University of Twente

Master's Thesis

Validation of the loading constraints in vehicle routing problems

Author:
Mathijs W.H. Waegemakers

Supervisors:
Dr. Ir. M.R.K. Mes
Dr. Ir. J.M.J. Schutten
Dr. Ir. A.L. Kok

January 2016

This page is intentionally left blank.

Management summary

Motivation:
ORTEC develops a wide range of optimization tools for companies, one of which is a planning tool for distributors. The software uses vehicle routing heuristics that construct (partial) routes and uses local search algorithms to optimize the result. During construction and optimization, the algorithm checks, for every partial solution, a wide range of constraints on their feasibility. One of these are the load constraints, the constraints that ensure the orders of the route fit inside the used resource combination (truck). The vehicle routing algorithm calls a separate load validation algorithm to do this check.

The current load validation tool uses Integer Linear Programming (ILP) to solve the given problem of the vehicle routing algorithm. A major disadvantage of Integer Linear Programming is that its computation time can get extremely large for solving even a single problem. In the larger datasets of clients with hundred of orders, the number of calls can go up to several millions. This causes the computation to be long and unpredictable, which is undesirable for the customer. In addition, the number and type of restrictions in the current tool are still limited, and ORTEC wishes to include more restrictions in favour of their retail customers. These customers, frequently use the software with custom adjustments to account for their load restrictions. Customizations are more expensive for the customers, which has a negative effect on the competitiveness of ORTEC. We want to come up with a fast and reliable standard method to check the load constraints for retail customers in general.

Method:
First, we identify all relevant constraints of the retail customers, based on experts' views within ORTEC. We do not foresee any major changes in the future that can influence the load validation model. We base our model on a tree traversal algorithm, because of its flexibility regarding the constraints, the simplicity to explain it to third parties, and the ability to calculate an exact answer fast.

At the beginning of a load validation call, the tree traversal algorithm puts all orders in reversed order of delivery. One by one, the orders are inserted into the tree, always as a child of the previous order. A branch in the tree corresponds with the compartments of a resource combination, meaning the left branch equals the first compartment, the second the..., etc. After each insertion of an order into the tree, the algorithm tests the affected load constraints for the given situation at that point in the tree. If no feasible allocation can be found for an order in any of the compartments, we move up the tree to consider an assignment in another compartment. The algorithm continues until all orders are assigned, or if no feasible assignment can be found.

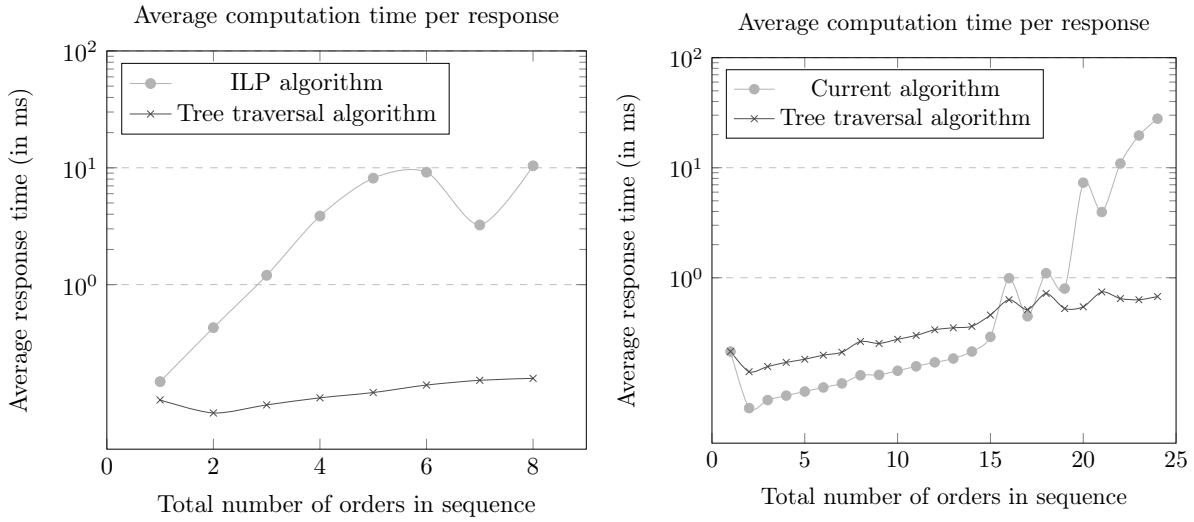
Additionally, we add two methods to speed up the process. First, we add two pre-checks that determine in a fast way if the route is infeasible. This check relatively easily in some cases prevents more expensive calls to the load validation algorithm. Second, we add two caching functions that store all calculated sequences during the optimization call, and checks if a sequence has been checked before. This way, the load validation algorithm does not have to repeat checks on duplicate sequences.

We use two datasets of actual retail customers of ORTEC to verify and test our algorithms. The two retailers have been anonymized, and we will refer to them as retailer 1 and retailer 2. Retailer 1 is a large Australian retailer, while retailer 2 is a large Russian retailer. The two datasets differ from each other, which makes impossible to directly compare the results between them. However, we have access

to the original algorithms, which gives us the ability to compare our algorithm with the current practices within ORTEC.

Results:

Figures 1 and 2 summarize the main results of the research. Figure 1 shows the computation time of the current and the tree traversal algorithm. We see that the tree traversal algorithm is faster and has a stable performance compared to the current algorithms, for both datasets. When the routes get longer and the complexity increases, the tree traversal algorithm is still able to calculate the feasibility equally fast.



(a) Results based on the Retailer 1 dataset.

(b) Results based on the Retailer 2 dataset.

FIGURE 1: The average computation time per response compared to the length of the route.

Figure 2 shows the percentage of calls that are prevented using the pre-checks and caching function. We notice that the pre-checks are not effective when the caching functions are active, as the number of calls prevented is relatively low. When the caching functions are disabled, we reach percentages of 76.8% and 82.9% of the calls that get prevented by the pre-checks. The caching functions reach percentages of 75% and 97.5% of the calls to the load validation algorithm can be prevented. This speeds up the load validation procedure.

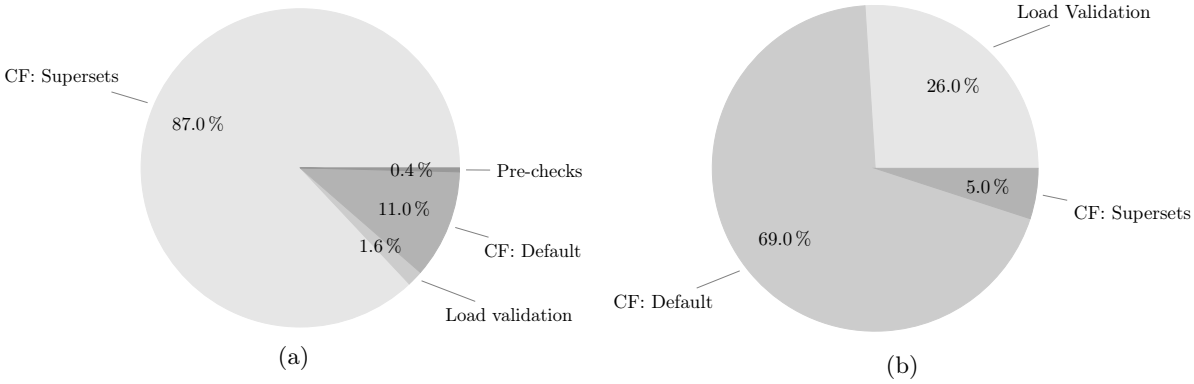


FIGURE 2: Percentage breakdown of the different functions. All Caching Functions (CF) cancel a large part of the calls to the load validation algorithm. (a): Retailer 1 dataset. (b): Retailer 2 dataset.

Recommendations:

We recommend to implement the caching functions into the current software of ORTEC. Because the caching functions are based on the principle of not calculating something twice, it is easy to implement, even if load validation is done with different algorithms than the tree traversal algorithm. The results of the pre-checks vary, so we recommended to first do additional research, before implementing it product wide. It is not directly clear if the tree traversal algorithm will be a success for all customers. The results show large improvements related to the total computation time, as well as more constant response times. We recommend to further research the use of the tree traversal algorithm, into the load validation algorithm, before full implementation.

This page is intentionally left blank.

Acknowledgements

Tuesday, 31st of March 2015, a normal day like any other at the office. Coffee, work, lunch, more work, more coffee, business as usual inside our bubble in Zoetermeer. Outside, the country was ravaged by a "code orange" storm, causing over 10 million Euro in material damage in The Netherlands alone. A remarkable thing, that day a total of 15 trucks were blown over because of the heavy wind, causing major delays on the Dutch roads. The weather forecast on the radio recommended to stay off the road with an empty truck, due to the risk of being blown away. However, as a driver you tend to forget your truck is full in the morning, while it is empty at the end of the afternoon. The topic of this thesis is the validation of loading constraints in vehicle routing problems, the focus is on how to properly load a truck. Hearing this on the news, made me wonder if I needed to extend my research to include weather forecasts as well.

This thesis is the result of a process starting back in November of 2014, with an e-mail conversation between Martijn Mes, Leendert Kok, Joaquim Gromicho and me. Almost a year later I am able to present the result of my research. Although I cannot mention everyone explicitly, I like to thank the following; ORTEC, for giving me the opportunity to graduate at a wonderful company. Leendert Kok, who trusted me to come up with a solution that both benefits ORTEC as it also functions as a great thesis topic. Chagiet Bloemendal, who had the grateful task to answer all my minor questions about almost everything when I just started my graduation assignment. She also helped me, together with Joep Olde Junnick and Jacob Kooijman, to understand the programming code of the software. From the university, Martijn Mes and Marco Schutten, who were a great help in reviewing my work, and taking the time to provide me with great feedback. Without all of you, this thesis would not have been a success.

Looking back on this period, I am satisfied with what I have accomplished. Looking at my thesis I can say that I produced some useful results to the retail case. I am even more pleased with everything I learned during last year, especially the C++ programming skills I developed. I cannot say that I am an expert now, but I have made a good start. Looking forward, I will be spending the next years in software development, giving me more than enough opportunity to develop my programming skills. However, my first task will be to start over, and focus on my second thesis dedicated to my master Civil Engineering.

This page is intentionally left blank.

Contents

Chapter 1

Introduction

This research is conducted at the Product Development department of ORTEC within the area of Transport & Logistics. ORTEC delivers optimization solutions in the field of Operation Research (OR), as well as consulting services in which OR techniques are applied at companies. One of the typical problems within OR is the Vehicle Routing Problem (VRP), which is a problem commonly faced by logistics enterprises. The VRP arises when a number of locations have to be visited, with a limited amount of resources. The standard definition of the VRP can be extended with constraints to match real-life problems, making it a so called rich-VRP [5].

ORTEC delivers software that is able to solve the VRP while taking into account many real-life constraints, and that computes a near optimal solution to this problem. The software is called ORTEC Routing & Dispatch (ORD), which allows companies to manage the distribution of goods with a fleet of vehicles. Company planners enter information on which goods have to be delivered to which customer, together with applicable boundary conditions. The output is a schedule showing which goods should be delivered by which vehicles at what time.

Another typical OR problem is called the Packing Problem (PP), which is the problem of packing items into a container. This problem is also faced by the logistics companies, when trying to load their vehicles. Both the VRP and the PP have been studied extensively over the years as separate problems, but more recently, some researchers are trying to solve these problems simultaneously. A common approach is to first find a solution to the VRP, and then check if all items can be properly packed within the truck. Both problems are therefore solved independently and in sequence. Figure 1.1 illustrates this process.

The current functioning of ORTEC Routing & Dispatch (ORD) corresponds to the schematic of Figure 1.1, meaning that the VRP and PP are solved independently of each other. The ORD uses different solving techniques for the VRP and PP. A solver is the part of software which contains the solving technique for a particular problem. The solver for the VRP uses construction and local search heuristics, while the solver for the PP uses Dynamic Programming (DP) and Mixed Integer Linear Programming (MILP). For a detailed explanation of both solvers, we refer the reader to Section 1.2. Currently, the computation time of the MILP explodes for larger instances, creating a limit on its use. On the contrary, DP gives a fast response, but is only suitable in straightforward instances with a limited amount of restrictions. This research aims to develop of a new approach of the solver for the PP, to ensure that larger instances of the VRP can be solved in a reasonable amount of time.

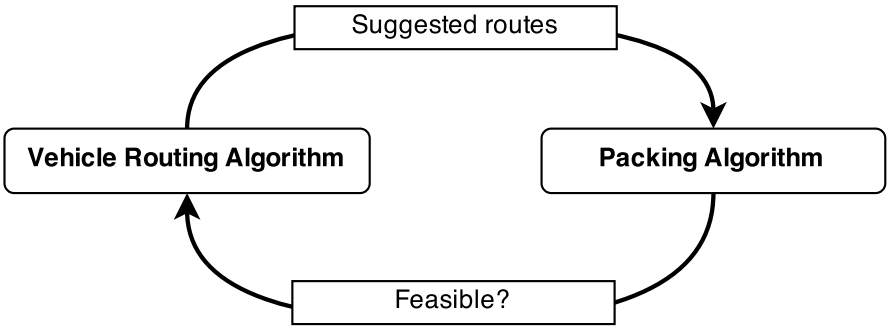


FIGURE 1.1: The interaction between the vehicle routing and packing algorithm.

In the next part of this chapter we provide an overview of our research. Section 1.1 explains the terminology we use throughout this research. Section 1.2 explains the current functioning of ORD, which is crucial to understand the context of our research. Section 1.3 describes the definition of the problem we want to tackle with this research. Sections 1.4, 1.5 and 1.6 describe the research goal, research scope, and research questions respectively.

1.1 Terminology

This thesis contains a lot of technical terminology. Part of this terminology is related to the current techniques used in the software of ORTEC. ORD uses the giant-tour representation (GTR) [1] framework to solve the Vehicle Routing Problem (VRP). The major advantage of this representation, is that it allows problems with complex restrictions to be modelled into one conceptual framework. The GTR represents the VRP as a graph in which multiple routes of different vehicles are chained together to one large tour with no beginning or ending. Figure 1.2 shows a transformation of multiple vehicle routes to form one GTR.

