructing the second clique D_{p_2} the set W contains all products in D_{p_1}. This set W is used to break ties when selecting product y, before looking at the number of suborders present in V for that product. We first select the products that are not yet assigned to a non-conflict clique, such that we have larger probability of having all products in one of the non-conflict cliques.

5.2.3 Heuristic

The general framework of the heuristic can be found in Algorithm 7. It takes as an input C, the compartments from a configuration m with temperature regime z, and V, the suborders from $X_o^z \cup X_k^z$. A feasible assignment is found when all suborders in V have been assigned to a compartment.
In the first part of the algorithm the cliques are made as explained in Section 5.2.2. Each non-conflict clique D_p is first translated to a non-conflict clique of suborders, D'_p, which is added to the list L. Then the overlapping suborders $o(D'_p)$ are separated, which is then used in the matching procedure.

Algorithm 7 Load assignment with multiple compartments

Input: compartments C, suborders V
while $|V| > 0$ **do**
    conflict clique $D \leftarrow$ obtainConflictClique(V)
    non-conflict cliques $L \leftarrow \emptyset$
    products in a non-conflict clique $W \leftarrow \emptyset$
    for each product $p \in D$ **do**
        $D_p \leftarrow$ obtainNonConflictClique(V, p,W)
        $D'_p \leftarrow$ suborders of V corresponding to D_p
        $L \leftarrow L \cup D'_p$
    split $\forall D'_p \in L$ into overlapping, $o(D'_p)$, and non-overlapping, $D'_p \setminus o(D'_p)$, suborders
    while $|L| > 0$ **do**
        inserted $\leftarrow$ matchingProcedure(C,L)
        if not inserted **then**
            for $y = 2$ up to $\lfloor \frac{|C|}{|L|} \rfloor$ **do**
                $C^d \leftarrow$ all possible combinations of y different compartments from C
                inserted $\leftarrow$ matchingProcedure(C^d,L)
        if inserted **then**
            remove from V the inserted suborders
            update classification of overlapping and non-overlapping suborders
        else
            return configuration is not feasible
return feasible assignment found

The matching procedure can be found in Algorithm 8. This procedure inserts, if feasible, the non-overlapping suborders of the best fit clique in the best fit compartment. The best fit corresponds to the least unused capacity when inserting the non-overlapping suborders of the non-conflict clique in the compartment. Ties are broken by selecting the clique and compartment for which the overlapping orders also fit the best. If possible it also inserts the overlapping suborders in the same compartment. The overlapping suborders are easier to insert, because they can be inserted with multiple non-conflict cliques, and are therefore left for last.

It is possible that multiple compartments are needed to insert a clique. In this case, dummy compartments C^d are created. These dummy compartments are not actual compartments, but they represent a combination of y compartments. The size of a dummy compartment is equal to the total size of the compartments it represents and the dummy compartments are used to determine the best match in the matching procedure. All $\binom{|C|}{y}$ combinations of the compartments are considered. These dummy compartments are considered as one compartment in size, but for insertion the Best Fit Decreasing (BFD) heuristic is used with the actual compartments. This means that finding a dummy compartment in which the non-overlapping suborders of a clique fit due to enough total capacity, does not necessarily mean that the suborders actually fit in their corresponding compartments, because suborders are not allowed to be split. The size of the suborders might make this impossible. If this is the case the matching procedure tries to find a different match.

The dummy compartments consisting of y actual compartments are considered only if less than y compartments were not enough for any of the non-conflict cliques still present in L. Therefore y can be at most $\left\lfloor \frac{|C|}{|L|} \right\rfloor$, for each non-conflict clique to be inserted.

After inserting (a part of) the suborders in a non-conflict clique, this clique is removed from

L and the used compartments are out of consideration for next insertions. The classification of overlapping and non-overlapping suborders should also be revised. If there are no non-conflict cliques left in L, but V is not empty, the procedure starts over with the remaining suborders and compartments (in line 2 in Algorithm 7).
Based on the number of available compartments and the number of products in the conflict clique, by applying some checks in between the steps, the algorithm determines quicker if the configuration is not feasible. These checks are left out of Algorithm 7 for simplicity.

Algorithm 8 Match non-conflict clique and compartments

```
1:  function matchingProcedure(C, non-conflict cliques L)
2:      s_* ← ∞
3:      for each D'_p ∈ L do
4:          let c_p ∈ C be the best fit compartment for D'_p \ o(D'_p) and s_p the slack in capacity
5:          if s_p ≥ 0 and (s_p < s_* or s_p = s_* but o(D'_p) fits better) then
6:              s_* ← s_p
7:              (c_*, D'_*) ← (c_p, D'_p)
8:      if s_* < ∞ then
9:          if c_* is a dummy compartment, with corresponding compartments C_* then
10:             assign suborders from D'_* \ o(D'_*) to C_* using BFD if they fit
11:             if not all suborders assigned in line 10 then
12:                 restart line 2, where (c_*, D'_*) is no longer an option
13:             assign suborders from o(D'_*) to C_* using the BFD heuristic if they fit
14:         else
15:             assign suborders from D'_* \ o(D'_*) to c_* in decreasing order
16:             assign suborders from o(D'_*) in decreasing order if they fit
17:         L ← L \ D'_*
18:         remove c_* or its corresponding compartments from C
19:         return true
20:     else
21:         return false
```

5.2.4 Comparison with Integer Linear Programming

To evaluate the performance and solution quality of the presented load assignment heuristic, we also present an Integer Linear Programming (ILP) problem formulation.
We compare the final routing solution obtained by solving the ILP using CPLEX with the routing solution obtained by using our load assignment heuristic for the sub-problem. This helps us to quantify the possible gain of applying an exact method to solve the load assignment sub-problem, but mainly to evaluate the boundaries in terms of runtime of this exact solution approach.
Additionally, solving the ILP in an exact manner helps us to identify wrong results by the load assignment heuristic. When no feasible configuration is found for the load assignment problem in which the sub-problems are solved using our proposed heuristic, we check if this load assignment problem can be solved using CPLEX in the sub-problems.
We first state the relevant parameters and the decision variable for the formulation. Let the parameters l_i and q_j be the size of suborder $i \in V$ and the capacity of compartment $j \in C$, respectively. Parameter $h_{p_i p_k}$ $\forall i, k \in V$ is one if products p_i and p_k contaminate and zero otherwise. The binary decision variable x_{ij} $\forall i \in V, j \in C$ equals one if suborder i is assigned to compartment j and zero otherwise.

The load assignment sub-problem can now be formulated as

$$\begin{aligned}
\min \quad & 0 && && (12)\\
\text{s.t.} \quad & \sum_{j \in C} x_{ij} = 1, && \forall i \in V, && (13)\\
& \sum_{i \in V} l_i x_{ij} \leq q_j, && \forall j \in C, && (14)\\
& x_{ij} + x_{kj} \leq 2 - h_{p_i p_k}, && \forall i, k \in V, j \in C, && (15)\\
& x_{ij} \in \mathbb{B} && \forall i \in V, j \in C. && (16)
\end{aligned}$$

Here, the objective is to find an arbitrary feasible solution, which can be modeled by minimizing any arbitrary value. To make sure that each suborder is assigned to a compartment in a feasible solution, Constraints (13) are added. Constraints (14) ensure that the compartment capacities are not exceeded. Constraints (15) guarantee that any two suborders that consist of contaminating products cannot be placed in the same compartment.

5.3 Configuration sorting

As explained earlier, we consider the configurations in an order that is different for every vehicle. The n iterations of the routing algorithm are divided into segments of κ iterations. At the end of each segment, the configurations are sorted using a score which is based on the performance of each configuration in that segment and on the suborders that are currently in the vehicle. The most promising configurations have a lower score and are therefore at the top of the list. Let X_k be the set of suborders that are currently present in vehicle k and Z_k the corresponding set of temperatures required for the suborders in X_k. Furthermore, let θ_{km} be the number of times the load assignment algorithm tried to assign suborders to vehicle k using configuration m and let π_{km} be the number of times this succeeded. Equation 17 shows how the score σ_{km} is calculated in each segment for vehicle k and configuration m.

$$\sigma_{km} = \alpha \frac{\theta_{km} - \pi_{km}}{\theta_{km}} + (1-\alpha) \frac{1}{|Z_k|} \sum_{z \in Z_k} \frac{\max\{\sum_{x \in X_k} l_x \mathbb{1}_{\{z_{p_x} = z\}} - \sum_{c \in C_m} q_c \mathbb{1}_{\{z_c = z\}}, 0\}}{\sum_{x \in X_k} l_x \mathbb{1}_{\{z_{p_x} = z\}}} \quad (17)$$

The first term, of which its influence is controlled by parameter α, measures the fraction of previous failures of the configuration. The second term, which contributes with $1 - \alpha$ to the score, increases when the compartments of the configuration cannot fully accommodate the suborders that are present in the vehicle. Both terms have a value in the interval $[0, 1]$. The second term measures per temperature the fraction of total size of the suborders for which the compartments do not have enough space. The measure is summed over the temperatures and scaled by the number of considered temperatures.
Adding this sorting to the heuristic could lead to finding a feasible load assignment in each iteration faster. On the other hand, the total runtime would only improve if this gain outweighs the increase in runtime by sorting the configurations every κ iterations. A higher value for κ decreases the total time of sorting the configurations per vehicle, but also takes away the advantage of the sorting, as the suborders assigned to the vehicle could have changed a lot. The value for κ should be determined by taking into account this trade-off.

6 Consolidation of suborders into orders

6.1 Different variants of input for the ALNS

As explained in Section 3.5 and as can be seen from Figure 1, suborders are consolidated into orders. These orders are the input for the ALNS, in which each order is inserted in one vehicle by an insertion heuristic and the whole order is removed from the route by a removal heuristic. In this section we discuss the different variants of input for the ALNS.

No consolidation The reason we do not let each suborder in X_i be an order and thus input for the routing algorithm, is that this might result in much higher runtime as there are many more suborders than orders. Moreover, the software provided by ORTEC also routes groups of consolidated suborders for the retail company, so we do not want to deviate much from their approach. Furthermore, we want to limit the number of visits to the customer. If each suborder would be an order, the number of visits to the customer could in worst case be equal to X_i. This is also tested in Section 7.

Consolidation and planning per product type Without a load assignment algorithm, suborders of different product types cannot be planned together without violating the temperature and contamination constraints, as the routing algorithm itself assigns orders without restricting the different product types present in the orders. To avoid these violations when planning without a load assignment algorithm, the customer of the retail company is allowed to be visited multiple times, namely for each product type at least once. It might be more than once, if for that product type the total size of the suborders is larger than the vehicle capacity, Q. The suborders requested by a customer are, therefore, consolidated into orders that contain only one product. For each product type, the suborders are added in decreasing order of size to an order. If adding a suborder would lead to an order that is larger than Q, the suborder is added to a new order.

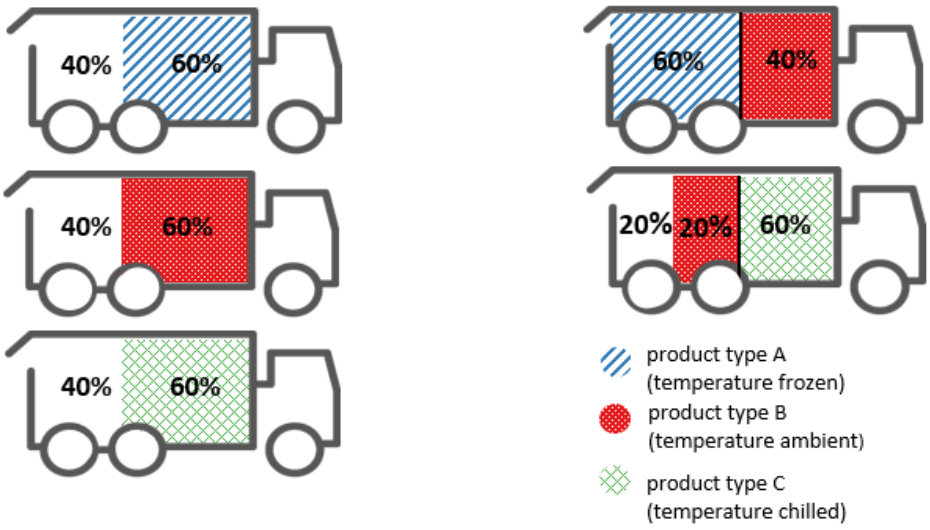
Consolidation per product type and planning all orders Having multiple compartments in the vehicles and having a load assignment algorithm, however, allow to plan all orders of a customer together in one run. A run is defined as applying all iterations of the ALNS. This could result in less visits to the customers and overall less distance travelled. We want to quantify the effect of having vehicles with compartments by planning all product types together. The same orders are used as input of the routing algorithm, as when planning per product consolidated orders separately for each product. Thus, all orders each consisting of one product type are the input in the ALNS framework.

Mixed consolidation We could also try to improve the composition of the orders, to obtain an overall better routing solution, which means a solution in which total distance is smaller. In this section, we propose a method to change the consolidation of suborders X_i of customer i into orders O_i. This method takes as input the initial orders O_i, which each consist of suborders of one product type, and makes new orders to replace them. An improvement refers to creating more and better insertion possibilities and the ability to more efficiently combine orders in vehicles. In Section 6.2 we present this method.