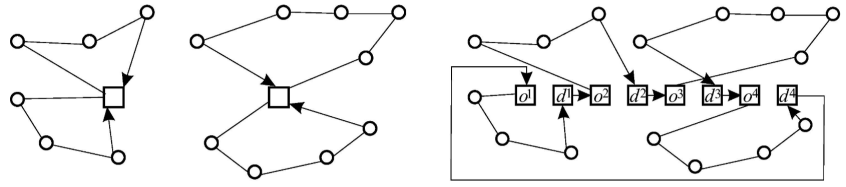


FIGURE 1.2: Left: A VRP with four routes and two depots. Right: The GTR of this VRP. The four routes are connected into one long chain. The destination nodes d^i of route i connect to the origin nodes o^{i+1} of route $i+1$. (Source: Funke et al. [1])

Even though most of the terminology is related to the VRP and the GTR, researchers tend to have different explanations for equal words, making the definition sometimes ambiguous. We notice that ORTEC as well has its own definitions related to transportation. To overcome this ambiguity problem, and to improve the readability of this thesis, we define the terminology as follows:

Order: An amount of a single type of product that has to be moved from a pick-up location to a delivery location. For instance, two pallets of frozen bakery. This definition is also known as an order line, but we retain ourselves solely using the word order(s).
Resource combination: We call a truck, a trailer, or a combination of both, a resource combination.

If we specifically refer to only the truck or trailer in a combination of both, we call it a resource.
Action: A relevant activity in a route. It is represented as a node in the GTR graph. Examples are a task and a travel.
Route: The sequence of actions performed by a resource or resource combination, starting and finishing at a depot.
Task: An action related to an order. In our case we are only interested in the pick-up and delivery task of an order.
Travel: The movement between two locations. A travel is also an example of an action and is included as a node in the GTR.
Requirement: A single documented definition of a demanded functionality of the software/algorithm.
Restriction/constraint: The mathematical representation of a requirement, as it is programmed into the software. We neglect the subtle difference between constraint and restriction, as it is not relevant for this thesis.
Load allocation: The way the orders are placed inside the resources. The load allocation constraints are defined as all constraints related to the placement of orders inside a resource or resource combination.
Call/response: When the vehicle routing algorithm wants to check if a sequence of orders is feasible, it sends a request to the load validation algorithm. We define this request as the "call" to the load validation algorithm. When the load validation algorithm comes up with an answer, it provides a response to the vehicle routing algorithm. We define this answer as the "response" to vehicle routing algorithm.

1.2 Context Analysis

In this section, we describe the current functioning of the software. Section 1.2.1 gives an overview of the interaction of the different components in ORD that are related to the VRP and PP. In Sections 1.2.2 and 1.2.3 we look in more detail level at the VRP and PP solvers of ORD.

1.2.1 Overview ORD

The ORD software is divided into multiple components. The components we focus on are the Vehicle Routing Algorithm and Packing Algorithm. These components are called COMTEC Vehicle Routing System (CVRS) and COMTEC Optimization System Loading Assignment (COPSLoad) respectively. A planner has two ways to build a schedule in ORD, this can be done manually or automatically. When a planner uses ORD to manually assign orders to a route, it allocates orders one by one into a route. After an allocation, it is the COMTEC distributed calculator system (CDCS) that checks on violations in all active constraints in a proposed route. To check if the load allocations are still valid, CDCS sends a request to COPSLoad for validation. If the planner wants to automatically plan, it selects all orders that need to be allocated over a set of routes and activates CVRS. CVRS searches automatically for a feasible solution, and then tries to optimize it. A schematic overview of the different components can be found in Figure 1.3.

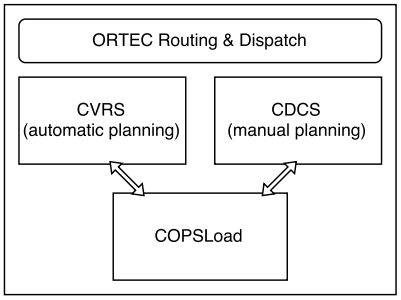


FIGURE 1.3: Schematic overview of the interaction with COPSLoad.

1.2.2 CVRS

In this section we give a brief explanation on how CVRS works. We need to understand the working of CVRS to be able to understand the context in which COPSLoad operates, because of their close relationship. We can roughly divide CVRS in two phases: a construction phase and an improvement phase [6]. The first phase constructs an initial solution, the second phase tries to improve it. The second phase searches iteratively for better solutions. The second phase terminates at a predefined criterion and then the best solution found is presented.

CVRS starts with a list of tasks, resource combinations, and empty routes. One resource combination is assigned to one route. In the construction phase, CVRS starts inserting the tasks to the routes. There are three possible strategies for the construction phase. These strategies are: parallel insertion, sequential insertion, and a sweep method. To ensure that the final solution is feasible, CVRS checks after each task insertion if all active constraints are still valid. This means for the load allocation constraints that CVRS sends a request to COPSLoad. If an inserted task results in an infeasible solution, it is removed from the route and another task is selected, depending on the strategy. The building phase continues until all tasks are assigned, or if no more tasks can be fitted in one of the routes.

The initial solution is further improved with a combination of optimization, deconstruction, and construction phases. The optimization phase uses one of five strategies: 2opt, swap, CROSS exchange, move, and a large neighbourhood move and swap. The deconstruction and construction phase run consecutively. The deconstruction phase consist of one of three strategies; random picking, worst-order picking and random neighbourhood picking. The construction phase uses the same strategies as when the initial solution is constructed. The decision which strategies to use, in which sequence, and how often, is taken by the consultants upon implementation of the software. Although, automating this process has recently been studied [7].

As in the initial construction phase, when a task gets (re)inserted, CVRS sends a request to check the feasibility of the load allocation. As CVRS is an iterative process running over a large amount of tasks, the amount of requests send to COPSLoad is enormous. After each optimization or (de)construction phase, CVRS tries to assign tasks that were not included into the initial solution. The rearrangement of tasks could have led to spare capacity in the routes. Figure 1.4 depicts an overview of the functioning of CVRS.

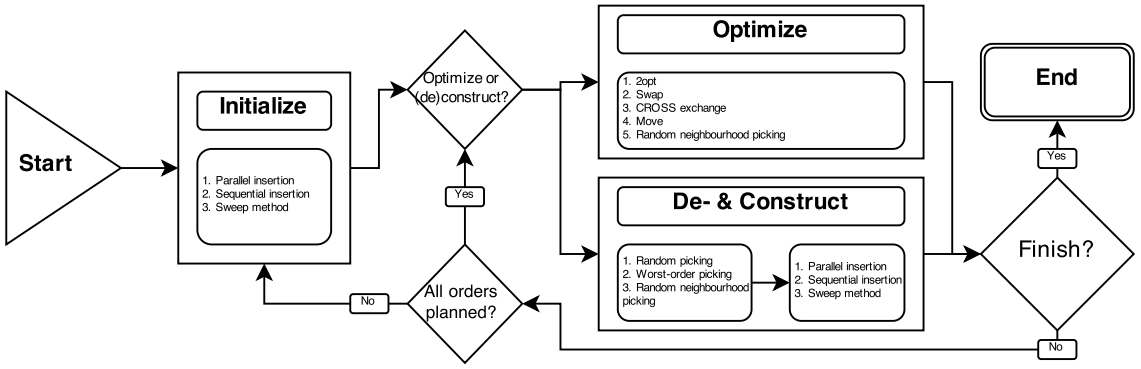


FIGURE 1.4: A schematic overview of the functioning of CVRS. The amount of iterations, as well as the strategies, are up to the consultant to decide.

1.2.3 COPSLoad

COMTEC Optimization System Loading Assignment (COPSLoad) is an independent component within ORD, which evaluates the load allocation constraints of a route. To evaluate these constraints, COPSLoad consist of a preprocessing, processing and post-processing phase. A schematic overview of the phases and decisions is given in Figure 1.5. This section describes these process steps in more detail.

The first phase consist of relatively easy checks, by calculating lower bounds of the constraints. For instance, if the capacity exceeds the moment the truck leaves the pick-up location, it is certain that the complete route is infeasible. This check is easily validated by checking the sum of the volume of all orders and the resource combination. Another example is to check if one of the orders in a route is not allowed in the resource combination. If one order in a route is not allowed to be loaded in the resource combination, the complete route is invalid. When the first phase fails, the algorithm terminates prematurely and immediately responds back to CDCS or CVRS. This saves time as it does not need to move on to the second phase.

However, if the first phase is successful the algorithm moves to the next phase. The second phase performs the complete validation of all active constraints. This process consist of two algorithms, one uses dynamic programming (DP) and the other MILP. The DP-solver is relatively fast, but the drawback is it only functions with a limited amount of constraints. The MILP solver is able solve problems where all constraints are active, but it is by far not as fast as CompartX. The second drawback is the computation time required by the MILP can explode, making it unpredictable if it will result an answer in a reasonable amount of time.

The current COPSLoad model contains 16 restrictions, 3 of which also are part of the pre-processing step as well. In addition, the model has 11 settings that have an effect on the functioning of the algorithm. We only focus on the case of orders with discrete amounts, so liquids (with continuous amounts) are outside the scope. The following list shows the requirements that the set of restrictions enables:

- Orders belong to groups that are allowed in certain compartments.
- Capacity on compartments.
- Capacity on resources.
- Minimum and maximum ullage of compartments.
- Weight distribution on the axles.

During preprocessing phase, three pre-checks are active. They check if the total capacity of the compartments and resources is sufficient, and if there are enough compartments available to hold the number of different (order) groups. In addition it is possible to fixate orders to compartments beforehand, and to prohibit orders being divided over multiple compartments.

At the start of the processing phase, COPSLoad decides if the DP-solver is suitable for the validation, or that the problem is too complicated and the MILP-solver should be used. Before the MILP-solver is used, a first attempt is made by the DP-solver. Because the DP-solver runs relatively fast, it is worth to first try this option. After the second phase, it is known if the solution will fit or not. The only job of the third phase is to optimize the solution where possible.

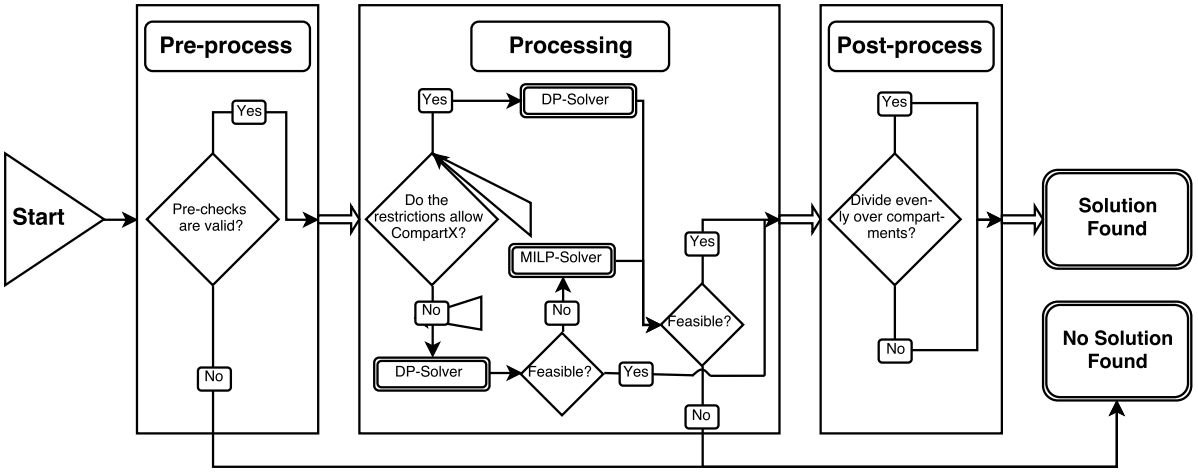


FIGURE 1.5: A schematic overview of the decision between DP-solver and the MILP-solver. Based on the functional guide of COPSLoad (ORTEC).

1.3 Problem Description

Section 1.2.1 described how the current software functions. We described how the VRP and the PP are solved as two separate problems. COPSLoad is able to solve the PP, but the computation time can be unpredictable and unreasonably long. It constructs an Integer Linear Programming (MILP) model for complex PP's with many active constraints. The model takes a relatively long time to run, and there is a risk no feasible solution will be found in reasonable time. This means that even though a feasible solution exists, the packing algorithm returns with no solution found. Because MILP models are complicated to solve, it is hard for ORTEC to explain their customers that the software does function the way it should. This lack of clarity together with the constant risk of running into extreme computation times, has led to the idea that the MILP model is not as suitable as previously thought.

In addition, COPSLoad is the subordinate algorithm compared to the route building algorithms. Because of this subordinate nature, no cooperation occurs between COPSLoad and the parts CVRS or CDCS. This lack of feedback makes that CVRS and CDCS constructs solutions that could be optimal for all other constraints, but not for the load allocation. This is a waste of time, if it somehow could have been known beforehand that the solution would fail on these load allocation constraints. It is the same the other way around. COPSLoad does not know if the proposed routes already (partially) have been validated, calculating possibly the exact same route multiple times.

Another problem is the lack of knowledge on how certain constraints affect the quality of the VRP solution. For instance, it is unknown if a Last-In-First-Out (LIFO) leads to an enormous detour, just because an order is not directly accessible from the back of the truck. If a company changes their policy to allow for mid-trip reallocation of orders, the final quality of the routing solution is affected. However, these impacts are currently unknown. Therefore, the consultants cannot advice their customers if it is better to adjust their load allocations requirements.

1.4 Research Goal

This research is subdivided into three successive goals. The first goal is to develop a comprehensive model that represents the functional needs of the customers concerning the load allocation constraints they face. This means we first need to define the characteristics of a customer, and afterwards the set of constraints that can be derived from these characteristics. Our second goal is to develop a fast algorithm that will validate the load allocation constraints of a trip. The developed algorithm should have a computation time that is acceptable for the clients. Our third goal is to compare the inclusion of some of the most comprehensive constraints, and study how they affect the computation time and the quality of the solution. This way, we can give recommendations to the customers on how to make decisions on their loading processes.

1.5 Research Scope

To limit the scope to which we develop the load allocation algorithm, we will limit ourselves to relevant constraints of retail distributors. Retail distributors are a major part of the customers of ORTEC and at the moment there is no software solution that supports all of the retailers simultaneously. The distribution to retailers has a potential for additional customers in the future. This creates the desire to develop an algorithm that performs effectively and efficiently in the situation of these retailers. In addition, retailers tend to have large problem sets that cause problems in the current version of COPSLoad. This makes it defiant for our research. Overall, retail distributors are the ideal customers to focus on in this research and we include them as a case study.

As described in Section 1.2, COPSLoad can be called by two different components. CDCS and CVRS are both components of ORD, in which CDCS requests COPSLoad in manual schedule building, and CVRS sends requests as it automatically searches for a VRP solution. When CVRS searches for a possible trip, it sends an enormous amount of requests to COPSLoad, as every partial solution needs to be checked individually. Therefore, it makes more sense to optimize COPSLoad for CVRS instead of CDCS, as there is more to gain. Besides, the response to the requests send by COPSLoad to CVRS is less extensive than to CDCS. CVRS only receives an answer on feasibility, while for CDCS, COPSLoad sends the complete load summary on what order needs to go where. Hence, the CVRS to COPSLoad interaction is easier to solve and there is more to gain.