6.2 Change the suborder consolidation

We believe that we can improve the composition of the orders by taking into consideration the suborders of different products at the same customer location and the orders of neighboring

customers. By taking into account neighboring customers, we already anticipate on possible partial routing solutions when consolidating the suborders.
Suppose initially the demand of the customer is consolidated into three orders containing suborders of three different product types, each requiring a different temperature: frozen, chilled and ambient. If each of these orders would have a size of 60% of the vehicle capacity, none of the orders could be delivered by the same vehicle and three vehicles are required for deliveries to this customer. This is illustrated in Figure 4a. Assume that the suborders in these orders each have a size of 1% of the vehicle capacity and no capacity is lost by placing bulkheads. By transferring suborders between the three orders, less visits are required. The ambient order is combined with the frozen order by splitting of a part of 20% of the vehicle capacity of the ambient order. Now the frozen order and part of the ambient order are combined to fill a vehicle completely and the rest of the ambient order can be delivered together with the chilled order (see Figure 4b). Note this also depends much on the available configurations to hold these different product types.



(a) Three vehicles required

(b) Two vehicles required

Figure 4: Example showing that mixing suborders of different products may result in less vehicles visiting a customer

Improvement by taking into account neighboring orders can be illustrated as follows. Suppose an order consists of frozen goods with size equal to approximately 80% of the vehicle capacity and the only vehicle configuration that can hold this order is a configuration with 100% frozen capacity. If no close customer requires frozen products, the vehicle will only be filled for 80%. However, by re-distributing the frozen suborders and the suborders of another order of this customer, maybe more configurations can deliver these orders. This makes it easier to combine with neighboring orders.
In the remainder of this section we propose a method to change the orders. The algorithm can be separated in two phases. We first set out some specific element of which the algorithm makes use and then elaborate on the two phases.

Approximately full vehicle The algorithm tries to combine different orders which together could fill up a vehicle approximately. Therefore, we need to define what a full vehicle is. If a vehicle does not contain any compartments, a full vehicle would be a vehicle of which the whole capacity Q is used. However, if an order consists of multiple products, it might be that bulkheads have to be placed if that order is to be transported by a single vehicle. In that case, the actual available capacity is less than Q for any vehicle that would transport this order. To determine if an order o would fill up a vehicle, we take into account this loss in capacity, by defining the following capacity measure

$$Q(o) = Q - \text{lossCap}(\text{minComps}(o)). \tag{18}$$

Here the function minComps(o) gives the minimum number of compartments required in a vehicle to hold order o. This number is determined by iterating over all available configurations and determining the feasible configuration which has the least number of compartments. To determine whether or not a configuration is feasible, the load assignment heuristic of Section 5.2 is applied. The function lossCap() then gives the capacity that is lost by placing the bulkheads required for minComps(o). Now we have that an order o fills up a vehicle if its size is exactly equal to $Q(o)$.
To combine orders, we do not require an order to be exactly $Q(o)$, but approximately $Q(o)$. We define two measures for the approximate size of an order, namely

$$a(o) = Q(o) - \text{average suborder size} \times \gamma \tag{19}$$

and

$$b(o) = Q(o) - \text{average suborder size} \times \beta \tag{20}$$

Here we impose that $a(o) \leq b(o) \leq Q(o)$ and thus $0 \leq \beta \leq \gamma$. The average suborder size corresponds to the average size of all suborders of the instance.

Sorting of the orders per customer The orders of each customer are sorted based on the number of feasible configurations. This sorting is applied throughout the algorithm in both phase 1 and phase 2 for the orders per customer. When considering orders for combining, we want to start with the most problematic order first. This is the order which is most difficult to insert in a vehicle, thus, improving this order would increase its flexibility for insertion. Moreover, we start with the most problematic order, because it then still has the most possibilities for combining with other orders. An order is considered to be more problematic when it has the least number of feasible configurations when inserting it in an empty vehicle. The number of feasible configurations measures both the difficulty due to the size of the suborders as the difficulty due to the different products present in the order.

Neighboring orders In the second phase, the algorithm determines for each order o a set of orders from different customers O_i^n, with which it could possibly be inserted in the same vehicle. As we do not want to combine orders from customers that are too far away from each other, we define the neighboring orders in the set O_i^n to be from customers j that have a distance between them of at most $\delta(i, j)$. Moreover, the time windows of the customers should permit that the customers can be visited by the same vehicle. Therefore, a different customer j is only added to O_i^n if the time window restrictions of i, j or the depot are not violated if i is visited after j or the other way around.
The orders in O_i^n are sorted in increasing order of distance to customer i. Per customer, the orders are again sorted based on the number of feasible configurations.

Overview The general idea of the first phase is to merge, where possible, for each customer its orders to obtain orders $\tilde{o}$ that have a size $l_{\tilde{o}}$ larger than or equal to $a(\tilde{o})$. The merged orders are added to the set A, indicating that these orders are out of consideration for the remaining part of phase 1. In the second phase it tries to find combinations of orders of different customers for which the merged order $\tilde{o}$ has a size $l_{\tilde{o}}$ larger than or equal to $b(\tilde{o})$. The orders of these different customers are not actually combined into $\tilde{o}$, as we want to keep some flexibility and we do not want to restrict these customers to be in the same route. However, the separate orders that are combined in $\tilde{o}$ are added to the set B, indicating that a good combination is found and these orders do not have to be combined with any of the other orders.
We choose for a difference between the two measures $a(o)$ and $b(o)$, as we think it is most beneficial to combine orders of the same customer as much as possible. Moreover, as there are possibly more neighboring orders, we want to have a stricter requirement, in order to combine with the best option. In the remainder of the section we explain the two different phases, of which the pseudocode can be found in Algorithm 9.

Phase 1 In this first phase we look at each customer $i \in N$ separately. For each customer the algorithm keeps track of the number of extra splits σ_i that are allowed, which will be used in the second phase of the algorithm. It is initially zero for each customer. Moreover, the sets A and B are updated in each step of the algorithm, where B is only updated to be used in the second phase. Throughout the algorithm the orders of customer i in O_i are always sorted based on the number of feasible configurations, which is also updated when orders are added to or removed from the list.
Starting with the most difficult order $o \in O_i \setminus A$, the algorithm first tries, in line 7 up to line 15, to find a different order $o' \in O_i \setminus A$, such that the merged order $\tilde{o} = \{o\} \cup \{o'\}$ is a feasible order and $l_{\tilde{o}} \geq a(\tilde{o})$. A feasible order is an order for which a feasible load assignment can be found if it is inserted in an empty vehicle, with the load assignment heuristic. The value for σ_i is increased by one, as we have merged two orders.
If no such order can be found, the algorithm tries to combine o with another order $o' \in O_i \setminus A$ where a part of one of the two orders may be split off, in lines 16 up to 28. The merged order $\tilde{o}$ is either o combined with some suborders of o', or o' combined with some suborders of o. Again the requirement is that $\tilde{o}$ is feasible and its size satisfies $l_{\tilde{o}} \geq a(\tilde{o})$.
These steps are repeated until all orders of customer i have been considered, after which we finally in lines 29 up to 41 combine the not yet combined orders of $O_i \setminus A$ without imposing that the size of the merged order $\tilde{o}$ should be larger than $a(\tilde{o})$. Starting with the most difficult order, the other orders are added if adding leads to a feasible merged order. Multiple orders can be combined in one order and for each merging of orders, the value of σ_i is increased by one.
These steps are repeated for each customer.

Phase 2 In the second phase the algorithm follows similar steps as in phase 1 with neighboring orders instead of orders of the same customer. The differences are the following. First of all, two orders of different customers are not actually combined. That is, if a feasible order $\tilde{o}$ could be obtained from merging $o \in O_i$ and $o' \in O_i^n$ such that $l_{\tilde{o}} \geq b(\tilde{o})$, both o and o' are added to the set B, but $\tilde{o}$ is not actually kept as order. We do so, to keep the most flexibility and not restricting these two customers to be in the same route.
Secondly, in this phase the algorithm first tries to find combinations of orders $o \in O_i$ and $o' \in O_i^n$ for all customers without having to split off a part of one of the orders, before allowing to split off parts of orders (see lines 42 up to 48). In the first phase, the algorithm

first tries to combine the most difficult order with any other order, even if that means a part of an order must be split off, before considering the next most difficult order. We choose to do so, because we want that the difficult orders of neighboring customers also have enough possibilities for combining with other neighbors without splitting.
Thirdly, when combining with an order with the requirement to split parts of an order off, the allowed number of splits for this customer, σ_i should be larger than zero. Because we combine orders in the first phase, this could actually be the case. In contrast, in the first phase, the two separate orders would actually be combined into one order and one order would be split off, resulting in the same number of orders.
Finally, if no combination with a neighboring order could be found for order $o \in O_i \setminus B$ of customer i and if $\sigma_i > 0$, this order is split based on another criterion in line 62 up to line 70. The algorithm iterates over all suborders in o and transfers them to a new order o^2 if the total number of feasible configurations for o and the total number of feasible configurations for o^2 summed does not decrease. The suborders are sorted per product and suborders of the same product are sorted decreasing in size. We choose to already split order o, instead of first trying to find a neighboring order for the other orders of customer i, because o is more problematic than the next orders of the customer and it is important to improve the flexibility for this order.
When all orders of customer i are in the set B, it might be that $\sigma_i > 0$, but the algorithm does not split any of the orders further. These orders can already be combined with other orders to fill a vehicle efficiently and therefore do not require extra splits.

Algorithm 9 Change composition of the orders

1: **Input:** customers N, initial orders $O_i \ \forall i \in N$
2: $B \leftarrow \{o \in O_i \ , i \in N | l_o \geq b(o)\}$
3: **for each** $i \in N$ **do**
4: $\quad \sigma_i \leftarrow 0$
5: $\quad A \leftarrow \{o \in O_i \ , i \in N | l_o \geq a(o)\}$
6: $\quad$ **for each** $o \in O_i \setminus A$ **do**
7: $\quad\quad$ **for each** $o' \in O_i \setminus (A \cup \{o\})$ **do**
8: $\quad\quad\quad \tilde{o} \leftarrow \{o\} \cup \{o'\}$
9: $\quad\quad\quad$ **if** $\tilde{o}$ is feasible and $l_{\tilde{o}} \geq a(\tilde{o})$ **then**
10: $\quad\quad\quad\quad O_i \leftarrow O_i \setminus \{o, o'\} \cup \{\tilde{o}\}$
11: $\quad\quad\quad\quad \sigma_i \leftarrow \sigma_i + 1$
12: $\quad\quad\quad\quad A \leftarrow A \cup \{\tilde{o}\}$
13: $\quad\quad\quad\quad$ **if** $l_{\tilde{o}} \geq b(\tilde{o})$ **then**
14: $\quad\quad\quad\quad\quad B \leftarrow B \cup \{\tilde{o}\}$
15: $\quad\quad\quad\quad$ go to next order (line 6)
16: $\quad\quad$ **for each** $o' \in O_i \setminus (A \cup \{o\})$ **do**
17: $\quad\quad\quad \bar{o} \leftarrow \{o\}$
18: $\quad\quad\quad$ add suborders from o' to $\bar{o}$ in decreasing order if feasible
19: $\quad\quad\quad \hat{o} \leftarrow \{o'\}$
20: $\quad\quad\quad$ add suborders from o to $\hat{o}$ in decreasing order if feasible
21: $\quad\quad\quad$ **if** $l_{\bar{o}} \geq a(\bar{o})$ and/or $l_{\hat{o}} \geq a(\hat{o})$ **then**
22: $\quad\quad\quad\quad \tilde{o} \leftarrow$ largest order of $\bar{o}$ and $\hat{o}$
23: $\quad\quad\quad\quad A \leftarrow A \cup \{\tilde{o}\}$
24: $\quad\quad\quad\quad o^r \leftarrow (\{o\} \cup \{o'\}) \setminus \{\tilde{o}\}$
25: $\quad\quad\quad\quad O_i \leftarrow O_i \setminus \{o, o'\} \cup \{o^r, \tilde{o}\}$
26: $\quad\quad\quad\quad$ **if** $l_{\tilde{o}} \geq b(\tilde{o})$ **then**
27: $\quad\quad\quad\quad\quad B \leftarrow B \cup \{\tilde{o}\}$
28: $\quad\quad\quad\quad$ go to next order (line 6)
29: $\quad$ **for each** $o \in O_i \setminus A$ **do**
30: $\quad\quad \tilde{o} \leftarrow \{o\}$
31: $\quad\quad$ **for each** $o' \in O_i \setminus (A \cup \{o\})$ **do**
32: $\quad\quad\quad \tilde{o} \leftarrow \{\tilde{o}\} \cup \{o'\}$
33: $\quad\quad\quad$ **if** $\tilde{o}$ is feasible **then**
34: $\quad\quad\quad\quad O_i \leftarrow O_i \setminus \{o'\}$
35: $\quad\quad\quad\quad \sigma_i \leftarrow \sigma_i + 1$
36: $\quad\quad\quad$ **else**
37: $\quad\quad\quad\quad \tilde{o} \leftarrow \{\tilde{o}\} \setminus \{o'\}$
38: $\quad\quad O_i \leftarrow O_i \setminus \{o\} \cup \{\tilde{o}\}$
39: $\quad\quad A \leftarrow A \cup \{\tilde{o}\}$
40: $\quad\quad$ **if** $l_{\tilde{o}} \geq b(\tilde{o})$ **then**
41: $\quad\quad\quad B \leftarrow B \cup \{\tilde{o}\}$

42: **for each** $i \in N$ **do**
43: **for each** $o \in O_i \setminus B$ **do**
44: **for each** $o' \in O_i^n \setminus B$ **do**
45: $\tilde{o} \leftarrow \{o\} \cup \{o'\}$
46: **if** $\tilde{o}$ if feasible and $l_{\tilde{o}} \geq b(\tilde{o})$ **then**
47: $B \leftarrow B \cup \{o\} \cup \{o'\}$
48: go to next order (line 43)
49: **for each** $i \in N$ **do**
50: **for each** $o \in O_i \setminus B$ **do**
51: **if** $\sigma_i = 0$ **then**
52: go to next customer (line 49)
53: **for each** $o' \in O_i^n \setminus B$ **do**
54: $\tilde{o} \leftarrow \{o'\}$
55: add suborders from o to $\tilde{o}$ in decreasing order if feasible
56: **if** $l_{\tilde{o}} \geq b(\tilde{o})$ **then**
57: split o into o^1 and o^2, suborders present and not present in $\tilde{o}$, respectively
58: $O_i \leftarrow O_i \setminus \{o\} \cup \{o^1, o^2\}$
59: $\sigma_i \leftarrow \sigma_i - 1$
60: $B \leftarrow B \cup \{o^1, o^2\}$
61: go to next order (line 50)
62: **if** no order in O_i^n left to consider **then**
63: $o^1 \leftarrow o$
64: $o^2 \leftarrow$ new order without suborders
65: **for each** $x \in X_{o^1}$ with X_{o^1} sorted on product and decreasing in size **do**
66: **if** summed # feasible configurations of o^1 and of o^2 doesn't decrease **then**
67: transfer x to o^2
68: $O_i \leftarrow O_i \setminus \{o\} \cup \{o^1, o^2\}$
69: $\sigma_i \leftarrow \sigma_i - 1$
70: go to next order (line 50)

7 Experiments

7.1 Overview

A few sets of experiments are performed in order to evaluate the added value of considering vehicles with multiple compartments and test the methods developed in this thesis. The first set of experiments serves to tune our algorithm. In this set of experiments, we include experiments to tune the parameters in the proposed sorting method for the configurations and to tune the parameters of the proposed method to change the composition of the orders. In the second set of experiments the proposed load assignment heuristic is compared with the exact approach of solving the ILP formulation using CPLEX in the routing problem. This set of experiments serves two purposes. First of all, we can evaluate the boundaries in terms of runtime of the exact solution approach by solving the routing problem with the two different load assignment methods. Secondly, we evaluate how often our proposed heuristic, does not find a feasible configuration for the insertion, while there is in fact a feasible configuration. To test this, the load assignment problem is solved by using the heuristic in the sub-problems. If no feasible configuration has been found, the load assignment problem is solved again with the ILP for the sub-problems being solved by CPLEX. The algorithm

does not actually use the results found by CPLEX, but only stores them to evaluate how often our heuristic does not find the exact solution. An exact solution corresponds to returning a feasible configuration if there exists one and only returning no configuration if there is indeed no feasible configuration. Summarizing, this second set of experiments consists of the following experiments.

- solveHeuristic: solve routing problem where load assignment problem is solved with the heuristic in the sub-problem
- solveExact: solve routing problem where load assignment problem is solved by CPLEX in the sub-problem
- solveHeuristicEvaluateExact: solve routing problem where load assignment problem is solved with the heuristic in the sub-problem, but if no feasible configuration is found, load assignment problem is solved for evaluation of correctness using CPLEX in the sub-problem

The third set of experiments is carried out to quantify the effects for the retail company of having vehicles with multiple compartments and applying a load assignment algorithm in order to be able to plan all product groups together, instead of using different vehicles for different product types. Here we also test the method that changes the composition of the orders. For each test instance, the following settings can be distinguished.

- nonMixedOrdersPlannedSeparate: each order contains one product type and the orders are planned per product type (one optimization call per product type)
- nonMixedOrdersPlannedTogether: each order contains one product type, but all orders are planned together
- mixedOrdersPlannedTogether: mixed orders are made using the heuristic in Section 6.2 and all orders are planned together
- suborderOrdersPlannedTogether: each suborder is an individual order and all orders are planned together

All methods developed and discussed in this thesis are coded in Java, with CPLEX to solve the ILP formulation as presented in Section 5.2.4. The experiments are performed on a 2.60 GHz Intel Core i7-5600U with 4 cores and 16 GB RAM, running Windows 10.

7.2 Test instances

The above mentioned experiments are conducted on two sets of instances.

7.2.1 Data from the retail company

The first set of instances is provided by the considered retail company. It consists of customer demand for three different depots, each instance corresponding to one day and demand for one depot location.
The customers request suborders with three required temperatures: ambient, chilled and frozen. These temperatures correspond to the different compartment temperatures. The different products that can be requested are aggregated to five product types. One product type requires frozen delivery, two product types need to be chilled and two product types need to be transported in ambient climate. For the retail company the two chilled product types contaminate, whereas the others are compatible.
The vehicle set is limited, but consists of so many available vehicles that it is never restricting.

The capacity, Q, is equal to 1960 for all vehicles. Moreover, the company provides a list of 72 available configurations for the compartments in the vehicles. This list is made by the retail company based on the positions at which bulkheads can be placed and the different requirements for the different compartment temperatures. The list includes one configuration corresponding to a vehicle without compartments for each temperature. The maximum number of bulkheads that can be placed is two, to obtain a maximum of three compartments, which leads to approximately a 7% loss in vehicle capacity.
The set of instances that is used in this thesis consist of 18 instances denoted by $W1$ up to $W18$. We denote the three different depots by A, B and C. A description per instance can be found in Table 17 in Appendix A.1.
The average number of customers per instance is 131 and the average number of suborders is 3832. The instances of depot A contain the least number of customers and suborders, on average 98 and 2820. Depot B has on average 149 customers which request on average 4359 suborders and depot C has on average 146 customers per instance demanding on average 4318 suborders.
The average size of the suborders is approximately 60, which means that approximately 33 suborders would fit in a vehicle without compartments. When making orders per product, thus using setting nonMixedOrdersPlannedSeparate or nonMixedOrdersPlannedTogether, each order contains on average 8 suborders. Note that an order for one customer is constructed by adding suborders until adding a suborder would result in a total order size that is larger than vehicle capacity and then continuing creating new orders in the same way with the remaining suborders. For most customers, the request amount per product is smaller than vehicle capacity.

7.2.2 Constructed instances

The presented load assignment algorithm is developed to handle a wide range of problems without limiting on specific graph types. Therefore, to make the experiments a bit more challenging, we also test on self constructed instances based on instances for different problems in literature. There are no benchmark instances for the specific problem that is studied in this work, so we combine the insights, methods and instances from different problems.
At the basis of these instances are the benchmark instances with 100 customers from Solomon (1987). These instances can be downloaded from SINTEF. The customer locations, time windows and service times are not changed for our instances. The following adaptations of these instances will be discussed in the next paragraphs. The order of each customer is split into suborders, to suit our problem. To better be able to divide the capacity of the vehicle in compartments of integer size, which is more intuitive, we multiply the vehicle capacity and demand size per customer by ten. We assume that an infinite number of vehicles is available. Further adaptions involve the division into suborders of different products, determine the conflict-graph to indicate which products contaminate and generating a set of configurations for the vehicles.

Product types Each instance consists of 10 product types. Each product type is randomly assigned one of the three temperature regimes: ambient, chilled or frozen. For each instance we use one of the following probability sets, each element being the probability of assigning one of the temperatures to a product: $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ and $(\frac{3}{5}, \frac{3}{10}, \frac{1}{10})$. With the first set, each temperature is equally likely to be assigned to a product, whereas the products are more likely to be assigned the temperature ambient and less likely to be assigned a frozen required temperature using the second set. By varying the number of products with the same required

temperature, we also vary in the amount of contamination possible per temperature.
The conflict-graph per required temperature that indicates contamination of different product types is made using the $\hat{p}$-generator described in Gendreau, Soriano, and Salvail (1993). In this procedure, each node, corresponding to a product, i, in the conflict graph is assigned a number $\hat{p}[i]$ uniformly at random in the interval (a, b), with $0 \leq a \leq b \leq 1$. For each combination of nodes i and j, edge (i, j) is generated with probability $\frac{\hat{p}[i]+\hat{p}[j]}{2}$. This procedure creates a conflict graph with expected density $\frac{a+b}{2}$. The authors have improved this procedure compared to what had been studied earlier in literature, where $a = b$ and $\hat{p}[i]$ is equal for all nodes i. By having different values for a and b, more diverse graphs are created. For details we refer to the work of Gendreau et al. (1993). We distinguish between two cases: density of 0.25, with $a = 0.0$ and $b = 0.5$, and density of 0.75, with $a = 0.5$ and $b = 1.0$. The non-conflict graph is constructed using the information of the conflict graph. The reason we do not allow for more products in the case, is that too much variety of products between different customer orders might make it almost impossible to plan orders of different customers into one route, limiting the routing possibilities too much. Moreover, the vehicles would be less filled, which makes the load assignment problem less challenging.

Configurations In this paragraph we explain step by step how we generate configurations, starting with the number compartments and their temperature regimes. Thereafter, we explain how we determine the sizes of the compartments.
We distinguish between four different cases for generating the configurations of compartments for the vehicles.
First of all, the maximum number of compartments, $c_{\max}$, is either low or high. With low maximum number of compartments, the configurations contain one up to three compartments, whereas the configurations contain one up to five compartments in the case of a high number of compartments. For $r = 1, \ldots, c_{\max}$, we include one configuration for each combination of the three temperature regimes in the r compartments, where we allow for repetition. We allow for this repetition of temperature regimes inside one configuration, because multiple compartments of the same temperature regime might be required to hold products with the same required temperature that contaminate. For the case in which we have a high number of compartments, this results in

$$\sum_{r=1}^{5} \binom{r+2}{r} = 55 \tag{21}$$

configurations. When the maximum number of compartments is equal to three, the number of configurations is 19.
Instead of only considering a low or high number of compartments, the second classification is into a low or high number of configurations. A low number of configurations corresponds to the previously discussed 19 and 55 configurations. A high number of configurations is obtained by generating three configurations for each r-combination of the temperatures. So instead of only generating one configuration with two compartments with ambient and chilled temperature regime, we generate two such configurations. The compartment sizes may differ between these two configurations. Only for the configurations with one compartment, we do not add additional configurations, because they would be exactly the same. This results in two additional cases, one for which the number of configurations is equal to

$$3\sum_{r=1}^{5} \binom{r+2}{r} - 6 = 159. \tag{22}$$

When $c_{\max}$ equals three and the number of configurations is high, there are 51 configurations generated.
For each created configuration, we still need to determine the capacities of the compartments. For each bulkhead that is placed, the total capacity is reduced with 5% of the initial vehicle capacity. In addition, we impose that each compartment should have a capacity of at least 10% of the initial vehicle capacity, so the compartments are not extremely small. For each configuration the positions of the bulkheads are iteratively drawn uniformly at random from the set of integers between zero and the vehicle capacity. In case that drawing a bulkhead position results in one of the compartments being too small, we draw a different position. It might be that the previous placed bulkheads might make it very difficult to draw the next bulkhead, such that all compartments are large enough. Therefore, after drawing 15 times a position without success, we remove all bulkheads and start over for this configuration.
Finally, for each configuration we randomly match the temperature regimes with the compartment sizes.

Customer orders For the customer orders we first determine how many suborders should be in the order uniformly at random from the interval $[5, 15]$. The different products for this order are also determined randomly. We draw from the interval $[1, 7]$ to determine of how many products the order will consist, after which this number of products is drawn from the set of all products.
The customer orders are divided into suborders in a similar way as for dividing the vehicle capacity in different compartment sizes, where we do not restrict a suborder to have a minimal size. Each suborder can have any integer size larger or equal to one.
Each suborder is then assigned uniformly at random one of the products that have been assigned to the order previously.
Each of these orders is directly input for the ALNS. It might be that some of the generated orders cannot be placed in a vehicle using one of the generated configurations. Such an order will not be planned and will stay in the order bank. This only happens for a few orders.

7.3 Computational results

7.3.1 Algorithm tuning

In this section we tune the algorithm and discuss which parameters we use for the methods presented in this thesis.

Adaptive Large Neighborhood Search For the ALNS, we take the same parameters as in Ropke and Pisinger (2006). The authors have tuned these parameters on benchmark instances and extensively and successfully tested ALNS with these parameters, also in the subsequent work (Pisinger & Ropke, 2007). Besides, we focus in this work on testing the elements of the methodology specifically designed for the presented MCVRPTWSD.
The number of orders closest to the seed order to consider for removal in the time related removal heuristic, τ, is not specified by the authors. We choose it to be $0.7|O|$, where $|O|$ is the number of orders in the case. This way, the furthest orders are not considered to be related to the seed order, but still a substantial amount of different orders are left for comparison.
Hence, we use the following parameter vector $(p, p_{\text{worst}}, \tau, w, c, \rho, \sigma_1, \sigma_2, \sigma_3, \eta) = (6, 3, 0.7|O|, 0.05, 0.99975, 0.1, 33, 9, 13, 0.025)$. Here, p is the parameter that determines the amount of randomness in the distance related removal heuristic and time related

removal heuristic. Parameter p_{worst} determines the randomness for the worst related removal heuristic. Parameter w indicates how much worse a new solution can be to be accepted with probability 0.5 in simulated annealing and parameter c determines how much the temperature T decreases in each ALNS iteration. Parameter ρ indicates how much of the weight is determined by the scores obtained during the previous segment, whereas $1 - \rho$ indicates the part that is determined by the previous weight. Parameters σ_1, σ_2 and σ_3 are the score increments when a new global best solution, a better not yet accepted solution and a worse not yet accepted solution are obtained, respectively. Finally, η determines the maximum amount of noise that is added to the objective value.
The number of iterations is $n = 10,000$, with some exceptions for the tuning which will be discussed in the next paragraphs.