As ORTEC has already widely implemented the CVRS component at customers, it is unfavourable to completely redesign the module. CVRS is a very comprehensive piece of code, especially compared to COPSLoad. We are not able to analyse it entirely within our time frame. Besides, CVRS on its own

performs properly, and therefore major changes are not recommended. Therefore, we focus on improving COPSLoad, and we keep CVRS out of scope. The programming environment C++ will be used to implement our solution, given that CVRS and COPSLoad are also programmed in this environment.

1.6 Research Questions

To come to an appropriate answer for the problem and reach the goal of this research, we formulate a number of research questions. First, we want to know what is currently known in the literature about our research problem.

1. What is currently known in the literature about the vehicle routing problem with load allocation constraints?
   (a) What type of constraints can be identified within the loading problem?
   (b) What type of problems are related to the loading problem?
   (c) Which algorithms can be used for solving the load allocation problem?
   (d) Which algorithms can be used for solving the combination of the VRP and load allocation problem?

Second, we want to have an up-to-date set of load allocation constraints. Distributors of retailers that currently use ORD are the starting point of identifying the set of characteristics. Once we have this set that characterize retailers, we translate the characteristics to constraints for the model.

2. Which load allocation constraints are relevant to our case study?
   (a) What are the characteristics of distributors of retailers?
   (b) Which requirements are foreseen in the future?
   (c) How are the characteristics translated to constraints?

Third, we want to come up with a suitable algorithm to quickly check if it is feasible to load a given set of orders inside the resource combination. This algorithm should meet the characteristics and constraints set beforehand. It is important that it fits the current ORD framework as well.

3. What load validation algorithm is suitable to quickly validate if the load allocation constraints are met?

Fourth, we want to know how the algorithm performs. We test our algorithm against the current one using existing datasets from the retail distributors. As said before, it is valuable to know what happens to the quality of the solution if certain constraints are disabled. So, we are also going to test the algorithm with different constraints disabled, to conclude if distributors need to adjust their policies.

4. How well does the developed algorithm perform?
   (a) How well does the proposed algorithm perform compared to the current methods?
   (b) How well does the proposed algorithm perform under different requirement sets?

The remainder of this thesis is structured as follows. In Chapter 2 we answer question 1, by giving an literature review on what is currently known. In Chapter 3 we discuss the characteristics of our customers and the related restrictions we have to deal with. We combine the literature review of Chapter 2 and the insights of the case study in Chapter 3, to develop the load validation algorithm that we present in Chapter 4. In Chapter 5 we present the experiments we ran to test the algorithm and analyse the results. In Chapter 6 we summarize our conclusions of this research and present our recommendations and opportunities for further research.

Chapter 2

Literature Review

This chapter describes the current state of literature related to our research. First, in Section 2.1 we describe the literature on load allocation as it is an independent problem. As mentioned in Section 1.2, our load allocation problem is a separate problem compared to the vehicle routing problem. We will only focus on the load validation part, because we need to remain in a feasible scope. However, as Pollaris et al. [8] suggest, the key to a good solution of a VRP with load constraints is to have interaction between both the VRP and the PP. This makes it is worth considering, because the solution methods for the combined problem of VRP and PP can have pieces that are interesting for our problem. We give an overview of possible solving methods for this combined problem. In Section 2.3 we conclude with the most promising approach for our research.

2.1 Load allocation

Load allocation can be treated separated from the VRP, because the VRP algorithm only asks for a feasibility check. No additional information on, for example, search space or guidance is shared. Therefore, every proposed schedule by the vehicle routing algorithm can be seen as an unique problem to the load algorithm. Note that this is the current practice of CVRS and COPSLoad. We start this section with a classification of the different problems that are relevant to our problem. Secondly, we give an overview of the types of constraints related to load allocation. Lastly, we give an overview on the solution methods on solving load allocation.

2.1.1 Classification

The Packing Problem as we describe in Chapter 1 is part of a much more general problem, namely the Cutting & Packing Problem (CPP). The CPP appears under different names in the literature, e.g., bin packing problems, knapsack problems, pallet loading problems, and container loading problems (CLP) [9]. Bischoff and Ratcliff [10] considered a large amount of papers regarding these problems and concluded that not a lot of practical situations have been incorporated so far. Even though a lot has been done since this research was published, the complex world creates a huge amount of possible restrictions. Therefore, there is a high possibility our problem will not has been considered so far.

Dyckhoff [9] developed a classification of the CPP's. He argues the CPP can be described by four characteristics, these are:

1. dimensionality
2. kind of assignment
3. assortment of large objects
4. assortment of small items

Dimensionality: the minimal number of geometric dimension that are required to describe the layout. This can be in either 1, 2, 3, or n>3 (e.g., time is a component) dimensions.
Kind of assignment: two objectives can be distinguished, either all items need to be included and the number of objects (vehicles/containers etc.) needs to be minimized; or the number of objects is fixed, and we try to assign as much items as possible. Note that the objective of our load problem is neither of these two, as we just want to know if it fits. Meaning both the number of objects and items are fixed.
Assortment of large objects: Is the diversity of the objects. The simplest assortment is one large object. Two other types are multiple objects but with a homogeneous configuration and multiple objects with a heterogeneous configuration.
Assortment of small items: the characteristics of the items. Distinction can be made between a few items, items of many different shapes, many items of relatively few different shapes, and congruent shapes.

Wäscher et al. [2] updated the typology of Dyckhoff [9]; in their opinion the typology got incomplete and inconsistent due to developments over the years. Their typology can be found in Figure 2.1. The classification differs from the classification of Dyckhoff [9] on some aspects. The characteristic "kind of assignment" with originally four classes, has been changed into only two classes: output maximization or input minimization. Output maximization is trying to assign as much orders as possible to a given amount of objects. Input minimization tries to use a minimal number of objects to fit all the orders. They also include variable (alterable) dimension(s) as a characteristic and combined "many items of many different shapes" with "few items of different shapes" into one case named strongly heterogeneous assortment. Lastly, they differentiate between weakly and strongly heterogeneous assortment of objects in the case of output maximisation. The remainder of the characteristics in general matches the original classification.

2.1.2 Constraints

Bortfeldt and Wäscher [3] made a categorisation of load constraints that they based on in literature. They categorized these load constraints into five different classes: container-related, item-related, cargo-related, positioning, and load-related. To our knowledge, this classification of load constraints is the first in its kind. Since the publication of the paper, no extension has been made or other classification scheme has been published.

The classification of Bortfeldt and Wäscher [3] is quite extensive, but some relevant aspects of certain retailer cases only get little attention. For instance, it is common for retailers to have two containers, in which again multiple compartments are created. The authors only consider a single container in their framework. They do consider contamination constraints, in which certain goods have to be separated. However, this is only considered on a truck level and not on a smaller compartment level. Furthermore,

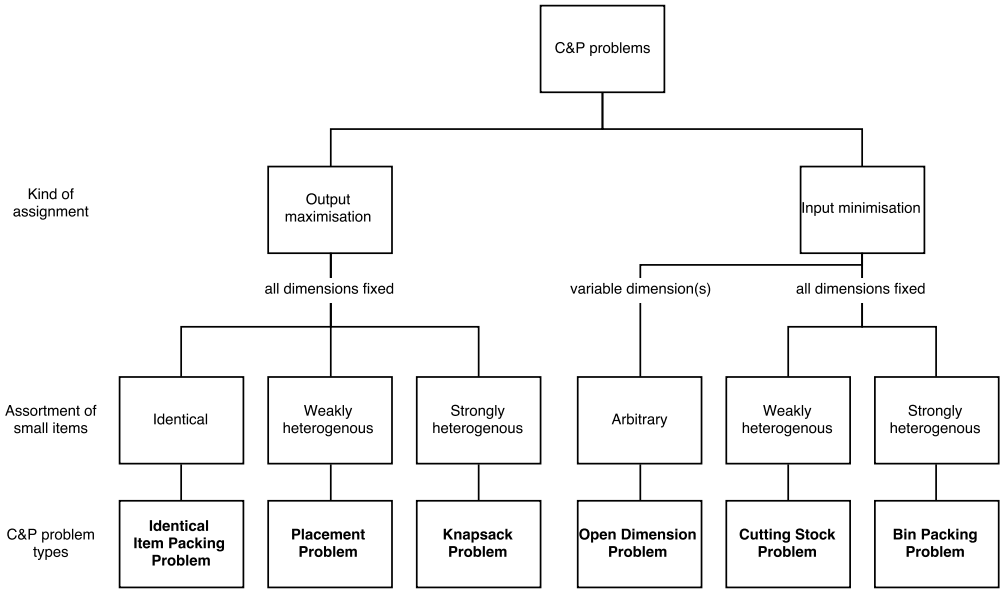


FIGURE 2.1: The basic problem types as Wäscher et al. [2] developed them.

flexible compartments are completely absent. Therefore, we create a framework for our understanding of the interrelationships between classes. We accomplish this by making one addition and some small modifications to the current classification of Bortfeldt and Wäscher [3]. The five classes together with our addition and modifications can be found in Figure 2.2.

We add a hierarchy among the five classes, in which the higher classes consist of constraints that focus on relatively larger aspects of load allocation. For instance, cargo-related constraints (highest) consider the division of orders over multiple trucks, while item-related constraints (lowest) consider constraints on the individual boxes themselves. Therefore, higher classes are of larger magnitude than smaller classes. We describe also the influence of the classes among each other. The individual characteristics have a strong impact on the decisions higher up in the classes. For instance, the constraints on how to orient the item (item-related) directly affect how to stack them on a pallet (load-related), and position them inside a compartment (positioning). This affects the weight constraints of the container (container-related), what again affects how to assign which orders to what trucks (cargo-related). In the following paragraphs we describe the classes of Bortfeldt and Wäscher [3], together with the small modification we made to these classes.

Cargo-related:
Cargo-related constraints focus on the allocation of the orders in a set of vehicles. If orders consist of multiple items, it is possible to ensure that the complete order is shipped (complete-shipment constraint). Partial loading is therefore not allowed. Note that this only occurs in output maximization instances [2], as in input minimization the goal is to load all orders with the least possible vehicles. A special type of this constraint is if all items of one order needs to be allocated close to each other (connectivity constraint). This is only applicable if the complete order stays together and therefore no split deliveries can be made. Besides that, it is also possible to have certain orders separated (separation constraints), based on their characteristics. For instance, food and toxic items.

Container-related:
Container-related constraints consist of constraints related to one transportable object, which can consist

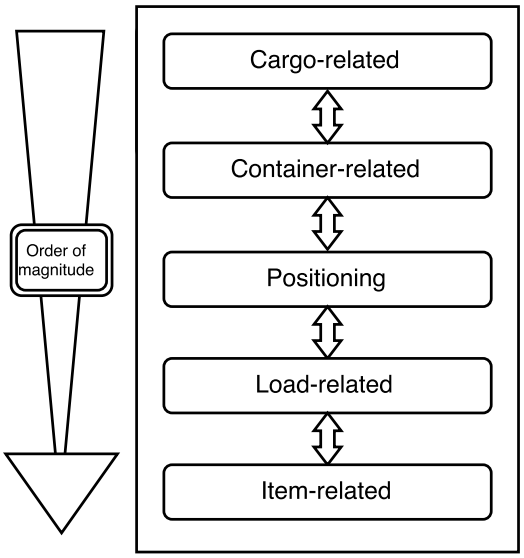


FIGURE 2.2: The updated load classification of [3] as we propose it.

of multiple containers (typically 1 or 2). Constraints in this class are typical weight related. Bortfeldt and Wäscher [3] describe weight limits and weight distribution constraints. Weight limits are limits on the total weight of the container or vehicle as a whole. Weight distribution is related to the centre of gravity of the vehicle. These constraints can apply along the three dimensional axis of the vehicle.

Positioning:
Within a container, it is possible to have multiple compartments. Orders are allowed in a container, if they are allowed inside at least one of the compartments of that container. Orders can be forced to be placed in certain compartments, or deliberately kept out of certain areas (absolute positioning constraints). Orders can also be banned from specific compartments, based on different orders that are placed inside that compartment (relative positioning constraints). A combination of these two constraints is also possible, especially if the load order is of importance (multi-drop constraints). Well known example of a multi-drop constraint is the Last-in-First-out (LIFO) policy.

Load-related:
Inside compartments, items are usually stacked to increase the utilization (note: if no compartments are present, we are talking about the whole container). Stacked items can be dangerous if they are unstable. There needs to be a minimum amount of support from the item below the stacked item, e.g., a minimal percentage of surface (vertical stability). However, high stacks can still be unstable and tip over during transportation. A solution is to ensure they are packed to other items or the container wall (horizontal stability). This means they are sandwiched and do not move horizontal.

Item-related:
On the lowest level we find constraints related to individual items. To certain items it is important how they are orientated relatively to the container (orientation constraints). Think of dishwashers that cannot be put on their side ("This way up!"), or livestock that should face the front of the vehicle. When items are allowed to be stacked, the load-bearing strength needs to be taken into consideration (stacking

constraints). This means an item can only bear a certain number of items, or a maximum force on top of it.

2.1.3 Solution methods

In general, we can distinguish two types of solution methods of the CPP. These two types are also used in many other Combinatorial Optimization problems (CO). The first type consist of exact algorithms, which are often used on smaller instance sets and result in an optimal solution. The disadvantage of this method is that the computational time can get extremely large in already medium large instance sets, making it not useful in practical situations [11]. Typically, heuristics are used to overcome this problem because they are able to find solutions in a relatively short amount of time. The solutions are typical of reasonable to high quality, but there exists a gap between the optimal solution and the found solution. Often used heuristics are construction and tree traversal algorithms. In more recent literature the use of meta heuristics have been proposed [6], like genetic algorithms, tabu search, and Greedy Randomized Adaptive Search Procedure (GRASP) [12].

MILP models

PP is known to be a NP-hard problem [13]. NP-hard problems are not guaranteed to be solvable in polynomial time, creating the risk to have a model that does not deliver an answer. This could be the reason why there are in the literature only a few contributions made on MILP models that solve PP [12].

Wu et al. [14] developed a mathematical model to solve a 3-dimensional BPP (3D-BPP) with flexible bin heights. They derived their model from a model that did not have the flexible heights. During experiments they noticed that the algorithm did not lead to any results for a majority of the test cases, even though the maximum running time was set on 2 hours (using proper equipment). Therefore they developed a genetic algorithm which obtained results in seconds, concluding that MILP is not ideal in complex situations.