Tuning of method that changes orders We tune the parameters β, γ and $\delta(i, j)$, that we presented in Section 6.2. The first two parameters are used to determine what size of a merged order is acceptable in one of the phases of the algorithm and the $\delta(i, j)$ is the acceptable distance for neighboring customers.
For β a value in the set $\{1, 1.5, 2\}$ is chosen, indicating that we accept orders that have a size of vehicle capacity minus 1, 1.5 or 2 times the size of an average suborder. Note that this vehicle capacity already takes into account the capacity loss due to placing bulkheads. Parameter γ indicates this for the first phase, where somewhat smaller orders are accepted. For the values of γ we make use of the average number of suborders y that are in the orders before changing them, thus, when the orders contain one product. We choose from the set $\{y, \frac{y}{1.5}, \frac{y}{2}\}$.
Let i be the customer for which a set of neighboring customers must be defined, let j be a potential neighboring customer and let $\hat{d}$ be the depot node. We choose one of the following values for $\delta(i, j)$: $\frac{1}{3}d_{\hat{d}i}$, $\frac{1}{2}d_{\hat{d}i}$ or $d_{\hat{d}i} + d_{\hat{d}j}$. The first two values can be seen as a greedy indication, that draws a circle around customer i with the specified radius. The third value imposes that a route starting at the depot, visiting customer i and j and then returning to the depot, should cover less distance than when visiting each customer in a separate route. In total, 27 combinations of the parameters are, thus, tested for each of the instances.
This method is applied to the data of the retail company and therefore also tuned on instances of the company. These experiments are performed using the solveHeuristic setting. The tuning set consists of six instances: $W1, W6, W7, W11, W13$ and $W17$. This is a diverse set of instances, in which all depots are represented. The number of iterations n is set to 1000 for the tuning, due to time considerations. We believe that we can already have an indication of best parameter settings after this number of iteration when taking all six instances into account. Moreover, we do not sort the configurations which would not affect the distance and only the runtime.
As criterion for selecting the parameters, we take the average deviation from the best solution found in the experiments for each instance. We found that the best parameters are: $\beta = 2$, $\gamma = \frac{y}{2}$ and $\delta(i, j) = d_{\hat{d}i} + d_{\hat{d}j}$. The average deviations for each parameter combination can be found in Table 20 in Appendix A.2.

Tuning of the configuration sorting method We tune parameter α, which controls the influence of the previous failures in the configuration sorting score and parameter κ, the number of iterations during which the algorithm uses the same order of configurations. For each of the parameters, we choose four different values. For α we test the values $\{0.1, 0.2, 0.5, 0.8\}$ and for κ the values $\{10, 20, 40, n\}$. Here n is equal to the number of

iterations of the ALNS heuristic, thus $\kappa = n$ means no sorting takes place. There are therefore 13 different tuning cases.
For the tuning of α and κ, a better solution corresponds to a solution with a lower runtime, as this is what the configuration sorting should have improved. We, thus, base our decision on the total time spent in the load assignment algorithm and in the configuration sorting method together. The criterion for choosing the best parameters is the average deviation from the lowest total time spent in both methods obtained during this tuning experiment.
We tune the constructed instances separately from the instances of the retail company as the two sets of instances differ a lot and use the setting solveHeuristic. We use 16 constructed instances, $S1$ up to $S16$. Each of these instances is constructed using 1 out of the 16 different construction possibilities, discussed in Section 7.2.2, and one of the Solomon $C101$, $C201$, $R101$, $R201$, $RC101$ and $RC201$ instances. This way we have a diverse set of instances. For details on these tuning instances we refer to Table 18 in Appendix A.1. The number of iterations used for this tuning is $n = 10.000$.
For the data of the retail company, we take the same set of instances as for the tuning of the methods which changes the orders. To also have suitable parameters for the case in which orders consist of different product types, we tune $W1$, $W7$ and $W13$ with the nonMixedOrdersPlannedTogether setting and $W6$, $W11$ and $W17$ with the setting mixedOrdersPlannedTogether. Here, we use $n = 1000$ iterations. This should actually not influence the decision as the runtime is almost proportional to the number of iterations used in the ALNS.
The configuration sorting has shown to be beneficial for 15 out of 16 instances of $S1$ up to $S16$. Only for instance $S4$, the time spend in the load assignment algorithm is not improved when sorting the configurations. The total time spend in the load assignment algorithm and the sorting method together is for most instances the lowest for $\alpha = 0.2$ and $\kappa = 20$. These parameters are best for 4 instances, whereas $\alpha = 0.5$ and $\kappa = 20$ are best for 3 instances and have the best average deviation. However, as we value that the parameters perform well on the average instance, we choose 0.5 and 20. In Table 21 in Appendix A.2, the average deviations for each parameter combination can be found.
Table 1 shows for each of the tuning instances how many configurations were tried before finding a feasible configuration on average per load assignment call. It shows these numbers both for the case in which the configurations are sorted using the parameters $\alpha = 0.5$ and $\kappa = 20$ as for the case in which no sorting took place. For all instances, the average number of configurations that were tried is decreased by sorting. Note that the numbers are rounded up to two decimals.

Table 1: Average number of configurations tried with the solveHeuristic experiments per load assignment problem

	instance															
sorted	$S1$	$S2$	$S3$	$S4$	$S5$	$S6$	$S7$	$S8$	$S9$	$S10$	$S11$	$S12$	$S13$	$S14$	$S15$	$S16$
Yes	1.02	1.16	1.09	1.00	1.49	1.66	1.07	1.42	1.04	1.11	1.03	1.00	1.06	1.23	1.10	1.03
No	1.11	2.68	2.75	1.01	3.69	3.65	1.75	5.67	1.27	1.61	1.36	1.00	1.31	2.25	2.33	1.34

Notes. Average number of configurations tried by the algorithm during the solveHeuristic experiments. The values are shown for the case in which configurations are sorted using parameters $\alpha = 0.5$ and $\kappa = 20$ and the case of not sorting.

Nonetheless, sorting does not result in a large absolute decrease in the runtime. The average total time spend in the load assignment algorithm without sorting over all instances is 31.35 seconds. The average total time spend in the load assignment algorithm and sorting method together for the chosen parameters is only slightly smaller, 31.18 seconds. The total average runtime for the cases is rounded to 104 seconds. Even though the improvement is minimal, we continue with these parameters.

For the instances of the retail company, sorting did not improve the overall runtime and the average deviation from the best time is smallest for $\kappa = n$. A possible explanation for this could be that the instances only consist of five products with few contamination and the pre-checks on compartment capacities per temperature for each configuration determine quickly whether or not the configuration should be tried. The average deviations for each parameter setting can be found in Table 22 in Appendix A.2.

7.3.2 Load assignment algorithm

In this section, we present the solutions from the second set of experiments. We run the experiments solveHeuristic, solveExact and solveHeuristicEvaluateILP on the 48 constructed instances $S17$ up to $S64$. The same experiments are performed for $W2, W3, W4, W5, W8, W9$, $W10, W12, W14, W15, W16$ and $W18$ using the setting nonMixedOrdersPlannedTogether. We do not change the composition of the orders, as the method which changes these orders imposes that orders must be feasible. Taking into account this restriction when changing the orders could influence the load assignment problem.
Finally, we also analyze the added value of the presented order related removal heuristic.
For details on the instances we refer to Table 17 and Table 19 in Appendix A.1.

Results on the constructed instances First, we present the results on the constructed instances. Because the runtime of the problem is much larger when solving the sub-problem with CPLEX, we set an additional stopping criterion on the ALNS based on the runtime. When the runtime exceeds one hour, we stop the algorithm. We believe one hour is reasonable, as the runtime with the load assignment heuristic in the sub-problem always solves within 10 minutes. The iteration is always completely finished before stopping, so the actual runtime can be larger than 3600 seconds. This time limit is set for both the experiments solveExact as well as for experiments solveHeuristicEvaluateExact.
This paragraph is structured as follows. First we analyze the results in three parts, each part corresponding to one set of instances based on the underlying Solomon instances: RC207, C102 or R205. Then, we discuss two examples of load assignments that were not found by the proposed heuristic and finally we summarize the overall result.
Because we want to see if a similar pattern can be found when we allow for a longer runtime than one hour, we set the time limit to four hours for the instances constructed based on the Solomon RC207 instance, namely $S17$ up to $S32$. The results for these instances can be found in Table 2 for the solveHeuristic and solveExact experiments.
The total distance traveled, the number of used vehicles in the final solution and the total runtime of the ALNS are reported. The primary objective is to minimize the distance, but to have a more complete overview of the solution, the number of vehicles is also reported. In bold we highlight the minimum over the two approaches, if it is not equal. Because the 100 constructed orders are not necessarily feasible, we report the number of planned orders, which could be different for both experiments if the heuristic is not able to plan an order in an empty vehicle, whereas CPLEX does find the feasible assignment.
To evaluate the cause of the different solutions of the solveHeuristic and solveExact experiments, we analyze the results together with the results of the solveHeuristicEvaluateExact experiments.

Table 2: Results of instances $S17$-$S32$ for experiments solveHeuristic and solveExact

	solveHeuristic				solveExact				
instance	distance	# vehicles	# orders	runtime(s)	distance	# vehicles	# orders	runtime(s)	n
$S17$	1331.83	9	94	53	1331.83	9	94	782	10000
$S18$	1647.93	10	91	89	1647.93	10	91	1493	10000
$S19$	1637.39	11	100	60	1637.39	11	100	10432	10000
$S20$	991.24	7	100	65	991.24	7	100	8430	10000
$S21$	2525.77	22	74	569	2525.77	22	74	8407	10000
$S22$	2099.42	16	80	227	2099.42	16	80	2247	10000
$S23^*$	**1203.77**	8	100	74	1228.21	8	100	14405	1802
$S24^*$	**1587.74**	11	100	81	1698.65	11	100	14400	1626
$S25$	1218.03	8	98	62	1218.03	8	98	11922	10000
$S26^*$	**1751.97**	14	85	176	1771.33	13	85	14404	6836
$S27^*$	**982.46**	6	100	70	986.77	6	100	14400	7274
$S28^*$	**984.65**	6	100	96	998.6	6	100	14402	3346
$S29^*$	**2445.05**	18	93	153	2474.4	18	93	14404	2434
$S30$	2305.18	19	81	277	2305.18	19	81	9221	10000
$S31^*$	**1432.11**	12	100	58	1550.35	13	100	14446	213
$S32^*$	**1492.97**	10	100	66	1525.46	11	100	14402	760
total	25637.51	187	1496	2176	25990.56	188	1496	168197	104291

Notes. Results of instances $S17$ up to $S32$, which are based on Solomon instance $RC207$, for the experiments solveHeuristic and solveExact. In bold is the best distance obtained from the experiments, if the solutions are not equal. Due to the maximum set runtime of 4 hours, the pre-defined 10,000 ALNS iterations n were not always performed completely for the solveExact experiments. The instances for which not all 10,000 ALNS iterations are performed due to the runtime limit of 4 hours, are marked with an asterisk (*).

Table 3 presents the results of the solveHeuristicEvaluateExact experiments. It shows for each instance the number of times the load assignment problem is solved, which is done after one of the insertion options is chosen by one of the insertion heuristics, and how many times a feasible assignment was found when applying the heuristic in the sub-problem. It also shows how many times the load assignment problem is solved using CPLEX to solve the ILP in the sub-problem in an exact manner, which is always when no feasible configuration could be found, and how many times this resulted in a feasible load assignment. For none of the instances, this was the case, meaning that the heuristic was able to solve all load assignment problems correctly during the 10,000 iterations or four hours runtime limit.
When all 10,000 iterations have been performed for both sets of experiments, the difference between solutions is due to different results in the load assignment problems. In this case, when all load assignment problems were solved correctly with the heuristic, the same solutions are obtained with both approaches.
The results from the solveHeuristic experiments are for none of the instances worse than with the solveExact experiments. Moreover, we can see that the number of planned orders is the same for both approaches. The better solutions obtained by the solveHeuristic experiments are only due to the fact that more routing iterations could have been performed, as all load assignment problems were solved correctly with the heuristic.

Table 3: Results of instances $S17$-$S32$ for experiments solveHeuristicEvaluateExact

	solveHeuristic			exact evaluations
instance	# problems	# feasible	# classified as infeasible	# incorrect results heuristic
$S17$	1533322	250036	1283286	0
$S18$	2619964	250033	2369931	0
$S19$	611566	250042	361524	0
$S20$	250288	250042	246	0
$S21$	10435525	250016	10185509	0
$S22$	6524010	250022	6273988	0
$S23^*$	70655	50723	19932	0
$S24^*$	98111	37709	60402	0
$S25$	727009	250040	476969	0
$S26^*$	2728272	161166	2567106	0
$S27$	250342	250042	300	0
$S28$	250042	250042	0	0
$S29^*$	674023	49058	624965	0
$S30$	6853674	250023	6603651	0
$S31^*$	10588	4741	5847	0
$S32^*$	39257	16536	22721	0

Notes. Results of instances $S17$ up to $S32$, which are based on Solomon instance $RC207$, for the experiments solveHeuristicEvaluateExact. The instances for which not all 10,000 ALNS iterations are performed due to the runtime limit of 4 hours, are marked with an asterisk (*). The third and fourth column denote the number of load assignment problems solved and the number of times a feasible configuration was found, respectively. The fifth column shows how often no feasible configuration was found with the heuristic and the sixth column shows how many times this result was incorrect.