Junqueira et al. [15] present a Mixed Integer Linear Programming (MILP) model for the load problem of boxes inside a container with the multi-drop constraint. Similar to our problem, the delivery route is already known in advance. This means the orders have to be loaded in reverse order, to make sure the right orders are available at the right time. The researchers add the constraint of the maximum reach of the driver, so he does not have to step on other orders who are still on the ground.

Construction algorithms

Over time, many wall building heuristics have been developed to solve the PP. Some of these wall building heuristic are purely used in the construction phase of the initial solution, but many of them made their way to also optimize the initial solutions. The following subsections present several wall building algorithms.

George and Robinson [16] were the first to attempt to pack a set of boxes inside a container. They used a layer or wall building algorithm to optimize the way the boxes are packed inside the container. The width of the layer is set by the width of the first box picked, the layer is filled continuously until it is

full. Only then a new layer will be formed. Due to the year of publishing, the algorithm was not only solvable by a computer, but it could also be used to solve the problem by hand.

Crainic et al. [17] propose a heuristic based on extreme points to place 3-dimensional boxes inside a container. Extreme points are the points in a current solution at which additional orders can be placed. One of the advantages is that several constraints can be included, like fixed placed orders.

Tree traversal algorithms

Pisinger [13] adapted a wall based heuristic to solve the 3-dimensional Bin Packing Problem, i.e., maximize the amount of boxes in a container. To overcome local optima of wall-building, they implemented a tree traversal heuristic. At every node, multiple layer depths are chosen according to a ranking rule, and are filled with boxes. They choose the layer filling which has the overall best use of the volume before proceeding to the next layer.

Christensen and Rousøe [12] developed a tree search heuristic to solve a 3-dimensional Bin Packing Problem with sequential loading. They combined the tree search heuristic with five greedy methods, plus a dynamic breadth strategy to make the framework more generic. The algorithm should function as a feasibility check on a suggested route, like in our case. At every insertion of an order in the tree, the remaining boxes are ranked on the volume usage. Only the top ranked boxes are considered for insertion, to limit the search space. The breadth of the search space is dynamic, and progresses as the search is going deeper into the tree.

Genetic algorithms

Gehring and Bortfeldt [18] developed a genetic algorithm to solve the 3-dimensional Bin Packing Problem. The algorithm is a derivative of the wall building idea; instead of walls, towers are constructed. Full support over the whole surface is needed for boxes to be stacked. The towers are build using a greedy algorithm that tends to minimize the spare spaces lying above the base boxes. These towers are the input for the genetic part of the algorithm.

Li et al. [19] developed a genetic algorithm to solve a 3-dimensional Bin Packing Problem with heterogeneous bins. The box packing sequence and container loading sequence is translated to a genetic string. This string is the input of the algorithm. Solutions are high in quality, while running time is around 10 seconds for small instances and around 100 seconds for larger instances. This is still more or less 10 times faster than using MILP.

Tabu search algorithms

To solve the 3-dimensional Bin Packing Problem, Lodi et al. [20] also developed an algorithm. They use a tabu search algorithm to generate near optimal solutions. Again layers are produced, this time with the height-first-area-second idea. A layer is constructed by selecting orders based on their heights, and are afterwards re-sorted based on the area. Tabu search is used to optimize this initial solution.

GRASP algorithms

Lim et al. [21] develop a heuristic for the single container loading problem with axle weight constraints (SCLPAW). They use an existing GRASP algorithm to builds walls out of the boxes, as an initial solution. Afterwards, a MILP model is used to rearrange the walls in such a way the axle constraints are valid. If no solution is found, boxes are removed from the container and a next attempt is made.

Hybrid algorithms

Ceschia and Schaerf [22] present an algorithm to solve a multi-drop multi-container loading problem. It builds initial solutions using random order (boxes) picking and assigning them to vertical layers. This initial solution is further optimized using local search. Afterwards Simulated Annealing and Tabu Search are used to optimize the solution.

Do Prado Marques and Arenales [23] propose a revised version of the Knapsack problem, to handle different classes and flexible compartments. The problem is modelled as an integer non-linear optimisation problem for which four variation methods were designed. They use a branch-and-bound algorithm to find the final optimal combination of compartments, as the number of items to be selected is not so large. Leão et al. [24] proposes two additional solutions methods to the exact same problem. The first is a combination of two of the four methods of Do Prado Marques and Arenales [23]. The second approach is a reformulation of the ILP model. First, all possible compartment configurations are generated. Within the ILP model all constraints are removed that ensure the use of feasible compartments only. This reformulation makes the ILP easier to solve.

2.2 Two-Phase approach

In Section 2.1 we focussed solely on load allocation. In this section we focus on the combined problem of the VRP with load allocation. In Section 2.2.1 we discuss the different classes of the VRP with load allocation based on literature. In Section 2.2.2 we give an overview of possible solution methods for this combined problem. Note that we are not interested in solving the combined problem, but only in the load allocation itself. In this section, we only use the most relevant elements of the methods found .

2.2.1 Classification

Our problem can be divided into a vehicle routing part and a load allocation part. This division is described in literature [11]. Two papers use a similar taxonomy to classify different type of routing problems with load characteristics. Iori and Martello [25] and Pollaris et al. [8] defined nine classes, in their search for articles they include in their taxonomy. These classes are: Three-Dimensional Loading CVRP (3L-CVRP), multi-pile VRP (MPVRP), Traveling Salesman Problem with Pickups and Deliveries (TSPPD) with loading constraints, Pallet Packing VRP (PPVRP), multi-compartments VRP (MCVRP), Two-Dimensional Loading CVRP (2L-CVRP), Minimum Multiple Trip VRP (MMTVRP) with incompatible commodities, Double TSP with Pickups and Deliveries with Multiple Stacks (DTSPMS),

and Vehicle Routing Problem with Pickups and Deliveries with multiple vehicles (VRPPD) with loading constraints.

In Chapter 3 we discuss the details of the two case studies we identified. It becomes clear that the three dimensional aspect is not relevant for our load allocation. Therefore, only the latter five of these nine classes defined by Iori and Martello [25] and Pollaris et al. [8] are worth considering in our research. The literature review of Pollaris et al. [8] gives a good overview of relevant articles related to these five classes. This literature review forms the basis of Section 2.2.2 of the articles we include in our literature overview. To expand our view, we also searched for other sources.

2.2.2 Solution methods

This section gives an overview of different solution methods of the combined problem of the VRP with load constraints. The articles are not categorized based on the evaluated problems, but based on the proposed type of solution method.

MILP

Pollaris et al. [26] developed a MILP model to solve the CVRP with axle constraints and sequential pallet-loading. The model was solved using CPLEX and tested on 128 different instances. Smaller instances with 10 customers are solvable in reasonable time. The instances with 15 customers or more have solution times of far over 60 seconds.

Branch-and-Cut

Lahyani et al. [27] developed a fullly integrated model to cope both with routing as well as loading. The model can solve a multi-product and multi-compartment vehicle routing problem. It is assumed that the fleet of vehicles is fully homogeneous, as are the compartments inside the vehicles. A branch-and-cut algorithm is used to propose optimal solutions.

Iori et al. [28] developed an exact approach to solve the CVRP with 2D bin packing. Among other restrictions, sequential loading and item clustering are included. The CVRP is solved using a branch-and-cut algorithm, while the feasibility of the routes regarding loading is checked using a branch-and-bound algorithm. Infeasible routes are added to an infeasibility pool that works both as a pre-check and a constraint in the ILP model. The latter can been seen as a feedback mechanism.

Local search

Zachariadis et al. [29] propose a Static Move Descriptor (SMD) algorithm together with a separate load heuristic to solve the 2L-CVRP. The SMD algorithm acts as input for the load heuristic, checking only promising (partial) routes. All combinations of the constraints LIFO and rotations of orders are considered. There is no additional information shared between the load and the VRP part, besides true/false. The load heuristic uses a utility function based on the touching perimeter heuristic to choose the most promising orders to be loaded first. A hash table memory structure is added to the utilize function to diverse from more promising, but finally infeasible solutions.

Variable Neighbourhoods Search

Henke et al. [30] developed a complete integrated approach to search for a solution in a multi-compartment VRP with flexible Compartment Sizes (MCVRP-FCS). The algorithm assigns orders deterministically to the vehicles, by filling a vehicle until it is full and then picking the next one. Afterwards the Variable Neighbourhood Search is used to find better solutions. When the solution uses more trucks than available, this is translated as a heavy penalty in the objective function. To overcome getting stuck in a local optimum, a multi-start approach is introduced.

Wei et al. [31] came up with a variable neighbourhood search for CVRP with 2-D load constraints. The approach handles two different versions of load constraints, one unrestricted and one with sequenced loading. The load constraints are validated using a skyline heuristic. The perimeter of the top view of the container is considered as the skyline. This heuristic tries to minimize waste that results in gaps. To speed up the process, a special data structure called Trie is used, which keeps track of previously examined sequences. If the VRP wants to validate a (partial) sequence, Trie responds if a partial sequence was checked before and whether or not the result was valid.

Tabu Search

Another 2L-CVRP approach is presented by Leung et al. [32], in which an Extended Guided Tabu Search (EGTS) is proposed together with six possible heuristics to validate the load constraints. Also sequenced and no-sequenced loading are considered.

Gendreau et al. [33] propose a tabu search algorithm to find the best CVRP. The load constraints are checked using one of two procedures, with or without the sequenced constraint. Boxes are placed at the location where it has the highest touching perimeter. If the solution is infeasible, the sequence of two orders is swapped and it is tried again. This continues for a predefined number of times. If the solution remains infeasible, the best infeasible solution is used while inducing a penalty in the tabu search.

Tao and Wang [11] also developed a way to solve the CVRP-3D, using a tabu search for the VRP part and more complex heuristics for the load constraints. Two different heuristics are proposed to solve the loading part, making use of the strengths of both. The first load heuristic called “the least waste algorithm” uses the same placing strategies as described by Tarantilis et al. [34], and places orders at corner points of a so far build solution. To overcome the problem of waste gaps, which arise when corner points are neglected when boxes overlap, the touching perimeter heuristic is used. This combination of heuristics aims to deepen the exploration of the packing space. The approach is effective for fragility, LIFO, and support constraints.

Wang et al. [35] describe a reactive guided tabu search (RGTS) to solve the heterogeneous fixed fleet multi-compartment vehicle routing problem (HFFMCVRP). The RGTS automatically updates the penalties on the arcs of the network (guided part) and adjusts the size of the Tabu List (reactive part), to come to a near optimal solution. The loading part is submissive compared to the routing part of the problem.

Genetic algorithms

El Fallahi et al. [36] was the first to consider multi-compartment VRP with the practical application of groceries. Until then, papers concerned only liquid (petrol) distribution. The paper introduced a completed integrated way of solving both the VRP and the Multi Compartment constraints. A memetic algorithm, a kind of genetic algorithm, as well as a tabu search is proposed to solve the problem. First, a set of VRP's are built using the Clark & Wrights method, in which one VRP only consist out of one product. From the set of VRP's, two sets are drawn together that will produce one child, (which then will have 2 products). This child will be evaluated and local search searches for an even better solution. The child is split at first, as it is still one grand tour, and put back into the set of parents when it has a better performance. The algorithm stops after a fixed amount of iterations, or after a number of iterations without any improvement.

Ruan et al. [37] developed a hybrid approach to combine the CVRP with 3-D load constraints. A Honey Bee Mating Optimization (HBMO) is used to produce feasible solutions for the CVRP part of the problem. Afterwards, the top solutions are fed to a load heuristic, which validates if the items fit into the container. If it does not, it will take the second optimal solution and so on, until a feasible solution is found. The load heuristic consists of three load sequences and six packing heuristics. The three load sequences place the items in descending order of volume (wlh), bottom area (wl), and height (h), respectively. Per sequence, the following six heuristics are used to place the orders inside the container; Back-Left-Low, Left-Back-Low, Max-Touching-Area-W, Max-Touching-Area-No-Walls-W, Max-Touching-Area-L, and Max-Touching- No-Walls-L. The heuristics are sorted in increasing order of complexity, further explanation is given in [34]. So in total 3 times 6 strategies are used to validate one solution on the loading constraints.

2.3 Conclusion

Initially we chose not to alter the Vehicle Routing part of ORD, and only focus on the load allocation part. This is a choice we still support, even though the caching function of Wei et al. [31] seems a promising method to keep track of all solutions so far. This way, we prevent the load allocation algorithm of searching for solutions of sequences in which we can predetermine the solution. The major advantage of this, is that we can implement this without the need to interfere with the VRP algorithm.

The use of exact algorithms does not seem to be the right choice for our load allocation problem. Many of the approaches that use exact solution methods have difficulty to solve larger instances. Therefore, it is evident that many practical situations are not suitable for an exact approach. Logistic companies only benefit if the software they use allow for larger instances.

Meta heuristics for the PP based on GRASP, tabu search, and genetic algorithms are difficult to implement when sequential constraints are included. The heuristics are based on swapping boxes within a trip, or between two trips. However, a movement of box to an other position does not guarantee the sequential constraints still hold [12]. In Chapter 3 we show that the sequential constraints as Christensen and Rousøe [12] describe them are also relevant to our research.

Summarizing, we focus on implementing the caching function to keep track of previously examined sequences of orders. This prevents unnecessary calls to the load allocation algorithm. Beside caching, we

develop a variant of the tree traversal algorithm of Christensen and Rousøe [12] to solve load allocation. We adjust it so it matches all of our load constraints as we describe them in Chapter 3.

Chapter 3

Case Study

The previous chapter gave an overview of the literature related to our problem. We reduced the scope to only retail distributors, to limit ourselves to the relevant constraints for these type of companies. This chapter provides an analysis of our case study, which we will refer to as the retail case. In Section 3.1 we describe the companies of our retail case. In Section 3.2 we look at any future developments in the retail, relevant for the retail case. In Section 3.3 we limit the retail case in such a way that it contains all present cases and future developments.

3.1 Company profiles

This section describes two retail companies with a specialized department responsible for the distributions to the stores of the company. The two distributors are located in Australia and Russia. Both are current customers of ORTEC, using software for their distribution processes.