The results for the instances $S33$ up to $S48$ can be found in Table 4 and in Table 5. It can be seen that only for one of these instances, a better solution is obtained within one hour using CPLEX in the load assignment sub-problem, whereas for 13 instances a better solution is obtained for the solveHeuristic experiments.
When the solveExact experiments are stopped based on the runtime criterion, a worse solution might be only due to the fact that not enough iterations were performed, which is the case when zero feasible evaluations are stated in Table 5. Instances $S33$, $S37$, $S38$, $S39$, $S45$ and $S46$ were solved correctly with the heuristic in the load assignment sub-problem and have, therefore, a better or equal solution with the solveHeuristic experiments compared to the solveExact experiments.
In total, for 10 instances, the load assignment problem did not return the exact solution (see Table 5). However, for nine of these instances, a better final solution is obtained with the solveHeuristic experiments. Even for the three such instances that were solved with both experiments up to 10,000 iterations ($S34$, $S35$ and $S36$), a better solution was obtained with the solveHeuristic experiments. This indicates that for these cases, the final solution is not necessarily better when all feasible assigments are found.
Only for instance $S41$ a better solution is obtained with the solveExact experiment. From the 664485 load assignment problems, 4363 problems are not solved correctly by the heuristic. However, the lower distance with the solveExact experiment comes at the expense of the runtime. When we allow the solveHeuristic experiment to run for more than 10,000 iterations until the total runtime is 3600 seconds, the total distance of the final solution, 1730.63, is lower than with the solveExact experiment with the same runtime.

Table 4: Results of instances $S33$-$S48$ for experiments solveHeuristic and solveExact

	solveHeuristic				solveExact				
instance	distance	# vehicles	# orders	runtime(s)	distance	# vehicles	# orders	runtime(s)	n
$S33$	1963.26	21	87	156	1963.26	21	87	989	10000
$S34$	**1219.79**	14	100	40	1222.08	14	100	1018	10000
$S35$	**1162.78**	14	100	35	1163.19	14	100	2250	10000
$S36$	**1213.48**	14	100	49	1222.70	14	100	2751	10000
$S37^*$	**2501.41**	30	79	392	2518.04	30	79	3600	8294
$S38$	1602.32	22	76	263	1602.32	22	76	1094	10000
$S39^*$	**2275.95**	29	100	88	2382.13	30	100	3600	1086
$S40^*$	**1823.52**	22	99	93	2039.85	22	99	3602	660
$S41^*$	1771.50	21	97	62	**1731.37**	19	97	3600	7400
$S42^*$	**1989.41**	19	90	127	1997.11	22	90	3600	5471
$S43^*$	**1387.36**	17	100	54	1415.44	17	100	3600	8319
$S44^*$	**1142.43**	13	100	47	1143.45	13	100	3600	7406
$S45^*$	**2029.72**	21	82	234	2101.69	22	82	3601	1197
$S46^*$	**2147.65**	24	75	410	2160.30	24	75	3600	3387
$S47^*$	**2069.33**	28	99	99	2214.86	26	99	3615	600
$S48^*$	**1891.78**	21	100	70	1954.31	22	100	3601	612
total	28191.69	330	1484	2219	28832.10	332	1484	47721	94432

Notes. Results of instances $S33$ up to $S48$, which are based on Solomon instance $C102$, for the experiments solveHeuristic and solveExact. This table should be interpreted as Table 2, with a runtime limit of 1 hour.

Table 5: Results of instances $S33$-$S48$ for experiments solveHeuristicEvaluateExact

	solveHeuristic			exact evaluations
instance	# problems	# feasible	# classified as infeasible	# incorrect results heuristic
$S33$	3466193	250029	3216164	0
$S34$	387389	250042	137347	26
$S35$	353355	250042	103313	62
$S36$	380831	250042	130789	399
$S37^*$	4845866	153604	4692262	0
$S38$	6381107	250018	6131089	0
$S39^*$	141001	25068	115933	0
$S40^*$	64185	16290	47895	298
$S41^*$	664485	127985	536500	4363
$S42^*$	1702082	151971	1550111	136
$S43$	492331	250042	242289	171
$S44$	336080	250042	86038	930
$S45^*$	434375	21784	412591	0
$S46^*$	1730506	54942	1675564	0
$S47^*$	63197	11153	52044	1
$S48^*$	46992	14637	32355	18

Notes. Results of instances $S33$ up to $S48$, which are based on Solomon instance $C102$, for the experiments solveHeuristicEvaluateExact. This table should be interpreted as Table 3, with a runtime limit of 1 hour.

For the last set of constructed instances, the results can be found in Table 6 and in Table 7. Only for two instances, the load assignment problem was not solved correctly throughout the performed iterations, namely for instance $S58$ and instance $S63$. For instance $S58$ always when no feasible configuration was found using the heuristic, CPLEX was able to identify a feasible configuration. Nonetheless, this has not shown to be advantageous, as the final solution for the solveExact experiment after one hour runtime is worse than the solution obtained with the heuristic. It might be that more runtime would result in a better solution for the solveExact experiment for this instance, but the runtime is now already 55 times higher than the runtime of the solveHeuristic experiment.
Striking is the result for instance $S63$, as the solveExact experiment has only run for 13 iterations. The load assignment problem is, thus, solved extremely slowly for this instance.

In one hour runtime, none of the solutions for these instances are better with the solveExact experiments.

Table 6: Results of instances $S49$-$S64$ for experiments solveHeuristic and solveExact

	solveHeuristic				solveExact				
instance	distance	# vehicles	# orders	runtime(s)	distance	# vehicles	# orders	runtime(s)	n
$S49$	1559.37	15	86	111	1559.37	15	86	1267	10000
$S50$	1556.82	16	91	99	1556.82	16	91	1059	10000
$S51^*$	**954.75**	5	100	58	966.68	6	100	3601	3920
$S52^*$	**956.65**	5	100	49	959	5	100	3600	6670
$S53^*$	1563.01	17	70	320	1563.01	17	70	3601	6942
$S54$	1915.79	27	82	360	1915.79	27	82	2796	10000
$S55^*$	**1612.26**	18	99	86	1706.02	20	99	3607	763
$S56^*$	**1310.59**	12	100	71	1357.91	12	100	3603	1257
$S57$	1449.42	14	98	42	1449.42	14	98	1678	10000
$S58^*$	**979.32**	5	100	66	980.64	5	100	3600	1103
$S59^*$	**1081.64**	7	100	41	1084.39	8	100	3600	5688
$S60^*$	**964.36**	6	100	76	984.34	6	100	3601	1441
$S61^*$	**1979.03**	22	77	380	2010.27	24	77	3600	2384
$S62^*$	**1848.00**	22	86	229	1871.22	22	86	3601	459
$S63^*$	**1152.11**	9	100	53	1346.81	8	100	3791	13
$S64^*$	**1164.32**	10	100	69	1186.24	8	100	3601	1129
total	22047.44	210	1489	2110	22497.93	213	1489	50206	71769

Notes. Results of instances $S49$ up to $S64$, which are based on Solomon instance $R205$, for the experiments solveHeuristic and solveExact. This table should be interpreted as Table 2, with a runtime limit of 1 hour.

Table 7: Results of instances $S49$-$S64$ for experiments solveHeuristicEvaluateExact

	solveHeuristic			exact evaluations
instance	# problems	# feasible	# classified as infeasible	# incorrect results heuristic
$S49$	4348037	250028	4098009	0
$S50$	2966259	250033	2716226	0
$S51$	250044	250042	2	0
$S52$	250042	250042	0	0
$S53^*$	4464445	112759	4351686	0
$S54^*$	7559467	248384	7311083	0
$S55^*$	82982	17813	65169	0
$S56^*$	69603	30474	39129	0
$S57$	1010662	250040	760622	0
$S58$	283714	250042	33672	33672
$S59$	306214	250042	56172	0
$S60$	250042	250042	0	0
$S61$	1492422	41255	1451167	0
$S62^*$	201874	9177	192697	0
$S63^*$	552	123	429	4
$S64^*$	64422	43133	21289	0

Notes. Results of instances $S49$ up to $S64$, which are based on Solomon instance $R205$, for the experiments solveHeuristicEvaluateExact. This table should be interpreted as Table 3, with a runtime limit of 1 hour.

We give an example of a feasible load assignment that is not found by the load assignment heuristic for instance $S34$. The first time that the heuristic did not find the feasible configuration is in iteration 133 of the ALNS. It did not find the assignment of ambient suborders to two ambient compartments of capacities 584 and 284, where none of the suborders contaminate. The total size of the suborders to be inserted is 868, which is equal to the total capacity of the compartments. The assignment that CPLEX found can be seen in Table 8.

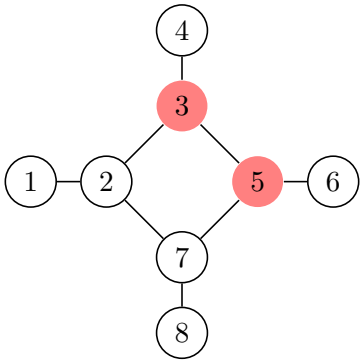
Table 8: Feasible load assignment not found by the proposed heuristic for instance $S34$

suborder size																	
24	25	14	44	3	32	65	21	19	16	36	6	8	1	17	2	93	31	= 584
2	36	16	55	21	48	5	53	10	11	1								= 284

Notes. Example of feasible load assignment found by CPLEX and not found by the proposed heuristic for instance $S34$. The suborders consist of one chilled product and have to be assigned to ambient compartments with capacities 584 and 284.

We also take a closer look at the wrong load assignment response with the solveHeuristic experiment for instance $S58$. The sub-problem for which CPLEX found a feasible assignment, but the heuristic did not, can be described as follows. Two ambient compartments of large enough size are available to hold suborders containing eight different types of products. The conflict graph for these products is shown in Figure 5. The maximum conflict clique, containing products 3 and 5, is identified. First, a non-conflict clique containing product 3 is created. Products 1 and 6 are added first and the tie breaking rules determined to add 8 instead of 7, because both have an equal degree in the remaining graph and more suborders consist of product type 8. Secondly, the non-conflict clique with product type 5 is constructed, which cannot contain product type 7 due to contamination. Product 7, therefore, remains unassigned and because there are only two compartments for this temperature, no feasible assignment is found. A different tie breaking rule could have been more suitable for this example.
Even though not all feasible assignments were found, for 88% of the load assignment problems solved by the heuristic, a feasible load assignment was found.

Figure 5: Conflict graph illustrating ties breaking when making non-conflict clique



Overall, it can be seen that the instances constructed from the Solomon $C102$ instance have been more difficult to solve correctly by our heuristic. This could be explained by the fact that $C102$ has a short scheduling horizon, where few customers can be visited by the same vehicle, whereas $RC207$ and $R205$ have a longer scheduling horizon. It could be that due to the different products contained in the different customer orders, the vehicles are not filled completely for the $RC207$ and $R205$ instances, resulting in an easier to solve load assignment problem.
However, even though the load assignment problem is not solved in an exact manner with the solveHeuristic experiments, the final solution is only for one out of 48 cases better with the solveExact experiments. Possibly, because more iterations are required and CPLEX solves the load assignment sub-problem too slowly. However, this also happens many times because solving the load assignment in an exact manner is not always necessarily the best.

Results on the instances from the retail company In Table 9 the results for the 12 instances of the retail company are shown for the experiments solveHeuristic and solveExact.

Table 9: Results of instances from the retail company for experiments solveHeuristic and solveExact with setting nonMixedOrdersPlannedTogether

	solveHeuristic			solveExact		
instance	distance	# vehicles	runtime(s)	distance	# vehicles	runtime(s)
$W2$	71609.18	104	1107	71609.18	104	1496
$W3$	65398.64	93	843	65398.64	93	1321
$W4$	68076.89	99	953	68076.89	99	1274
$W5$	69008.29	101	930	69008.29	101	1437
$W8$	**32165.41**	158	3097	32285.99	159	3743
$W9$	30390.27	158	3067	**30160.89**	157	3710
$W10$	**29357.32**	150	2575	29582.74	151	3125
$W12$	30328.79	150	2744	**30292.08**	152	3286
$W14$	20427.09	150	2483	20427.09	150	3136
$W15$	**22125.58**	165	3131	22145.60	165	3847
$W16$	23730.54	172	3918	**23632.37**	173	4555
$W18$	**20185.53**	147	2493	20527.27	149	3153
total	482803.52	1647	27342	483147.02	1653	34082

Notes. Results of instances from the retail company for the experiments solveHeuristic and solveExact. In bold is the best distance obtained from the experiments, if the solutions are not equal.