Retailer 1
Retailer 1 is an Australian supermarket chain and operates many stores throughout Australia. Retailer 1 is already using software of ORTEC for their liquor and ambient temperature products planning and would like to extend their planning with temperature controlled vehicles. There are four types of vehicles:

FIGURE 3.1: Operating areas of the two distributors (shown in dark). Retailer 1 in Australia and Retailer 2 in Russia.

standard trailers with Single, Dual and Triple compartments and truck-trailer combinations. A truck-trailer combination consist of two resources, both in which orders can be transported. There are three temperature categories: Ambient (18 °C), Chilled (4 °C) and Frozen (-18 °C). In each compartment only one temperature can be programmed and only products of that temperature type are allowed in that compartment[1]. Baffles separate the compartments. The baffles are flexible, meaning that they can take any size as long as the total volume of all compartments is equal to size of the container. An additional request is to add a contamination restriction on address level. This means that certain products are not allowed to be delivered together at the same location, even if they are allowed in the same compartment. The restrictions can differ on order location level, meaning that a certain contamination constraint can exist at on one location, but not at another.

Retailer 2

Retailer 2 is a Russian retailer with hundreds of stores and vehicles. Due to the distances in the area where they are located, deliveries go up to 2000 kilometres[2]. The retailer identified around 10 different product types, each with different requirements. Some of these product types should not be combined within the same resource (truck or trailer), e.g., fish and vegetables. The long distances together with these compartment restrictions, makes the right use of resources important. In addition, it is forbidden by law to drive with a combination where truck weight is lower than trailer weight.

3.2 Future of Retail

We should not only look at the current state of the distribution of retailers, but also at the future. Future changes in the supply chain could affect the completeness and effectiveness of our designed algorithm. Currently, the whole retail sector is undergoing a transformational change, triggered by digitalization. This creates a permanent impact on today's businesses [38].

There is a clear trend visible in which more and more products are both researched and purchased on-line. Chaturvedi et al. [4] looked at different types of retailers and studied the percentage of customers researching and purchasing products on-line. Among the different types of businesses, three major areas can be identified where digitalization is happening. These three areas are "Still in store", "Digital battleground", and "Gone to digital" and can be found in Figure 3.2. Purchases made on-line are directly sent to the customers, and skip delivery to a store. This affects the amount and frequency of store deliveries. However, our case study, retailers of supermarkets (grocery), can be found in the area of "Still in store". This means the amount of on-line purchases are low, so no major changes are happening at the moment. Besides, less demand for stores does not reduce the complexity of the problem, only the scale on which it is implemented.

The transformational change mentioned before is not only related to on-line purchasing [38], also in-store experience is changing rapidly. Changes currently introduced in retail stores are e.g., self-scanning cashiers[39], RFID chips in products [40, 41], and mobile-applications [38]. However, all of these changes do not affect the working of our load algorithm. Another challenge nowadays and in the near future, is the rising customer demand of high product availability. Customers are not willing to wait for products to be replenished, and take it for granted that they can buy what they want, whenever they want, wherever

[1] Retailer 1 Change Request
[2] Retailer 2 Change Request

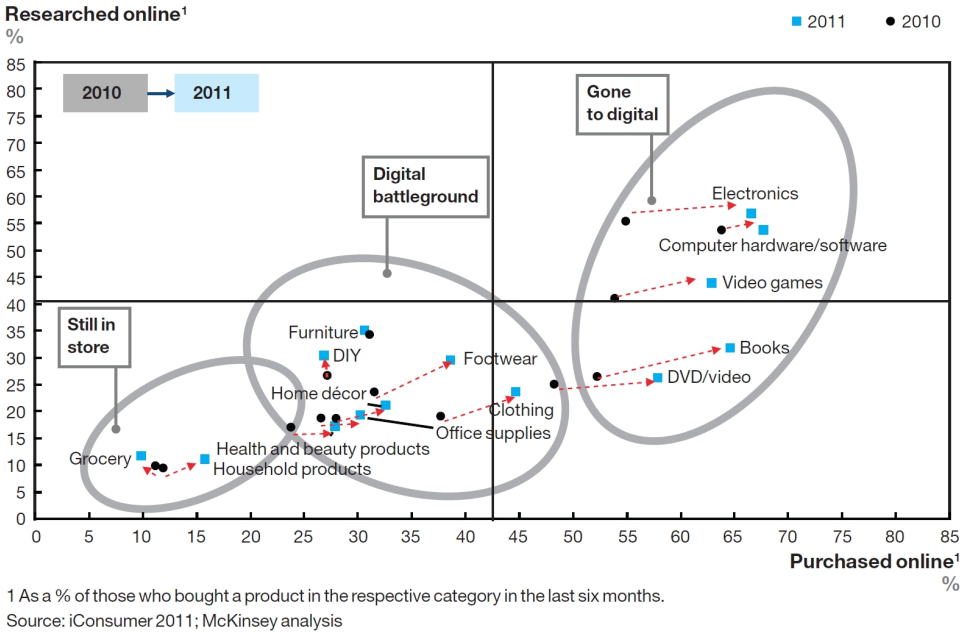


FIGURE 3.2: Trends of different categories of products that are researched and purchased on-line. Figure as presented in [4].

they want [42]. Besides the increasing need for a reliable and fast solution, this does not affect our case study and can be ignored.

Looking at the development of trucks, no major changes are expected in new freight trucks that will affect our model. Developments that are currently finding their way to the real world are higher capacity trucks, electronic detection of non-compliance, and fully automated trucks [43]. These developments do not directly affect the model. Therefore, it is highly unlikely that within reasonable time unforeseen truck development arise that will undermine the validity of the model. To conclude, there are no real threats or challenges that arise in the near future for the retail case.

3.3 The Retail Case

This section defines the requirements of the retail case. The retail case is a combination of the two customer requests of Retailer 1 and Retailer 2. This set of requirements is complete enough to be applicable for a typical retailer. The Sections 3.3.1 to 3.3.8 will individually explain a specific set of requirements of the retail case.

3.3.1 General

Retail is defined as all companies that sell products. However, it is common to restrict the definition to the subset of grocery stores, supermarkets, and hypermarkets. These stores offer a wide selection of products, often exceeding the classic daily purchases. Nevertheless, food and consumer products are still the major part of the sales, which results in a high inventory turnover. Therefore, stores are replenished a few times a week, to keep a certain level of stock and guarantee the quality of perishable goods. The maximum number of orders carried by a resource combinations is around 25.

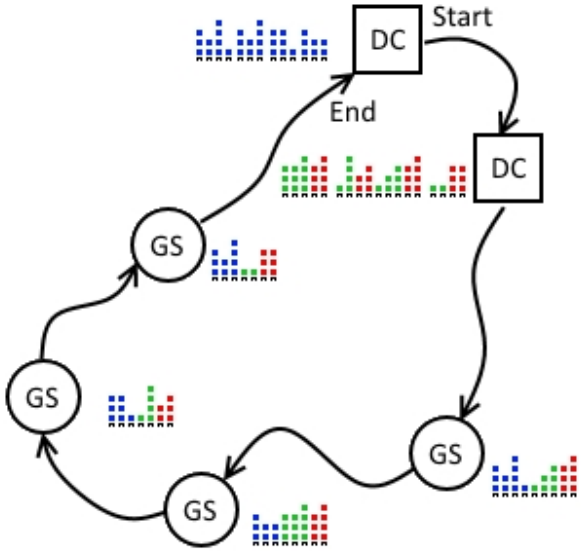


FIGURE 3.3: A possible pick-up & delivery sequence. First the frozen pallets are picked-up at the first depot, the chilled and ambient at the second depot. Afterwards the orders are delivered at the Grocery Stores (GS). Every stack represents one order, every box a transport entity .

As described earlier, supermarkets are replenished by a privately owned distribution branch. Trucks are used to fulfil the replenishments of these stores, in which one truck visits multiple stores in a row. It is possible to have all types of products delivered at once, this means frozen products (e.g., ice cream) are delivered together with chilled products (e.g., milk) and ambient products (e.g., rice). This is possible because the trucks are equipped with compartments with different temperature settings. A trip is a series of pick-ups and deliveries a truck has to make, starting and ending at the depot. In between, the truck visits the locations where the deliveries have to be made. Figure 3.3 displays this representation.

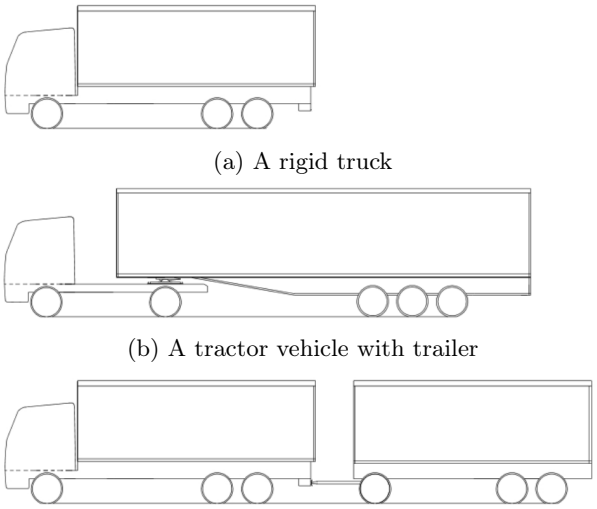
3.3.2 Resources

The distributor has a set of trucks to its disposal, which he can use to replenish the stores. The truck types are weakly heterogeneous, meaning that there are just a few different types of trucks compared to the total amount of trucks available. Common types or configurations are rigid trucks, trailers with tractor vehicle, and truck-trailer combinations. Figure 3.4 shows all three configurations. Configurations with more than 2 resources are considered as road trains, which are only used in remote areas to transport bulk goods. Therefore, we assume that a truck consists of at most 2 separate fixed-sized resources.

(1.1) At most one trailer can be attached to a truck (2 resources).

3.3.3 Orders

An order is the quantity of one product that needs to be delivered at a location. Typically, these orders are placed upon pallets, roll cages, or an equivalent transport entity which can be easily placed and picked from a truck. The height of this entity is irrelevant, as long as it is not taller than the height of the resource. Therefore we neglect the height of the transport entities. All transport entities are homogeneous, meaning they have the same size for all orders. All orders have temperature restrictions,



(a) A rigid truck

(b) A tractor vehicle with trailer

(c) A Truck with a trailer

FIGURE 3.4: Three different resource types.

meaning they can only be moved in compartments which match these temperature restrictions. In general, three temperature categories, or more generally called product groups, can be distinguished: ambient (18 °C), chilled (4 °C), and frozen (-18 °C) (similar to Retailer 1).

(1.2) 1, 2, or 3 product groups.

(1.3) Orders have to be placed inside compartments with equal compartment types.

(1.4) Orders belong to exactly one product group.

Besides temperature restrictions, orders have also restrictions regarding which orders are allowed together in the same compartment. Therefore, orders are categorized into product kinds (e.g., bakery, fish, or liquor), each having other characteristics. Section 3.3.7 gives a full explanation of the use of product kinds. Because the orders are differentiated over product kinds, a delivery to a store consists of different orders with the same address. When for instance milk, bread, and beer are delivered, it is regarded as three orders with the same address. It is not allowed to split orders between compartments or resources. This means that if an order consists of three roll-cages, all of them should be placed into the same compartment.

(1.5) Splitting of orders (over compartments or resources) is not allowed.

Additionally, weight can be added to the order as an additional characteristic. For example, an order has a unit size of three, meaning it consists of three transport entities. The weight of one single order is the total weight of all transport entities. For instance, when an order consists of three pallets weighing 90, 75, and 105 kilogram, the order has a total weight of 270 kilogram. The product kind is "frozen bread", meaning this has to be placed inside a frozen compartment.

3.3.4 Compartments

A resource (truck or trailer) can be separated into compartments using baffles. These baffles are flexible, making the compartments variable in size. Compartments are used to create different temperature zones, but also to separate orders which are not allowed together. The compartment types are related to the

temperature restrictions of the orders. That is why we have three compartment types; frozen, chilled, and ambient. We assume that is not possible to have more than three different compartment types.

(1.6) *Baffles separate the compartments.*

(1.7) *Only orders with matching product groups are allowed inside a compartment with matching compartment type.*

(1.8) *1, 2, or 3 compartment types.*

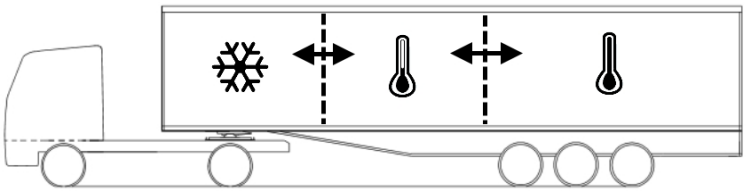


FIGURE 3.5: Compartments with different characteristics in a truck. Compartments are separated by flexible baffles.

The number of compartments per resource in the retail case will typically not exceed the amount of three. Therefore we limit ourselves to a maximum of three compartments. Distributors have the ability to assign the different compartment types to the compartments of a truck. They are not limited to the amount of compartment types per resource or in total. For example, this allows to have two times three frozen compartments in a truck trailer combination.

(1.9) *1, 2, or 3 compartments per resource are allowed.*

(1.10) *Compartments with equal compartment types are allowed within one resource.*

For example, the distributor has a rigid truck with three compartments. The upfront compartment has to be frozen, because the cooling unit is located on the front side of the resource. The compartment in the middle can be another frozen compartment, or be used for chilled products. The last compartment is used for either ambient products or chilled products. In our example, the layout of compartment types is fixed when all product kinds have to be loaded. However, when two resources are used, the quantity of possible layouts and therefore the solution space becomes large.

3.3.5 Co-loading

Large supermarket chains operate multiple depots from which they make their deliveries. A trip starts at one depot where the first orders are loaded into the truck. There is a possibility the depot cannot supply all orders, and the truck has to make an intermediate stop at a different location to receive the remaining orders. We allow for this to happen, but limit ourselves to the situation that deliveries occur after all pick-ups are made. This means that after the first delivery, no additional pick-ups are allowed before the truck is completely empty again. Allowing both pick-ups and deliveries interchangeably, complicates the retail case tremendously.

(1.11) *Pick-ups at more than one distribution centre is allowed.*

(1.12) *Only after all pick-ups are collected, deliveries are allowed.*

Commonly, stores will return items such as waste, empty roll cages etc. back to depot. These items are returned by the same truck that makes the delivery at the store, so no additional visit has to be

made. The volume of items returned is typically a small portion of compared to the volume delivered. It is therefore not necessary to check if these return items will fit in the truck/trailer, seeing it will only increase the number of calculations that have to be done. More complicated return orders such as returning previously delivered goods is outside the scope of this research.

(1.13) Return orders are allowed, but feasibility will not be checked.

3.3.6 Flexibility restrictions

The baffles are completely flexible, meaning it could be set at any location within the resource. However, we assume a baffle is only placed at one of a few fixed places, based on the number of roll cages or pallets loaded. Also, a resource may include side doors, which can be opened for (un)loading the orders. A baffle itself takes up space, due to the width of the object. This means that the capacity for the resource reduces. Because the length of a resource is not exactly divisible by the dimension of a transport entity, it is possible that one baffle does not reduce the capacity of the resource, while two baffles reduce the capacity by one row of transport entities. The capacity is discrete, because it is always a discrete amount of transport entities. This means the baffles can be placed after every row of pallets.