The runtime is for all instances the smallest when the heuristic is used in the load assignment sub-problem, compared to using CPLEX in the sub-problem, which is a result that we also found for the constructed instances. For instance $W3$ the runtime is 56.70% higher when using solveExact compared to solveHeuristic. The average increase in runtime over all instances when using CPLEX is 29.56%. The percentage differences in runtimes between the solveHeuristic and solveExact experiments is smaller for these instances than for the constructed instances. This can be explained by the fact that the load assignment problems are not so challenging for the instances of the retail company, because there are at most two products per temperature and, thus, at most two products per temperature contaminate. For instances $W2$, $W3$, $W4$, $W5$ and $W14$ the final solution obtained with both approaches is the same. For instances $W8$, $W10$, $W15$ and $W18$ the total distance travelled is lower for experiment solveHeuristic than for experiment solveExact, with equal or lower number of vehicles used. A better solution is obtained with solveExact for instances $W9$, $W12$ and $W16$, however, twice with a higher number of vehicles. The total distance travelled and the number of used vehicles is the lowest for experiment solveHeuristic.

Table 10: Results of instances from the retail company for experiments solveHeuristicEvaluateExact with setting nonMixedOrdersPlannedTogether

	solveHeuristic			exact evaluations
instance	# problems	# feasible	# classified as infeasible	# incorrect results heuristic
$W2$	1737113	452936	1284177	0
$W3$	1613097	452819	1160278	0
$W4$	1693562	452982	1240580	0
$W5$	1683940	452866	1231074	0
$W8$	1713637	453137	1260500	7
$W9$	1857191	453075	1404116	15
$W10$	1792752	452758	1339994	12
$W12$	1755353	452947	1302406	3
$W14$	1619439	453068	1166371	0
$W15$	1722622	453209	1269413	7
$W16$	1712418	452983	1259435	7
$W18$	1774040	452841	1321199	4

Notes. Results of instances from the retail company for the experiments solveHeuristicEvaluateExact using setting nonMixedOrdersPlannedTogether. The third and fourth column denote the number of load assignment problems solved and the number of times a feasible configuration was found, respectively. The fifth column shows how often no feasible configuration was found with the heuristic and the sixth column shows how many times this result was incorrect.

Table 10 shows for the same set of instances the results of the solveHeuristicEvaluateExact experiments. On average only 26% of the times the load assignment problem is solved, a feasible configuration is found. The other 74%, we evaluate if there indeed did not exist a feasible solution. In total only 55 of these evaluations resulted in a feasible load assignment. The final solution is, therefore, different with the seven instances for which this happened, as can be seen from Table 9.
We take a closer look at why the heuristic did not find the feasible solution, while CPLEX did provide us with a feasible solution. This happened when the total size of the suborders that had to be assigned was (almost) equal to the total capacity of the compartments. For example, for instance $W18$, the load assignment problem is not solved correctly by our heuristic four times. These four times, our heuristic failed in the sub-problem for the chilled temperature, for the configuration which consists of two compartments with this temperature regime with capacities 560 and 420. The heuristic failed twice when the suborders had a total size of 976 and twice when their size was 980, which is exactly the capacity of the compartments together. In Table 11 we provide an example of a feasible assignment to the two compartments that was found by CPLEX and not by our heuristic. The suborders are all of the same product type, but have different sizes.

Table 11: Example of feasible load assignment not found by the proposed heuristic

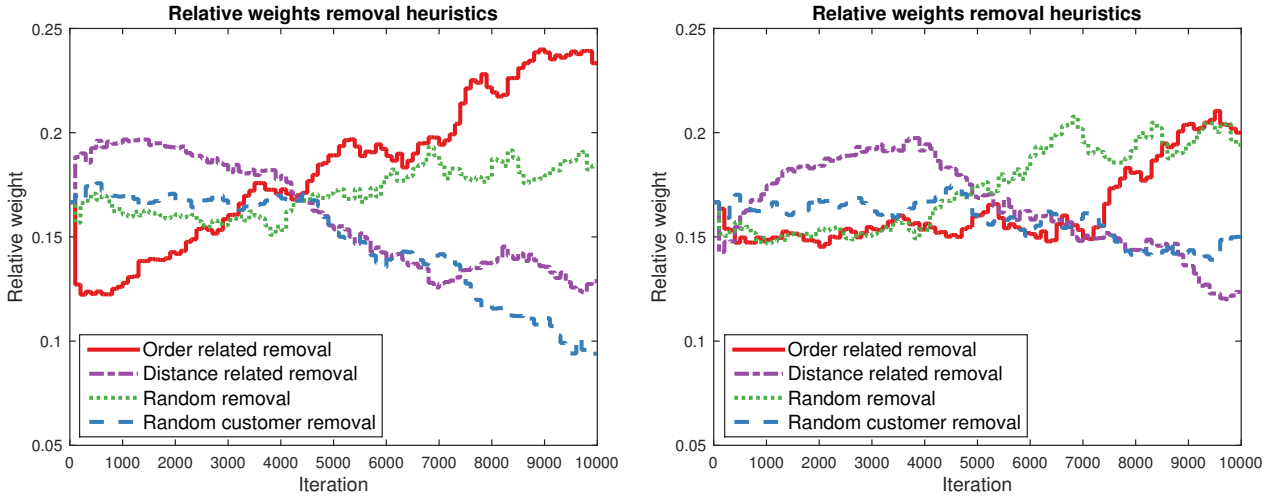
suborder size									
44.0	71.0	71.0	15.0	72.0	71.0	72.0	72.0	72.0	= 560
71.0	69.0	33.0	34.0	72.0	70.0	71.0			= 420

Notes. Example of feasible load assignment found by CPLEX and not found by the proposed heuristic. The suborders consist of one chilled product and have to be assigned to chilled compartments with capacities 560 and 420.

Even though the heuristic does not always give the correct solution, overall better solutions are obtained by solving the sub-problem with the heuristic. So by not accepting some insertions, the solution is not necessarily worse.

Evaluation removal heuristics To evaluate how well our presented order related removal heuristic works, we analyze the weights of the removal heuristics over all iterations.

The weights are updated after each segment of 100 iterations. As the ALNS uses in total six removal heuristics and the weights of these heuristics are initially equal, all heuristics have a relative weight of 0.167 at iteration 0 up to 100.
We present an example of the course of the relative weights of the removal heuristics, namely for instance $W14$, but a roughly similar pattern could be found for multiple instances.
Figure 6 shows the progress of the weights relative to one another, where in Figure 6a the setting nonMixedOrdersPlannedTogether and in Figure 6b the setting mixedOrdersPlanned-Together is applied. The time related removal heuristic and the worst removal heuristic show the least fluctuations, also for instance $W14$. These heuristics have a relative weight throughout the algorithm that is fairly equal to the start weight and are left out of the figure to keep it better readable.

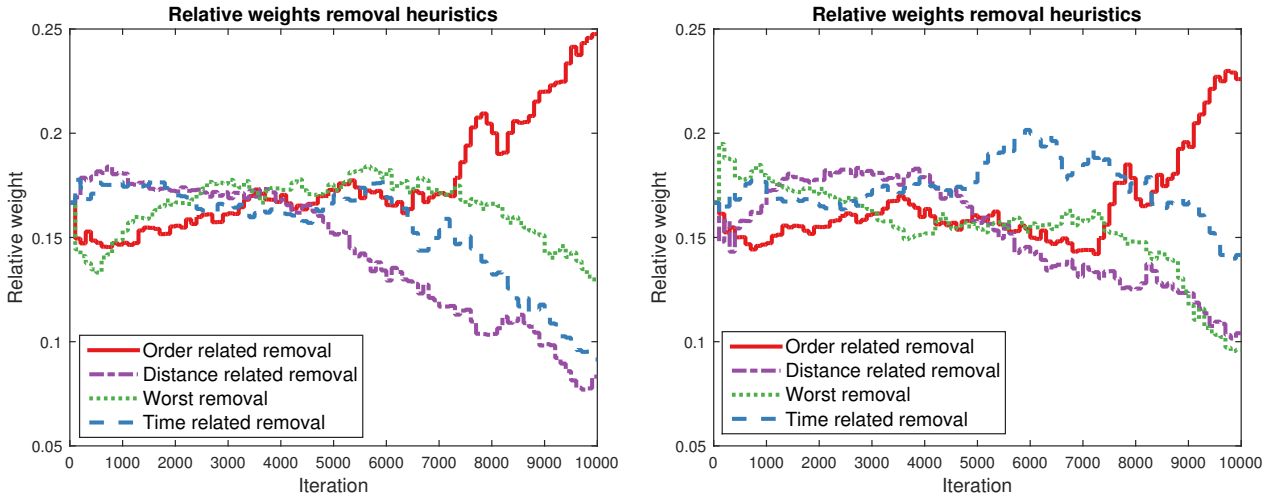


(a) nonMixedOrdersPlannedTogether (b) mixedOrdersPlannedTogether

Figure 6: Relative weights of removal heuristics for instance $W14$ and experiment solveHeuristic

In both figures, it can be seen that the distance related removal heuristic performs best in the first half of the algorithm and that its relative weight decreases much in the second half. Interchanging the orders from customers that are close to each other between their routes could first be beneficial, whereas towards the end other factors could play a role. For example, it could be that vehicles are more efficiently filled in later iterations, making it more beneficial to look at other factors.
The random removal method shows to actually work better in the second half. This shows that randomization might be required to make improvements at later iterations in the algorithm.
For both the nonMixedOrdersPlannedTogether setting as well as for the mixedOrders-PlannedTogether setting, the order related removal heuristic has the highest relative weight at the end of the algorithm. For the nonMixOrdersPlannedTogether case, this heuristic has relatively low weights in the first few segments, but its relative weights show an ascending pattern in the later iterations. Note that the weight in a segment is only partly directly based on the success in the previous segment and partly on the weight in the previous segment. Hence, the weights can only increase gradually.
For the case in which orders only consist of one product type, the order related removal

heuristic achieves a higher relative weight at the end of the algorithm compared to the final relative weight with the mixedOrdersPlannedTogether setting. This could be because there is more overlap in terms of product types between the orders in the latter case. Because the instances only consist of five products, many orders will consist of the same products. The relatedness measure is then less strong than when having orders with more difference between the products they consist of.



(a) $S25$ (b) $S48$

Figure 7: Relative weights of removal heuristics for instances $S25$ and $S48$ and experiment solveHeuristic

We also give examples of the relative weights for two different constructed instances in Figure 7. Here we focus on comparing the order related heuristic to the distance and time related removal heuristics and the worst removal heuristic, as there is a similar competition between the random removal and the order related removal as in Figure 6.
In contrast to what was visible for instance $W14$, the relative weights of the worst removal heuristic and the time related removal heuristic show a decreasing pattern towards the end of the algorithm, which is different for $S25$ and $S48$. The order related removal heuristic shows to perform worst up to approximately iteration 3500, except for the first few segments, and improves in later iterations.
It can be seen that the relative weights of the order related removal heuristic at the end of the algorithm are higher than in Figure 6b. We believe that this is the case because the constructed instances consist of more product types and are, thus, more diverse.

7.3.3 Effects of equipping vehicles with bulkheads

In this section we analyze the effects of equipping vehicles with bulkheads and solving the load assignment problem as additional restriction in the ALNS. To do so, we solve the load assignment sub-problems with the proposed heuristic. We compare the nonMixedOrders-PlannedTogether results with the different proposed settings: mixedOrdersPlannedTogether, nonMixedOrdersPlannedSeparate, suborderOrdersPlannedTogether.
In Table 12 the results of the mixedOrdersPlannedTogether are shown together with the previously shown results of the nonMixedOrdersPlannedTogether setting. For all instances a better final solution is obtained when changing the compostion of the orders with the

method proposed in Section 6.2. The method that changes the compositon of the orders, thus, has its desired effect on the final solution.

Table 12: Results of solveHeuristic experiments with settings nonMixedOrdersPlannedTogether and mixedOrdersPlannedTogether

	nonMixedOrdersPlannedTogether				mixedOrdersPlannedTogether			
instance	distance	# vehicles	# orders	runtime(s)	distance	# vehicles	# orders	runtime(s)
$W2$	71609.18	104	398	1107	**70453.52**	104	392	1146
$W3$	65398.64	93	393	843	**65333.28**	92	380	841
$W4$	68076.89	99	408	953	**66489.65**	98	392	912
$W5$	69008.29	101	393	930	**67703.51**	101	366	921
$W8$	32165.41	158	536	3097	**31459.61**	161	507	2798
$W9$	30390.27	158	510	3067	**29724.73**	153	495	2884
$W10$	29357.32	150	485	2575	**28687.37**	148	446	2331
$W12$	30328.79	150	531	2744	**30017.9**	153	520	2585
$W14$	20427.09	150	528	2483	**20090.37**	149	501	2527
$W15$	22125.58	165	528	3131	**21663.05**	164	499	3098
$W16$	23730.54	172	543	3918	**23190.00**	176	521	3750
$W18$	20185.53	147	524	2493	**19994.77**	149	498	2502
total	482803.53	1647	5777	27341	474807.80	1648	5517	26295

Notes. Results of instances from the retail company for the experiments solveHeuristic with settings nonMixedOrdersPlannedTogether and mixedOrdersPlannedTogether. In bold is the best distance obtained from the experiments, if the solutions are not equal.

Table 13 shows the results of the same instances for the nonMixedOrdersPlannedSeparate setting, in which the orders are planned per product type. The table shows the summed distance, number of used vehicles, number of planned orders and runtime over the five product types.

Table 13: Results of solveHeuristic experiments with setting nonMixedOrdersPlannedSeparate summed over five runs

instance	distance	# vehicles	# orders	runtime(s)
$W2$	72241.69	97	398	115
$W3$	67531.83	88	393	106
$W4$	**67555.17**	94	408	97
$W5$	70473.35	96	393	94
$W8$	**30877.92**	157	536	276
$W9$	**28906.98**	151	510	285
$W10$	**28791.22**	150	485	237
$W12$	**29309.07**	153	531	238
$W14$	**20270.51**	140	528	198
$W15$	**21731.28**	156	528	259
$W16$	**22902.66**	161	543	289
$W18$	**20177.82**	138	524	200
total	480769.50	1581	5777	2394

Notes. Results of instances from the retail company for the experiments solveHeuristic with setting nonMixedOrdersPlannedSeparate. The distances, number of used vehicles, number planned orders and runtime in seconds are summed over the five runs. The distances in bold indicate that a better solution was obtained than for the nonMixedOrdersPlannedTogether setting.

Because the number of orders per run is much lower, on average 96, and no load assignment problem needs to be solved, the total runtime is much lower than when solving all orders in one run with setting nonMixedOrdersPlannedTogether.

For eight instances the setting mixedOrdersPlannedTogether resulted in less distance travelled in total than with the nonMixedOrdersPlannedSeparate setting.
The results for the nonMixedOrdersPlannedTogether setting are only for three instances better than the nonMixedOrdersPlannedSeparate setting. We believe that this is due to the fact that for such small problem, that is the problem of routing one product type, the routing algorithm finds good solutions faster. The combined problem solved with nonMixedOrdersPlannedTogether, might require more iterations to obtain better solutions. Moreover, the number of visits to each customer will be larger when planning all products separately, as at least one visit is required for each delivered product type. The minimization of the number of visits to the customer is not an explicit objective in our problem, but this should also be taken into account when determining the best solution approach. It might be very important for the retail company to lower the inconvenience for the customers of different delivery moments.
It might be beneficial to first plan orders of different products separately and take the routes of these five runs together as an initial solution for the ALNS with setting nonMixedOrdersPlannedTogether. To see if this is beneficial, we run the ALNS with setting nonMixedOrdersPlannedTogether with the routes obtained from the nonMixedOrdersPlannedSeparate experiments as initial solution and allow the total runtime of this two-phase approach to be maximum the runtime reported in Table 12. That is, we evaluate if better solutions can be obtained without additional runtime.

Table 14: Results of solveHeuristic with setting nonMixedOrdersPlannedTogether and initial solution from nonMixedOrdersPlannedSeparate

instance	distance	# vehicles
$W2$	71636.06	103
$W3$	**65048.56**	95
$W4$	67006.25	100
$W5$	69283.94	101
$W8$	**30665.19**	156
$W9$	28906.98	151
$W10$	28712.09	150
$W12$	**29303.29**	152
$W14$	20201.48	149
$W15$	**21731.28**	156
$W16$	**22899.62**	162
$W18$	20177.82	138
total	475572.6	1613

Notes. Results of instances from the retail company for the experiments solveHeuristic with setting nonMixedOrdersPlannedSeparate, where the results from Table 13 form the initial solution. In bold the solutions that are better than for any of the other settings.

These results can be found in Table 14. The solutions from Table 13 have been improved in the second phase, with the nonMixedOrdersPlannedTogether setting, for nine instances.
For two instances, the solution without using the initial input with the nonMixedOrdersPlannedSeparate setting is still better than when using this start solution.

In total, for six instances this two phase approach results in overall best solutions, whereas the other six best solutions are obtained with the mixedOrdersPlannedTogether settings. The reason that the mixedOrdersPlannedTogether setting does not always show to be better, could be that more iterations are required for such a large problem. It might be beneficial to also split the mixed orders into different smaller problems, solve these smaller problems and use the output as initial solution for solving the entire problem with the mixedOrdersPlannedTogether setting. We leave it for future research to determine a suitable method to split the mixed orders in different smaller problems which can be solved better separately and later on be used in the combined problem to obtain overall better results.

Table 15: Results of solveHeuristic experiments with setting suborderOrdersPlannedTogether

	suborderOrdersPlannedTogether				best in all experiments
instance	distance	# vehicles	# orders	runtime(s)	distance
$W3$	81116.59	100	2623	2033	65048.56
$W10$	37507.61	154	4267	10248	28687.37
$W18$	25958.6	139	4091	8011	19994.77

Notes. Results of instances from the retail company for the experiments solveHeuristic with setting suborderOrdersPlannedTogether, comparing the objective value with the best objective value obtained in the experiments.

For three instances we solve the solveHeuristic experiments with setting suborderOrdersPlannedTogether. The results are presented in Table 15. As expected, the runtime increases much when planning all solutions together. Moreover, as the solutions are worse than for the nonMixedOrdersPlannedTogether and mixedOrdersPlannedTogether settings, this shows that more iterations are needed to obtain good solutions.
As explained earlier, it might be important for the retail company to have a lower number of stops in total over all routes to have less inconvenience for the customers. Therefore, we analyze for instances $W3$, $W10$ and $W18$ how many stops there are in total. The results can be seen in Table 16.
As expected, the total number of visits when planning all products separately is higher than when planning the orders all together. Moreover, planning all suborders as separate orders, results in a very high number of customer visits. On average each customer is visited 7 times for instances $W3$ and $W10$ and 5 times for instance $W18$. For all analyzed instances, the number of customer visits is the lowest with the mixedOrdersPlannedTogether setting.

Table 16: Number of customer visits with the different settings for experiments solveHeuristic

setting	$W3$	$W10$	$W18$
nonMixedOrdersPlannedTogether	281	376	380
mixedOrdersPlannedTogether	273	352	355
nonMixedOrdersPlannedSeparate	393	485	524
suborderOrdersPlannedTogether	709	1062	721

Notes. Total number of customer visits for all different settings with experiments solveHeuristic.

8 Conclusion

In this thesis we introduced the extension of the VRPTW in which each customer requests many suborders of different sizes and different types of products, which need to be transported in a certain temperature regime. This problem is inspired by a real-life case of a large retail company for which ORTEC provides the optimization software. The vehicles can be equipped with bulkheads according to some pre-defined configurations, to obtain compartments with possibly different temperature regimes. The suborders requested by a customer are consolidated into groups, which we call orders. Each order must be delivered by one vehicle to the customer. Each suborder contained in this order must be transported in only one of the compartments of that vehicle, such that it is transported within the required temperature and no contaminating suborders are in the same compartment.
We proposed to solve this problem using an ALNS framework, which uses the ruin and recreate principle for creating the routes. We recommend to add the order related removal heuristic, which is specifically designed for this problem, as this heuristic has shown to work well.
We have developed an algorithm to solve the so-called load assignment problem. This problem has to be solved as an additional check for feasibility when inserting orders in routes. This check determines whether or not a feasible vehicle configuration exists, such that all suborders contained in the orders assigned to the vehicles are feasibly assigned to one of the compartments in the vehicle.
We have noticed that this load assignment problem can be solved per configuration and per temperature separately and we split the problem accordingly. For each configuration a sub-problem should be solved per temperature. The problem is easy when only one compartment of the temperature is present in the configuration, but becomes more difficult if there are more. We propose a heuristic for the so-called load assignment sub-problem, which is the load assignment problem for one configuration and one temperature for which there are multiple compartments. The heuristic makes use of the concept of cliques in the conflict graphs and non-conflict graphs, to identify groups of suborders that can be inserted together in the same compartment. The heuristic proposed in this thesis is able to handle all types of conflict graphs, which makes it widely applicable. The proposed heuristic is compared to solving the presented ILP formulation of the sub-problem in an exact manner, using CPLEX.
The test set consists of instances obtained from a large retail company and instances that were constructed in this thesis, based on the Solomon benchmark instances.
The results show, that solving the sub-problem in an exact manner is not necessarily better, while resulting in larger runtime for all instances.
From the constructed instances, only one instance has a better final solution obtained with CPLEX, within the runtime limit. When considering only the instances for which both methods performed the same number of ALNS iterations, all the instances were solved better or equally good with the heuristic.
The instances from the retail company, were generally better solved with the heuristic. No additional runtime limit was necessary, so an equal number of iterations is performed with both methods. For four instances better final solutions were obtained with the heuristic, whereas this was only three instances for solving with CPLEX. This shows also for these instances that solving the load assignment problem in an exact manner does not necessarily lead to better results.
The load assignment heuristic mainly has difficulty with finding the feasible assignment when the suborders that have to be assigned have a size that is (almost) equal to the

total compartment capacities. Also, the load assignment problem sometimes is not solved correctly by the heuristic due to the assignment of product types to the non-conflict cliques. However, within the time limit, the final solution for these instances was still better than when solving the load assignment sub-problems with CPLEX.
Initially, we considered the suborders to be consolidated into orders that each contain one product type. Consolidation, which is also applied by ORTEC, is required to obtain good solutions in reasonable time and limit the number of visits to each customer.
Later, we have presented a method which changes the composition of these customer orders, to have a more efficient filling of the vehicles and possibly less customer visits. This method has shown to improve the solution for all instances with a lower distance travelled and has shown to deliver the same suborders in less visits to the customers.
Moreover, we provided some insights for the retail company in the effects of having vehicles with multiple compartments and, thus, being able to plan all product types together. For most instances, it has shown beneficial to first solve the smaller problems per product type and take the routes obtained by these experiments as input for improving the routes of all orders together.
It would be interesting to investigate the benefits of splitting the problem with mixed orders, to first optimize the routes for these sub-problems and then take the routes of these problems as input for the problem in which all orders are planned together. In future research a method could be developed to determine a good manner of splitting the problem.
In this work, we did not focus on finding a more efficient exact solution method for the load assignment sub-problem. Although this might speed up the runtime for the overall problem, the presented results do not show that an exact method is required for solving it.
Additionally, the load assignment heuristic proposed in this thesis could still be further improved on efficiency, to lower the runtime.
Finally, in this study we took into account one capacity dimension. The inclusion of different capacity dimensions, which introduces another level of difficulty to the problem, would also be interesting to study in future research.

References

Ambrosino, D., & Sciomachen, A. (2007). A food distribution network problem: a case study. *IMA Journal of Management Mathematics*, *18*(1), 33–53.

Archetti, C., Bianchessi, N., & Speranza, M. G. (2015). A branch-price-and-cut algorithm for the commodity constrained split delivery vehicle routing problem. *Computers & Operations Research*, *64*, 1–10.

Archetti, C., Campbell, A. M., & Speranza, M. G. (2014). Multicommodity vs. single-commodity routing. *Transportation Science*, *50*(2), 461–472.

Archetti, C., & Speranza, M. G. (2012). Vehicle routing problems with split deliveries. *International transactions in operational research*, *19*(1-2), 3–22.

Archetti, C., Speranza, M. G., & Hertz, A. (2006). A tabu search algorithm for the split delivery vehicle routing problem. *Transportation science*, *40*(1), 64–73.

Archetti, C., Speranza, M. G., & Savelsbergh, M. W. (2008). An optimization-based heuristic for the split delivery vehicle routing problem. *Transportation Science*, *42*(1), 22–31.

Baldacci, R., Mingozzi, A., & Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, *218*(1), 1–6.

Bomze, I. M., Budinich, M., Pardalos, P. M., & Pelillo, M. (1999). The maximum clique problem. In *Handbook of combinatorial optimization* (pp. 1–74). Springer.

Caramia, M., & Guerriero, F. (2010). A milk collection problem with incompatibility constraints. *Interfaces*, *40*(2), 130–143.

Coelho, L. C., & Laporte, G. (2015). Classification, models and exact algorithms for multi-compartment delivery problems. *European Journal of Operational Research*, *242*(3), 854–864.

Cordeau, J.-F., Gendreau, M., Hertz, A., Laporte, G., & Sormany, J.-S. (2005). New heuristics for the vehicle routing problem. In *Logistics systems: design and optimization* (pp. 279–297). Springer.

Cornillier, F., Boctor, F., Laporte, G., & Renaud, J. (2008). An exact algorithm for the petrol station replenishment problem. *Journal of the Operational Research Society*, *59*(5), 607–615.

Derigs, U., Gottlieb, J., Kalkoff, J., Piesche, M., Rothlauf, F., & Vogel, U. (2011). Vehicle routing with compartments: applications, modelling and heuristics. *OR spectrum*, *33*(4), 885–914.

Derigs, U., Li, B., & Vogel, U. (2010). Local search-based metaheuristics for the split delivery vehicle routing problem. *Journal of the Operational Research Society*, *61*(9), 1356–1364.

Dror, M., & Trudeau, P. (1990). Split delivery routing. *Naval Research Logistics (NRL)*, *37*(3), 383–402.

Eglese, R. W., Mercer, A., & Sohrabi, B. (2005). The grocery superstore vehicle scheduling problem. *Journal of the Operational Research Society*, *56*(8), 902–911.

Epstein, L., Favrholdt, L. M., & Levin, A. (2011). Online variable-sized bin packing with conflicts. *Discrete Optimization*, *8*(2), 333–343.

Fagerholt, K., & Christiansen, M. (2000). A combined ship scheduling and allocation problem. *Journal of the operational research society*, *51*(7), 834–842.

Gendreau, M., Hertz, A., & Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, *40*(6), 1086–1094.

Gendreau, M., Laporte, G., & Semet, F. (2004). Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research*, *31*(3), 347–358.

Gendreau, M., Soriano, P., & Salvail, L. (1993). Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, *41*(4), 385–403.