(1.14) A baffle reduces capacity of the resources.
(1.15) The stepsize of the baffles is discrete.

Compartments are not completely flexible in size. For instance, a compartment with type frozen has typically a cool unit inside the compartment with a maximum cooling capacity. This results in a minimum size to accommodate the unit, and a maximum size to not exceed the cooling capacity of the unit. There are also other reasons to have the ability to set the minimum and the maximum size, for instance to force orders into specific compartments.

(1.16) A minimum compartment size can be set.
(1.17) A maximum compartment size can be set.

3.3.7 Contamination

In Section 3.3.3, product kinds are introduced as a further differentiation among the product groups. Retailers are facing many different rules regarding food safety and perishability of products, especially on long distances cross-contamination may occur. Therefore, certain product kinds should be transported in separate compartments to reduce this risk. Orders are assigned to a product kind, which inherits the characteristics of the related product group. So within the product group chilled, we have for instance bread, fish, and meat. This results into the product kinds chilled bread, chilled fish, and chilled meat. Note: frozen meat and chilled meat are considered as two completely different kinds, even though they share the name “meat”. This holds for all product kinds.

(1.18) Every product kind is part of a product group.
(1.19) Every order is assigned to one and only one product kind.
(1.20) There can be at most as many product kinds as orders to be assigned in a truck.

(1.21) *Product kinds of different product groups never can be in the same compartment.*

(1.22) *Product kinds of equal product groups can be excluded to be in the same compartment.*

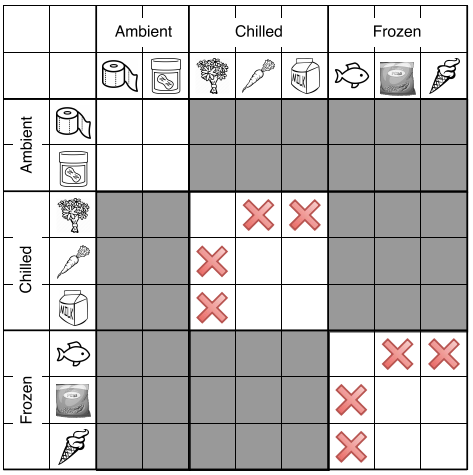


FIGURE 3.6: All product kinds of a product group are allowed in the same compartment. However, this contamination matrix does not allow flowers to be together with the other chilled products, and fish to be together with the other frozen products. Flowers and fish should be put into separate compartments.

3.3.8 Weight distribution

Due to safety regulations, it is obligatory to have a larger mass within the truck than within the trailer at all times. Checking the initial weight distribution upon departure from the depot is not sufficient, because every loading and unloading results in a different weight distribution between truck and trailer. This means that at every loading and unloading activity, the weight distribution has to be validated by the load validation algorithm.

(1.23) *The weight of the trailer should never exceed the weight of the truck.*

The main difference between the weight distribution and all previously mentioned requirements, is that the weight distribution restriction causes the sequence of orders to be relevant. So not only the characteristics of the orders are relevant, but also the way in which they are loaded and unloaded.

3.4 Conclusion

In the previous sections, we clustered the retail case into a list of 23 requirements. We use the framework we developed in Chapter 2, to classify our requirements. All of our requirements classify as "positioning", except *1.1* and *1.23* that classify as "container-related". Therefore we conclude that the problem is mainly concerned with the positioning of orders inside resources or compartments, and a bit on container-related issues. The other three classes as described in Section 2.1.2 can be neglected. The container-related requirements affect the load problem in a different way than the other load requirements. The two container-related requirements focus on the weight distribution of the resource combination. While the other requirements only focus on the combination of orders in a route, the weight distribution causes the sequence of orders to be relevant as well. It were especially the item-related and load-related constraints

on which a lot of literature was available. Note that the load-related constraints as described in Section 2.1.2 are not the same as when we refer to the load constraints in general.

According to Dyckhoff [9], the retail case can be defined by the following characteristics: "one-dimensional", "multiple objects with a heterogeneous configuration", and "many items of relatively few different shapes". Our load problem does not comply with one of the two assignments described Dyckhoff [9]. Either we minimize the number of objects and check if all items can be assigned, or vice versa. Our load problem can be reduced to one dimension, as we use number of pallets, volume, or weight. Note that we only consider one truck per schedule request, so the assortment of "One large object" would be more logical. However, the trucks consists of flexible compartments which can be translated to the assortment of "multiple objects with a heterogeneous configuration".

Using the typology of Bortfeldt and Wäscher [3], our load problem can be classified as a packing problem with items that are weakly heterogeneously assorted, which will be assigned to (weakly) heterogeneous objects. As explained before, the kind of assignment does not matter, as it can be translated both to output maximisation and input minimization. Therefore our load problem can be classified as either a Multiple Stock Size Cutting Stock Problem (MSSCSP) or a Multiple Heterogeneous Large Object Placement Problem (MHLOPP). Our load problem possibly includes flexible compartments, so the Open Dimension Problem (ODP) classification also matches. This kind of problem is the only within cutting & packing problem which includes flexible compartments. Wäscher et al. [2] pointed out that multi-compartment with flexible walls are not studied in the field of cutting & packing.

Chapter 4

Load validation algorithm

In this chapter, we present our solution method. The literature review from Chapter 2 and the findings from Chapter 3 form the basis of this method. In Section 4.1 we look at the solution space of the problem. In Section 4.2 we give an overview of our method and explain the decisions we make. Our algorithm is a combination of three consecutive processes; we describe these processes in Sections 4.3 to 4.5 individually. Note that Section 4.5 explains the caching function and is both the first and the last step of the algorithm. Because it makes more sense to first describe how the sequences are stored, before describing how they are retrieved, we moved it to end of this chapter.

4.1 Solution Space

In this section we determine the size of our solution space, as this may affect the design of a suitable method. During the load assignment of orders, we need to make several choices on which orders go when, in which compartments. In Section 3.3 we defined the requirements of the model. From these requirements, we identify five parameters that affect the size of the solution space. Table 4.1 depicts these parameters together with the number of possible settings.

Parameters	**Number of settings**
Resources	1, 2
Compartments	1...6 *(max. 3 per resource)*
Orders	1...n
Baffle settings	1...m
Compartment types	1, 2, 3

TABLE 4.1: The five parameters of the solution space. Three of the five parameters have a pre-defined number of settings. Two of them have no maximum, but typical it does not exceed n = 25 and m = 10.

When a sequence of orders gets validated, we need to decide in which resource to place the order, followed by the decision in which compartment to place it. The number of possibilities equals c^n, with c as the total number of compartments and n as the number of orders. Note that we take the sum of the compartments over all resources, so the number of resources are not part of the equation. However, as we have a maximum of three compartments per resource, the number of resources affects the number of compartments and therefore the solution space. For example, a sequence of 25 orders is placed in a

resource combination with one resource of three compartments and one resource of two compartments. The total amount of arrangements is therefore $(3+2)^{25}$, which equals to $298 * 10^{15}$.

However, this is only the arrangement of orders over a given configuration of compartments. We have two more things that influence the solution space. The compartment types of the compartments together with the size of the compartments (baffle settings). First, we have the ability to predefine the compartment type of a compartment. We can also leave the compartment type blank, meaning it can be either one of the three. It is also possible to exclude one compartment type, so a compartment can be either one out of two types. In the hardest case, we do not predefine any compartment type to a compartment, so all of the compartments are blank. This means that there are t^c number of possibilities, with t as the number of compartment types and c as the number of compartments. In our example, this would be three compartment types (frozen, chilled, and ambient) to the power of five compartments, which equals 243. Second, we need to multiply this number with the number of baffle settings. We approximate the number of settings, by taking the number of baffle places to the power of the number of baffles. This number also contains duplicate configurations, so we divide it with the factorial of the number of baffles. A resource would have around eight to ten different locations to secure the baffle, so in our example we assume we would have nine baffle places. Our example has one resource with three compartments and one resource with two compartments, resulting in one resource with two baffles and one resource with one baffle respectively. So we would have $(9^2/2!) * (9^1/1!) \approx 365$ configurations. Multiplying the total amount of arrangements with the number of compartment configurations and the baffle settings, we have a solution space of $264 * 10^{20}$. If we take the same example, but take a sequence of only ten orders, the solution space will still be $866 * 10^9$.

Given this example, we can state that even for smaller instances, the solution space is tremendous. However, this solution space can be reduced with some simplifications. First, we can remove the number of baffle settings from the equation, because we can deduct the baffles settings directly from the required volume of the compartments. A compartment only needs to be large enough for the orders assigned to that compartment, taking into account the number of rows required by the orders. For example, if we have one resource with 2 compartments, we do not have to validate all baffle settings of these 2 compartments. We only validate if compartment 1 and 2 have enough room to accommodate the assigned orders, and if both compartment still fit inside the resource. If the resource has a width of three roll cages, and we place ten roll cages in the first compartment, we know that we need to place the first baffle after four rows of roll cages.

This way, we can also deduct the compartment type based on the assigned orders. When a batch of orders are assigned to a compartment, we do not have to check for every compartment type if the batch is valid. We know that orders with different product groups are not allowed together, and that orders are not allowed inside a compartment with an incompatible compartment type. So instead of validating for a given compartment type if all orders are compatible, we reverse it. Given a batch of orders, we search for a compatible compartment type. This way, we only validate all possible order arrangements over the compartments.

To conclude, even with the simplifications of removing the baffle settings and compartment types from the equations, we keep a solution space of c^n. The following sections describe our developed algorithm, and show that even for this large solution space, we can determine feasibility relatively fast. It becomes clear that it is unnecessary to validate every possible arrangement or orders, because we can exclude certain subsets of a sequence of orders successfully.

4.2 Overview of the load validation algorithm

This section gives an overview of our solution method. In general, we divide the solution method of determining the feasibility of the loading constraints into three sub-processes. These sub-processes operate in sequence, one after the another. The algorithm only continues to the next process if the current process cannot determine an answer. The idea is to identify feasible or infeasible routes as fast as possible, in order to avoid unnecessary calculations. The third and final process always gives the correct answer. Note that the load validation algorithm cannot control its input, the vehicle routing algorithm provides the routes when it calls for a feasibility check.

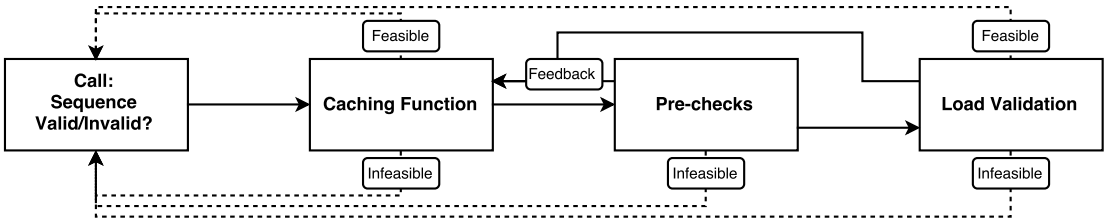


FIGURE 4.1: An overview of the three sub-processes of the algorithm. It starts with a call from the vehicle route algorithm (far left box). First, the caching function checks if this sequence has been verified before. If this is not the case, it moves to the pre-checks. If the pre-checks do not fail, it moves to the final process. Afterwards the caching function stores the response of either the pre-checks or load validation for future retrieval.

The vehicle routing algorithm calls the load validation algorithm when it wants to know if the load restrictions are valid for a composed route. During the optimization iterations of the vehicle routing algorithm, the algorithm just wants to check whether or not a solution is feasible, and it is not interested in the assignment of loads to resources itself. It sends the sequence of orders together with the corresponding resource combination that is used to execute the trip. The route consist of a sequence of pick-ups, deliveries, and travels that represent the route. Figure 4.2 gives a schematic overview of four consecutive calls to the load validation algorithm. To avoid confusion, from now on, we refer to a route as the four shown in Figure 4.2 as a sequence of orders, or a sequence for short.

A pick-up and a delivery are two independent tasks that refer to the same order. Tasks can be clustered together on an address, but it also possible to have only one task at a location. Note that pick-ups and deliveries are never at the same location, and that all deliveries are succeeding the pick-ups. The last three sequences in Figure 4.2 are similar, but their internal sequence differ. All three of them have five orders that have to be picked-up at two locations, and delivered at three different addresses. The load validation algorithm does not have any information on the depots or delivery addresses, but it derives this by looking at which tasks are separated by a travel. The nodes with a T_j in Figure 4.2 represent these travels, with j representing the location of the travel within the route. Besides the sequence, the vehicle route algorithm sends the information of the resource combination to the load validation algorithm, as two different resource combinations on the same sequence can lead to a different response.

The first process is a smart caching function that saves prior sequences and corresponding answers that can be retrieved to prevent recalculating what is already known. The caching function is also known in literature as trie or retrieval structure (Wei et al. [31]). Data structures that have been used are pool structures (Fuellerer et al. [44]), hash tables (Zachariadis et al. [29]), and tree structures (Zachariadis et al. [45]). A tree structure has as advantage to efficiently store similar sequences of elements, typically

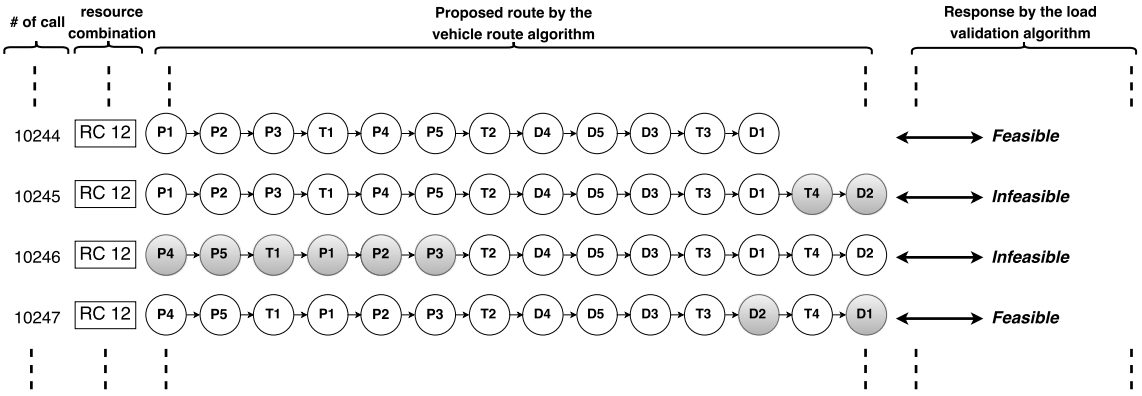


FIGURE 4.2: Schematic representation of four sequences of orders proposed by the vehicle routing algorithm. The grey nodes are the modifications the vehicle routing algorithm made after the reply of the load validation algorithm. An order (o_i) consist of a pick-up (p_i) and a deliver d_i. Travel t_j separates the different location, so we know which orders are loaded and unloaded together.

sequences of characters to form a string. This characteristic is relevant to our research, because the weight distribution restriction also causes the sequence of orders to be relevant. We use this in our advantage and develop three different caching strategies. Retrieving a sequence is the first step of our algorithm, but saving a sequence is the last step in the load validation process.