Gulczynski, D., Golden, B., & Wasil, E. (2010). The split delivery vehicle routing problem with minimum delivery amounts. *Transportation Research Part E: Logistics and Transportation Review*, *46*(5), 612–626.

Haouari, M., & Serairi, M. (2009). Heuristics for the variable sized bin-packing problem. *Computers & Operations Research*, *36*(10), 2877–2884.

Henke, T., Speranza, M. G., & Wäscher, G. (2015). The multi-compartment vehicle routing problem with flexible compartment sizes. *European Journal of Operational Research*, *246*(3), 730–743.

Henke, T., Speranza, M. G., & Wäscher, G. (2017). *A branch-and-cut algorithm for the multi-compartment vehicle routing problem with flexible compartment sizes* (Tech. Rep.). Otto-von-Guericke University Magdeburg, Faculty of Economics and Management.

Ho, S. C., & Haugland, D. (2004). A tabu search heuristic for the vehicle routing problem with time windows and split deliveries. *Computers & Operations Research*, *31*(12), 1947–1964.

Hvattum, L. M., Fagerholt, K., & Armentano, V. A. (2009). Tank allocation problems in maritime bulk shipping. *Computers & Operations Research*, *36*(11), 3051–3060.

Jansen, K. (1999). An approximation scheme for bin packing with conflicts. *Journal of combinatorial optimization*, *3*(4), 363–377.

Johnson, D. S. (1974). Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, *9*(3), 256–278.

Johnson, D. S., & Garey, M. R. (1985). A 7160 theorem for bin packing. *Journal of Complexity*, *1*(1), 65–106.

Karmarkar, N., & Karp, R. M. (1982). An efficient approximation scheme for the one-dimensional bin-packing problem. In *Foundations of computer science, 1982. sfcs'08. 23rd annual symposium on* (pp. 312–320).

Koch, H., Henke, T., & Wäscher, G. (2016). *A genetic algorithm for the multi-compartment vehicle routing problem with flexible compartment sizes*. Univ., Faculty of Economics and Management.

Maiza, M., Radjef, M. S., & Sais, L. (2016). Efficient lower bounds for packing problems in heterogeneous bins with conflicts constraint. In *Intelligent mathematics ii: Applied mathematics and approximation theory* (pp. 263–270). Springer.

Malaguti, E., Monaci, M., & Toth, P. (2008). A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, *20*(2), 302–316.

Mendoza, J. E., Castanier, B., Guéret, C., Medaglia, A. L., & Velasco, N. (2010). A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands. *Computers & Operations Research*, *37*(11), 1886–1898.

Moshref-Javadi, M., & Lee, S. (2016). The customer-centric, multi-commodity vehicle routing problem with split delivery. *Expert Systems with Applications*, *56*, 335–348.

Muritiba, A. E. F., Iori, M., Malaguti, E., & Toth, P. (2010). Algorithms for the bin packing problem with conflicts. *Informs Journal on computing*, *22*(3), 401–415.

Muyldermans, L., & Pang, G. (2010). On the benefits of co-collection: Experiments with a multi-compartment vehicle routing algorithm. *European Journal of Operational Research*, *206*(1), 93–103.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, *34*(8), 2403–2435.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, *40*(4), 455–472.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*.

SINTEF. (2008). *Solomon benchmark.* Retrieved from `https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/` (Accessed: 2017-05-05)

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, *35*(2), 254–265.

Sweeney, P. E., & Paternoster, E. R. (1992). Cutting and packing problems: a categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, *43*(7), 691–706.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, *231*(1), 1–21.

Yoshizaki, H. T. Y., et al. (2009). Scatter search for a real-life heterogeneous fleet vehicle routing problem with time windows and split deliveries in brazil. *European Journal of Operational Research*, *199*(3), 750–758.

A Appendix

A.1 Description of the instances

Table 17: Description of the instances from the retail company

instance	depot	weekday	# customers	# suborders	avg. suborder size	y	$K\|$
$W1$	A	Monday	98	2985	60.03	7.47	397
$W2$	A	Wednesday	97	2833	60.19	7.07	397
$W3$	A	Sunday	98	2623	58.83	6.67	397
$W4$	A	Saturday	98	2714	59.53	6.65	397
$W5$	A	Sunday	98	2802	59.88	7.08	397
$W6$	A	Tuesday	97	2964	61.63	7.93	397
$W7$	B	Wednesday	149	4220	59.97	8.39	375
$W8$	B	Friday	151	4511	59.65	8.26	375
$W9$	B	Tuesday	149	4440	59.04	8.50	375
$W10$	B	Wednesday	149	4267	59.84	8.71	375
$W11$	B	Thursday	148	4386	59.57	8.32	375
$W12$	B	Friday	150	4329	59.03	8.15	375
$W13$	C	Saturday	146	4157	60.61	7.50	375
$W14$	C	Sunday	146	4168	60.56	7.70	375
$W15$	C	Tuesday	146	4570	61.77	7.91	375
$W16$	C	Thursday	145	4660	62.68	8.20	375
$W17$	C	Saturday	146	4264	61.36	7.54	375
$W18$	C	Monday	147	4091	61.29	7.49	375

Notes. Description of the instances obtained from the retail company. The column denoted by y specifies the average number of suborders in one order when the orders are constructed per product type and the column denoted by $|K|$ denotes the available number of vehicles.

Table 18: Specification constructed tuning instances

instance	Solomon instance	product probabilities	density	$c_{\max}$	#configurations
$S1$	$C201$	equal	0.25	3	low
$S2$	$R101$	equal	0.25	3	high
$S3$	$R201$	equal	0.25	5	low
$S4$	$R201$	equal	0.25	5	high
$S5$	$C101$	equal	0.75	3	low
$S6$	$R101$	equal	0.75	3	high
$S7$	$RC101$	equal	0.75	5	low
$S8$	$R101$	equal	0.75	5	high
$S9$	$R101$	different	0.25	3	low
$S10$	$RC201$	different	0.25	3	high
$S11$	$RC101$	different	0.25	5	low
$S12$	$R201$	different	0.25	5	high
$S13$	$RC201$	different	0.75	3	low
$S14$	$RC201$	different	0.75	3	high
$S15$	$R101$	different	0.75	5	low
$S16$	$C201$	different	0.75	5	high

Notes. Description of the input for constructing the tuning instances. The columns specify the used Solomon instance and which of the two probability sets is used for assigning temperatures to product types in the second and third column. The fourth column specifies what the expected density is of the conflict graph. The column denoted by $c_{\max}$ indicates the maximum number of compartments and the final column states whether a lower or higher number of configurations is generated.

Table 19: Specification constructed instances

instance	Solomon instance	product probabilities	density	$c_{\max}$	#configurations
$S17$	$RC207$	equal	0.25	3	low
$S18$	$RC207$	equal	0.25	3	high
$S19$	$RC207$	equal	0.25	5	low
$S20$	$RC207$	equal	0.25	5	high
$S21$	$RC207$	equal	0.75	3	low
$S22$	$RC207$	equal	0.75	3	high
$S23$	$RC207$	equal	0.75	5	low
$S24$	$RC207$	equal	0.75	5	high
$S25$	$RC207$	different	0.25	3	low
$S26$	$RC207$	different	0.25	3	high
$S27$	$RC207$	different	0.25	5	low
$S28$	$RC207$	different	0.25	5	high
$S29$	$RC207$	different	0.75	3	low
$S30$	$RC207$	different	0.75	3	high
$S31$	$RC207$	different	0.75	5	low
$S32$	$RC207$	different	0.75	5	high
$S33$	$C102$	equal	0.25	3	low
$S34$	$C102$	equal	0.25	3	high
$S35$	$C102$	equal	0.25	5	low
$S36$	$C102$	equal	0.25	5	high
$S37$	$C102$	equal	0.75	3	low
$S38$	$C102$	equal	0.75	3	high
$S39$	$C102$	equal	0.75	5	low
$S40$	$C102$	equal	0.75	5	high
$S41$	$C102$	different	0.25	3	low
$S42$	$C102$	different	0.25	3	high
$S43$	$C102$	different	0.25	5	low
$S44$	$C102$	different	0.25	5	high
$S45$	$C102$	different	0.75	3	low
$S46$	$C102$	different	0.75	3	high
$S47$	$C102$	different	0.75	5	low
$S48$	$C102$	different	0.75	5	high
$S49$	$R205$	equal	0.25	3	low
$S50$	$R205$	equal	0.25	3	high
$S51$	$R205$	equal	0.25	5	low
$S52$	$R205$	equal	0.25	5	high
$S53$	$R205$	equal	0.75	3	low
$S54$	$R205$	equal	0.75	3	high
$S55$	$R205$	equal	0.75	5	low
$S56$	$R205$	equal	0.75	5	high
$S57$	$R205$	different	0.25	3	low
$S58$	$R205$	different	0.25	3	high
$S59$	$R205$	different	0.25	5	low
$S60$	$R205$	different	0.25	5	high
$S61$	$R205$	different	0.75	3	low
$S62$	$R205$	different	0.75	3	high
$S63$	$R205$	different	0.75	5	low
$S64$	$R205$	different	0.75	5	high

Notes. Description of the input for constructing the instances. This table should be interpreted as Table 18

A.2 Additional tuning tables

Table 20: Results of tuning the method that changes the composition of orders

β	γ	$\delta(i,j)$	average deviation
1.0	y	$\frac{1}{3}d_{\hat{d}i}$	0.028
1.0	y	$\frac{1}{2}d_{\hat{d}i}$	0.037
1.0	y	$d_{\hat{d}i} + d_{\hat{d}j}$	0.032
1.5	y	$\frac{1}{3}d_{\hat{d}i}$	0.037
1.5	y	$\frac{1}{2}d_{\hat{d}i}$	0.033
1.5	y	$d_{\hat{d}i} + d_{\hat{d}j}$	0.020
2.0	y	$\frac{1}{3}d_{\hat{d}i}$	0.026
2.0	y	$\frac{1}{2}d_{\hat{d}i}$	0.028
2.0	y	$d_{\hat{d}i} + d_{\hat{d}j}$	0.027
1.0	$\frac{y}{2}$	$\frac{1}{3}d_{\hat{d}i}$	0.026
1.0	$\frac{y}{2}$	$\frac{1}{2}d_{\hat{d}i}$	0.027
1.0	$\frac{y}{2}$	$d_{\hat{d}i} + d_{\hat{d}j}$	0.019
1.5	$\frac{y}{2}$	$\frac{1}{3}d_{\hat{d}i}$	0.020
1.5	$\frac{y}{2}$	$\frac{1}{2}d_{\hat{d}i}$	0.022
1.5	$\frac{y}{2}$	$d_{\hat{d}i} + d_{\hat{d}j}$	0.017
2.0	$\frac{y}{2}$	$\frac{1}{3}d_{\hat{d}i}$	0.020
2.0	$\frac{y}{2}$	$\frac{1}{2}d_{\hat{d}i}$	0.018
2.0	$\frac{y}{2}$	$d_{\hat{d}i} + d_{\hat{d}j}$	**0.011**
1.0	$\frac{y}{1.5}$	$\frac{1}{3}d_{\hat{d}i}$	0.033
1.0	$\frac{y}{1.5}$	$\frac{1}{2}d_{\hat{d}i}$	0.029
1.0	$\frac{y}{1.5}$	$d_{\hat{d}i} + d_{\hat{d}j}$	0.015
1.5	$\frac{y}{1.5}$	$\frac{1}{3}d_{\hat{d}i}$	0.024
1.5	$\frac{y}{1.5}$	$\frac{1}{2}d_{\hat{d}i}$	0.023
1.5	$\frac{y}{1.5}$	$d_{\hat{d}i} + d_{\hat{d}j}$	0.017
2.0	$\frac{y}{1.5}$	$\frac{1}{3}d_{\hat{d}i}$	0.026
2.0	$\frac{y}{1.5}$	$\frac{1}{2}d_{\hat{d}i}$	0.019
2.0	$\frac{y}{1.5}$	$d_{\hat{d}i} + d_{\hat{d}j}$	0.012

Notes. Results of tuning the method that changes the composition of orders. The average deviation from the best solution obtained in this tuning experiment is denoted and the best parameter setting, with the smallest average deviation in bold, is chosen.

Table 21: Results of tuning the configuration sorting method for the constructed instances

α	κ	average deviation
-	n	0.020
0.1	10	0.203
0.1	20	0.045
0.1	40	0.026
0.2	10	0.040
0.2	20	0.012
0.2	40	0.038
0.5	10	0.028
0.5	20	**0.011**
0.5	40	0.018
0.8	10	0.030
0.8	20	0.027
0.8	40	0.027

Notes. Results of tuning the configuration sorting method for the constructed instances. The average deviation from the best solution obtained in this tuning experiment is denoted and the best parameter setting, with the smallest average deviation in bold, is chosen.

Table 22: Results of tuning the configuration sorting method for the instances from the retail company

α	κ	average deviation
-	n	**0.022**
0.1	10	0.251
0.1	20	0.158
0.1	40	0.105
0.2	10	0.181
0.2	20	0.071
0.2	40	0.044
0.5	10	0.176
0.5	20	0.101
0.5	40	0.068
0.8	10	0.155
0.8	20	0.079
0.8	40	0.082

Notes. Results of tuning the configuration sorting method for the instances from the retail company. The average deviation from the best solution obtained in this tuning experiment is denoted and the best parameter setting, with the smallest average deviation in bold, is chosen.