The second process consist of a set of checks that are relatively fast, to filter any clearly infeasible solutions from going to actual load validation. These checks are called pre-checks and they only check if a combination of orders is infeasible, so regardless of the sequence in which they appear. If the pre-checks do not fail, it does not automatically mean the solution is valid. Hence, pre-checks can only check for infeasibility. If a combination of orders is invalid, it will be saved in the caching function. We have included three different pre-checks on restrictions that typically are exceeded. Section 4.3 further explains these pre-checks.

The third and last process is the load validation phase, which only gets activated if the caching function did not find a pre-known sequence and the pre-checks did not fail. The basis for load validation is a multi-branch tree traversal algorithm. The multi-branch tree consist of nodes that represent the orders of the sequence the load validation algorithm checks. Every node consist of a number of branches descending into the tree to the next node, and one branch going up the tree to the previous node. The number of branches going down the tree is equal to the number of compartments of the resource combination. The branches are numbered in the same way as the compartments of the resource combination. We say the branches represent the compartments. For instance, when a node is attached to the second branch of the preceding node, it means that the order is assigned in the second compartment of the resource combination. In the end, when we allocate the last order, we have reached the bottom of tree and are at one of the leaf nodes of the tree. From there, we go back up to the root node, by only moving up in the tree. The branches in that path represent the compartment location of the corresponding order. Note that the number of leaf nodes of the tree equals the solution space as we defined it in Section 4.1. Every leaf node has exactly one path back up to the root node, and every path equals one unique solution to allocate the orders over the compartments. Section 4.4 provides a full explanation of the algorithm.

Figure 4.1 represents the complete process of our solution method. Note that the caching function and the load validation algorithm can return both feasible and infeasible, while pre-checks only can find an infeasible solution. It is possible to interchange the positions of the pre-checks and the caching function.

Depending on the computation time of both procedures, it may be favourable to have the pre-checks before the caching function. However, in our experiments (See Chapter 5) we have not found an indication to switch the two processes.

4.3 Pre-checks

This section describes the three pre-checks we developed. The idea behind a pre-check is to verify easily discoverable violations, before the full feasibility check is done (hence the "pre"). As described in Section 4.2, if a pre-check does not fail, it does not automatically mean the sequence of orders is feasible. The three pre-checks run consecutively, and when one of them fails, it directly returns to the vehicle routing algorithm, as it makes no sense to continue.

Total capacity: A route always starts with picking all orders, before delivering them. So all pick-ups are completed before the truck performs a delivery. Therefore, we know that there exists one moment at which all orders are inside the resource combination. If the sum of all orders' weights or volumes exceed the capacity of the resource combinations, we know it does not fit. This answer is solvable in linear time.

Compartment type is unavailable Every order has a product type that pairs with an equivalent compartment type. This is one of the characteristics of the retail case as described in Section 4.1. If one of the compartment types of any order in the route is unavailable inside the resource combination, one of the orders is not allowed to be transported. Therefore the route is not feasible. This answer is solvable in linear time.

Not enough compartments Certain orders are not allowed together in one compartment, because of order characteristics. For instance, an order has to be frozen while transported. We argued that there at most three of these types, called Ambient, Chilled, and Frozen. If all three types are included in the route, we know for sure that we need at least three compartments. After all, only orders with equal compartment types are allowed together in one compartment. However, two orders of the same compartment type can be banned from the same compartment, if the contamination restrictions are active. Meaning, even though they need the same compartment characteristics, they still need to be separated through different compartments. So if an order containing frozen fish is not allowed together with frozen bread, we need two separate frozen compartments. This means, we need at least four compartments if we also have orders with chilled and ambient product types. Therefore we can count the minimal number of available compartments. If the total number of compartments available is less than the number of compartments that are needed, it is clear that this combination of orders does not fit into the resource combination. This is solvable in linear time.

4.4 Load validation

This section describes the final process of load validation. Prior to load validation, the algorithm has gone through the caching function and the pre-checks. These two process have been added to speed-up the process, but they are not a critical part of the algorithm. So if they were to be disabled, we still get the correct answer via the load validation process, but at the cost of an increase in computation time. In general, the algorithm of load validation keeps searching for a solution, until it finds a feasible solution or

it is certain that all possible solutions are explored, but no feasible solution has been found. The origin of our algorithm is a multi-branch tree traversal, also known as a tree search algorithm. The reason to use a tree search algorithm is that we are able to successfully ignore infeasible branches in the tree, early during the search. Therefore we can greatly reduce the solution space and therefore the computation time, one of the goals of this research.

In the remainder of this section, we discuss the exact functioning of the load validation algorithm. In Section 4.4.1 we describe the initialization of the data before we start the tree traversal algorithm. In Section 4.4.2 we present the tree traversal algorithm of load validation. The decision if a compartment is feasible or not, is done by the feasibility check and we describe this in Section 4.4.3. In Section 4.4.4, we finish by describing the two tree search strategies of the algorithm.

4.4.1 Initialization

Section 4.2 gave an overview of the algorithm and a brief explanation of the load validation algorithm. Figure 4.2 presented four different calls received by the load validation algorithm that were generated by the vehicle routing algorithm algorithm. For every call, the load validation algorithm needs to validate the sequence in the given resource combination. To explain the initialization of the algorithm, we isolate one of these sequences and we use this as an example. Figure 4.3 presents this isolated sequence, which is equal to call "10245" of Figure 4.2. The sequence consist of five orders, picked at two depots and delivered at three different locations. Note that the delivery sequence is not equal to the reversed picking sequence. Besides the sequence of the orders, we also know the characteristics of the orders and the characteristics of the resource combination. In this section we explain three initializations of the data, before we move to the tree traversal algorithm.

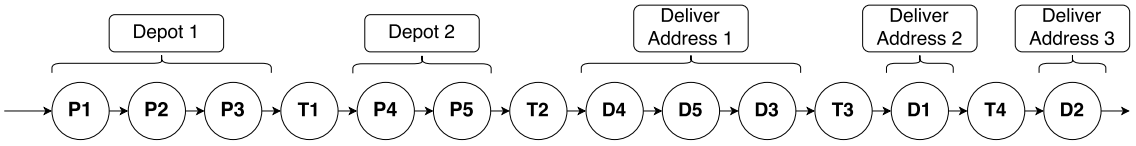


FIGURE 4.3: Schematic representation of a sequence of orders. An order (o_i) consist of a pick-up (p_i) and a deliver d_i. Travel t_j separates the different location, so we know which orders are loaded and unloaded together.

The first step of the load validation process is to segregate the deliveries from the other tasks and reverse its sequence for reasons we explain shortly. In our example, this would give the sequence of O_2, O_1, O_3, O_5, O_4. Note that the O stands for order and the subscript numbers correspond to the delivery and pick-up number. For now, we ignore the travels between the deliveries. The algorithm uses this reversed sequence to assign the orders one by one into the compartments of the resource combination. This means we first try to find a feasible compartment for order O_2 in the empty resource combination. If a feasible compartment has been found, the remaining capacity of the compartment and respective resource decreases. Next we try to assign order O_1 in the resource combination, given the situation of order O_2 being assigned.

We assign in a reversed sequence, because the situation of having to deliver order 2 after order 1, 3, 5, and 4 is equal to delivering solely delivery 2. The same holds for the situation when we need to deliver order 2 and 1 after order 3,5, and 4. This means that if we conclude that it is infeasible, for example, to

assign order 2 to compartment 1, all allocations of order 1, 3, 5, and 4 following the allocation of order 2 in compartment 1 are infeasible as well. Therefore we do not have to consider any of the allocations of order 1, 3, 5, and 4 given the allocation of order 2 in compartment 1, which saves time as we can safely ignore them. This reasoning holds for every sequence of orders, if we would only look at the "positioning" requirements as described in Section 3.4. Meaning the sequence does not influence the feasibility of the combination of orders. For instance, we take our example sequence and swap order O_2 and O_1. Now we need to assign order O_1 first, so let us assume only compartment 3 is feasible for this order. If we now try to assign order O_2 in compartment 1, it is still infeasible regardless in what compartment we assigned order O_1.

However, in the retail case we also consider the "container-related" requirement of weight distribution among two resources. This causes that the internal sequence affects the feasibility as well, meaning a different sequence of pick-up and/or deliveries can affect if a route is feasible or not. For example, if we would have three orders picked at one depot but delivered at three different addresses with a resource combination of two resources. All orders weigh 1000 kilograms, but due to volume restrictions it is only possible to transport order O_1 and O_2 in the front resource and order O_3 in the rear resource. If we have a sequence where order O_3 is the final delivery, we can immediately reject this solution as the last travel would be infeasible due to violation of the weight distribution constraint. However, in case order O_3 is the starting delivery, the sequence would have been feasible.

The weight distribution constraint should also be feasible between the picking addresses. Because we want to know as soon as possible if an allocation of orders is infeasible, we initialize at which insertion of an order, the algorithm needs to validate this weight distribution. In our example, we know that after the pick-up of order O_1, O_2, and O_3, we need to have a valid weight distribution. When we look at the reversed sequence of orders, we know after inserting order O_3 the locations of all tree orders. At that point we can determine if the allocation so far, is valid for travel T_1. This validation, together with all other validations is done at the feasibility check of the algorithm. We explain the functioning of this feasibility check in Section 4.4.3.

When orders are only allowed in one compartment due to dissimilar compartment and product types, we pre-assign these orders to the only allowable compartment. In this way we can reduce the compartment and resource capacities before the start of the tree traversal algorithm. So if the last order to be assigned is as large as the first compartment, which is also the only allowed compartment for this order, we do not have to consider the first compartment for all other orders. In the following section we discuss how we insert the reversed sequence into the multi-branch tree.

4.4.2 Algorithm

The basic structure of our tree traversal algorithm is presented in Algorithm 1. As mentioned before, the load validation algorithm starts with reversing the sequence of deliveries. In this way, the last order to be delivered, is the first order to be assigned to a compartment. Figure 4.4a shows the reversed sequence of orders of our example. After reversing, the algorithm tries to assign the orders one by one to the compartments. Figure 4.4b shows the empty tree structure, with three branches representing the three compartments of the resource combination. The resource combination is empty before the first order is assigned. We have two strategies of choosing the compartment to assign an order. For now we use the most simple strategy, meaning we always try to first assign the order in the most front compartment of

the resource combination. We have a separate feasibility check that validates if an order can be assigned to a compartment, given the orders that already have been assigned. This feasibility check returns to the load validation algorithm if it allows to add the order to the compartment. Based on the feasibility, the algorithm continues with trying another compartment, or moving to the next order.

Algorithm 1 Tree Traversal

```
1:  procedure INITIALIZE:
2:      reverse the sequence of orders
3:      pre-allocate orders in compartment                    ▷ Only if possible
4:  end procedure
5:
6:  procedure TREE TRAVERSAL ALGORITHM:
7:      current order becomes first order
8:      while solution not found OR infeasibility found do
9:          for c in compartments do          ▷ Sequence of compartments depends on strategy
10:             if current order is feasible in compartment c then          ▷ See Algorithm 2
11:                 assign order into compartment
12:                 update values of compartment and resource
13:                 current order becomes next order                    ▷ End of for-loop
14:             end if
15:         end for
16:
17:         if no compartment is feasible for current order then
18:             remove previous order from its current compartment
19:             cut the branch of the previous order in the current compartment from tree
20:             current order becomes previous order
21:         end if
22:
23:         if current order is last order AND current order is feasible in a compartment then
24:             solution found!                    ▷ Algorithm ends, Reply: "Feasible"
25:         end if
26:         if current order is first order AND all branches are cut then
27:             infeasibility found!               ▷ Algorithm ends, Reply: "Infeasible"
28:         end if
29:     end while
30: end procedure
```

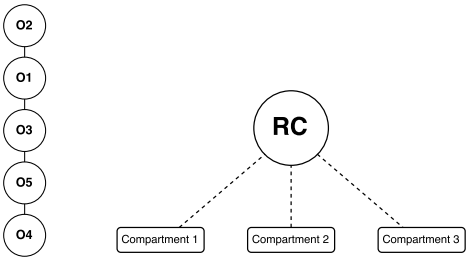


FIGURE 4.4: (a) The sequence of orders we want to insert into the tree. (b) The root node of the tree. It represents the resource combination (RC) in where we arrange the orders.

If it is infeasible to assign the order to the compartment, the branch representing this compartment is marked as infeasible. This does not mean we cannot assign other orders to this compartment. It means that all solutions following from the decision to put the order in that compartment, are infeasible and therefore ignored. The algorithm looks if there are other compartments that have not been visited, and tries to assign the order to next compartment. In Figure 4.5a the algorithm tries to assign order

O_2 in compartment 1, without any success. Figure 4.5b shows that the first compartment is infeasible for order O_2, and it has moved to the second compartment. If it finds a different compartment that allows the order to be added, the order is assigned in the way we explain in the next paragraph. If all compartments are infeasible, the algorithm moves back to the parent node, and looks for compartments that have not been visited before. The algorithm returns infeasible to the vehicle routing algorithm, if the algorithm reaches the root node, with all branches to the child nodes marked as infeasible. At that point we know that within all feasible assignments of orders over the compartments, there does not exist a feasible solution with all of the orders assigned inside the resource combination.

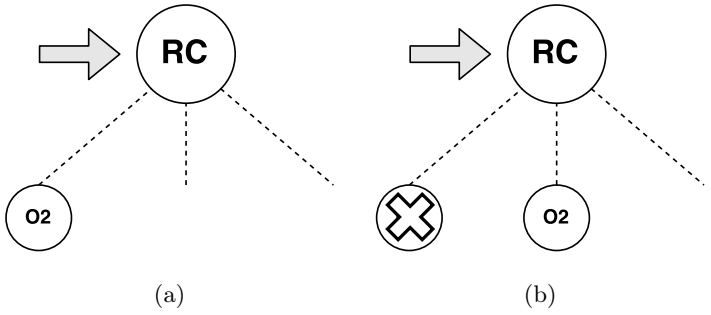


FIGURE 4.5: A infeasible insertion of an order into a compartment. (a) The order O_2 is assigned to the first compartment. (b) This is infeasible, so the algorithm tries the next compartment. The arrow represents the node we are currently looking to insert the order.

If it is feasible to assign the order to the compartment, the order is assigned to the compartment and we move to next order. Within the algorithm, this means we insert the order as a child node at the parent node. In our example of order O_2, the parent node would be the root node of the tree. The point of insertion in the parent node corresponds with the compartment we want to assign the order in. Therefore, we know at the child node where it was assigned to. Figure 4.6 shows the insertion of order O_2 into the first compartment. The resource combination was still empty at the root node, but now the first order gets assigned to the resource combination. This affects the remaining capacity and compatibility of the compartments. The configuration of the resource combination after the order is assigned, is saved on node itself. When the algorithm hits a point when an order cannot be assigned to any of the compartments and moves back to the parent node, it can easily retrieve the configuration of the parent node, and try a different path.

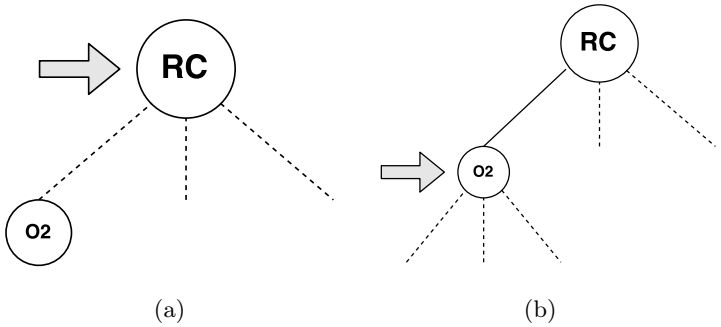


FIGURE 4.6: A feasible insertion of an order into a compartment. (a) The order O_2 is assigned to the first compartment. (b) This is feasible, so the algorithm moves down with the order, represented as the arrow pointed at the node. Again, order O_2 has three branches representing the three compartments.

The load validation algorithm returns feasible to the vehicle routing algorithm, if the last order, so the first delivery, is successfully assigned to one of the compartments. At that point we know it possible to

fit all orders within the resource combination. It is possible there exist other assignments besides the one we found, but we are not interesting in optimizing the load allocation. We are interested if it is feasible or not.

Figure 4.7 shows a possible tree structure of our example. These five orders are feasible inside the resource combination of three compartments. We first assign order O_2 in compartment 1 which is feasible to do. However, due to capacity constraints order O_1 does not fit in compartment 1, but is not allowed in compartment 2 and 3 as well. Therefore, the algorithm moves back to the parent node RC. From there, the algorithm tries to assign order O_2 into the second compartment. Eventually, it takes 12 allocations to find a feasible solution. Figure 4.8 shows how the orders are eventually assigned inside the resource combination.

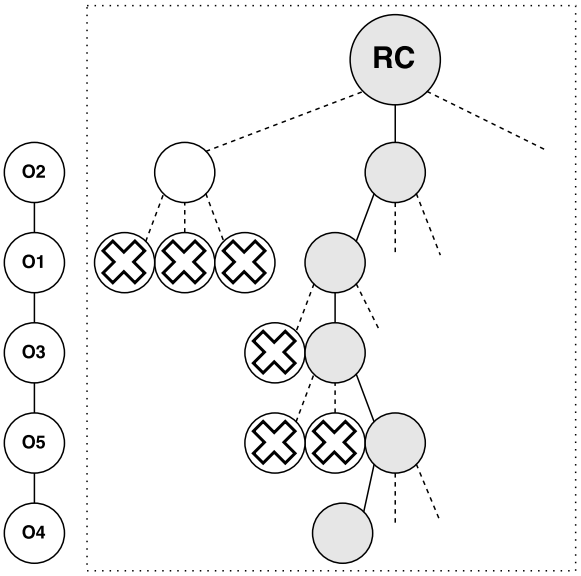


FIGURE 4.7: An example of tree structure of a feasible assignment. The five nodes on the left correspond with the five orders we want to assign. Inside the dotted box is the constructed tree.

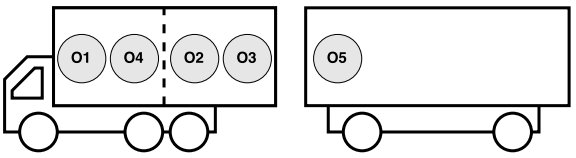


FIGURE 4.8: The resource combination with a front and rear resource. The orders inside the truck correspond with assignment you find in Figure 4.7.

4.4.3 Feasibility check

In this section we discuss the feasibility check of the algorithm, which validates if an order can be assigned into the compartment. Algorithm 2 presents this feasibility check. The feasibility starts when an order is added to a compartment. Previously assigned orders in the compartment are known, so we know if contamination could occur and how much spare capacity is left. First, we check if the compartment type matches the product group of the order. Second, we check if the order contaminates with previously assigned orders. Third, we look at the weight distribution on the travels to all the delivery addresses.

Beforehand, we know before which deliveries a travel occurs. In our example this would be the travels T_2, T_3, and T_4, which are before the orders O_2, O_1, and O_4. When we assign these orders to a compartment, we need to check if the foremost resource (truck) is still heavier then the rearmost resource (trailer). Because we assign the orders back to front, we know at the time of the insertion the location of all orders that are present.

We use a similar approach with checks of the weight distribution on the travels between the depots. However, we need to do a preprocessing step at the start of the load validation algorithm. Before assigning any order, we check which of the orders of depot 1 would be the last order we assign to the resource combination. When we insert that order, we know the location of all the other orders of the depot and therefore we can check if the weight distribution is still valid. If we would have more than two depots, we would not only look at the depot itself, but all preceding depots as well. So if we have three depots, we look at the last order to complete depot 1 and at the last order to complete the set of orders of depot 1 and depot 2. If we check the weight distribution between depot 2 and depot 3, we first have to remove all orders we have assigned so far who are not from the depot location 1 and 2. In our example it would be after O_3, that we check for the weight distribution between depot 1 and depot 2. The main advance of this approach is that we can identify infeasible arrangements of orders early on in the search.

The last four checks have to do with the capacity of the resource combination. It is possible for the planner to assign a maximum capacity to both the resource as the compartment. On the resource and compartment, we check for both the weight as the volume capacity. If no capacity has been assigned, we can simply skip it. Note that we do not continue with the other checks, if come across one that fails. We directly return to the load validation algorithm with the answer this compartment is not possible.

Algorithm 2 Feasibility check

```
1:  procedure ASSIGN ORDER TO COMPARTMENT
2:      procedure CHECK FOR THE FOLLOWING:
3:          order product type and compartment type are compatible?
4:          no contamination with other orders?
5:          weight distribution on the travels to all the delivery addresses?
6:          weight distribution on the travels between the depots?
7:          weight capacity of compartment?
8:          weight capacity of resource?
9:          volume capacity of compartment?
10:         volume capacity of resource?
11:         if lines 3 to 10 are valid then
12:             return: order is feasible in compartment.
13:         else
14:             return: order is infeasible in compartment.
15:         end if
16:     end procedure
17: end procedure
```

4.4.4 Compartment strategies

This section describes the two strategies for choosing the sequence of compartments in which we try to assign the order. At every node in the tree structure, we have the choice to assign the order in any of the c compartments. The two strategies are the front-first strategy and the history based strategy. The front-first strategy is the default strategy. The second strategy uses historical data of sequences that were successful before, and uses this information for a similar sequence. This strategy is only active for longer sequences. For shorter sequences the time spent on pre-processing, could outweigh the time we save. Besides, shorter sequences often use only a small amount of the capacity and have less contamination problems, and therefore we find relatively fast a solution anyway.

The front-first strategy always starts with foremost compartment. If this compartment is infeasible, it takes the second foremost compartment. As long as no compartment is feasible, it continues until it reaches the rearmost compartment. This strategy is simple but effective, as it tries to assign the orders that are delivered last as far as possible to the front. We want these orders to be in the front, because of the weight distribution restrictions.

We name the strategy that uses historical data of sequences the prediction-based strategy. During an optimization call, a large amount of sequences are evaluated. The algorithms of the vehicle routing part use search heuristics that adjust the sequences little by little. Therefore it is common to have quite similar sequences that need to evaluated by the load validation algorithm. It is likely that similar sequences have similar solutions. If we know beforehand which orders should go in what compartments, this could save a tremendous amount of time.

The idea is that when a sequence in specific resource combination has been successful before, a similar sequence also succeeds when the orders are assigned to the same compartments as the successful sequence. During the tree search algorithm we first try those compartments. If the prediction-based compartment turns-out to be infeasible, then we try the other compartments. To find a similar sequence, we look into all successful sequences that are at least a certain length. We use a tree structure to store all sequences, similar to the tree structure we used to do the tree search. Even though it looks similar, note that they use a tree structure for different reasons. The tree traversal algorithm uses the tree structure for an efficient search of all relevant allocations, while the prediction-based strategy uses a tree structure for efficient storage and specific search characteristics.

The main difference between both tree structures is that in case of the prediction-base strategy the number of child nodes can gets as large as n orders. Resource combinations differ among their characteristics, which has an effect on the feasibility of sequence of orders. Therefore, we use multiple trees, to differentiate between the different resource combinations. The root nodes of these trees represent the different resource combinations. Figure 4.9 gives a representation of how the current sequence needs to match a prior successful sequence. The top sequence in both figures is the one which has been stored. The bottom sequence is the one we compare it to. We start at the root node (RC), and look if every following node matches the node of the stored sequence. As long as the stored sequence and the current sequence match we continue. If the current sequence has a node that does not correspond with the stored sequence, it skips the node and checks if the next nodes corresponds. If a node of the stored sequence does not correspond with one of the nodes of the current sequence, the algorithm responds that no prediction-based sequence can be found. If that is the case, the tree traversal algorithm falls back to the front-first strategy. The algorithm only allows for one skip. If a previous stored sequence has been

found, the algorithm assigns the compartments to the orders that are both in the historical sequence and the current sequence. Then, the tree traversal algorithm first tries to assign the order to the same compartment as it was successfully assigned before. The algorithm tries orders that were not assigned in the historical sequence based on the front-first strategy. If the becomes infeasible to assign an order into the compartment based on the prediction, the tree traversal algorithm also falls back on to the front-first strategy.

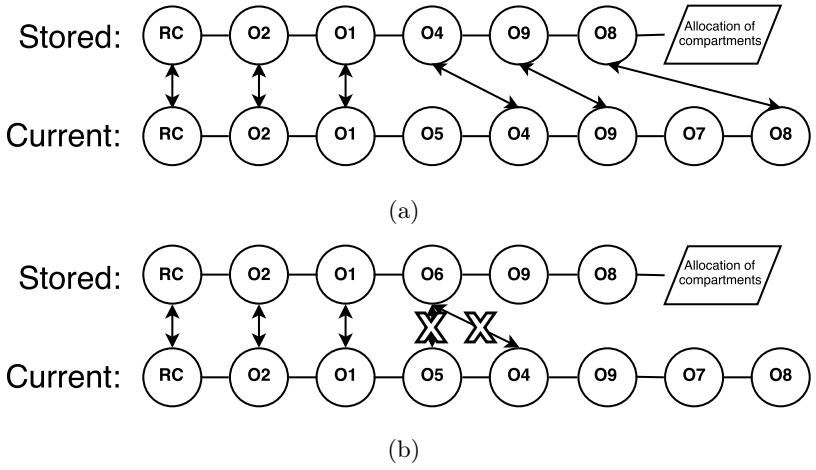


FIGURE 4.9: Two examples of the prediction-based compartment strategy. The sequence of five orders is in both examples the stored sequence. The two sequences of seven orders are the current sequences for which the algorithm searches a prediction.

4.5 Caching function

In this section we describe the caching function. Both the literature as well as preliminary experiments show that many of the calls to the load validation algorithm are duplicates. We introduce the caching function to cache all sequences (both feasible and infeasible), so we can determine in a quick way if a sequence is a duplicate. Besides duplicate sequences, we argue that if certain combination of orders fails, the set of combinations that have that specific combination of orders as subset, fails as well. All of these sets, are called supersets, the inverse of a subset.

The caching function is based on a tree structure, a technique we also used in the load validation algorithm (Section 4.4) and the prediction-based strategy (Section 4.4.4). In Section 4.5.1 we discuss the method of storing and retrieving a sequence in the normal caching function. In Section 4.5.2 we describe the method of storing and retrieving a sequence, but then to successfully search if any combination of a subset of a sequence has failed before (supersets). As mentioned before, storing a sequence is the last step of the load validation algorithm, while retrieving a sequence is the first step. However it makes more sense to explain first how a sequence is stored, before explaining how it is retrieved.

4.5.1 Default caching function

Every resource combination has different characteristics, such as volume, weight, and contamination. Therefore, instead of storing all sequences together in one tree, every resource combination has its own

Algorithm 3 Compartment strategies

```
1: procedure FRONT-FIRST STRATEGY:                                   ▷ Default strategy
2:     start with foremost compartment
3:     while order is not feasible in compartment do
4:         try foremost not tried yet compartment
5:     end while
6:     stop if compartment is found or no compartment is feasible
7: end procedure
8:
9: procedure PREDICTION-BASED STRATEGY:                              ▷ See Figure 4.9
10:    get reversed sequence of orders
11:    get tree with root node of corresponding resource combination
12:    current order becomes first order
13:    current node becomes root node
14:    while current order is not last order do
15:        if current order is among children of current node then
16:            current order becomes next order
17:            current node becomes child node
18:        else if next order is among children of current node then
19:            current order becomes next "next order"
20:            current node becomes child node
21:        else
22:            return: no prediction found                           ▷ Use front-first sequence.
23:        end if
24:    end while
25:    if current order is last order AND current order has a flag then
26:        return: prediction found, return previous allocation of orders over compartments
27:    else
28:        return: no prediction found                               ▷ Use front-first sequence.
29:    end if
30: end procedure
```

tree structure for storing the relevant sequences. The resource combinations is the root node of the tree, meaning they are the top of the tree structure. Whenever a sequence needs to be stored, the algorithm first searches for the root node that corresponds with the resource combination. Note that the sequence stored is the sequence of all pick-up and deliveries. This is different compared to the tree search algorithm of Section 4.4.2. If the tree already contains other sequences, the algorithm skips the children who already exists and starts inserting at the point where the tree differs from the inserted sequence. At the end of the branch, a flag is inserted with the result of the validation algorithm. This flag is set to either feasible or infeasible. Figure 4.10a depicts a representation of the process and Algorithm 4 shows the process of storing a sequence.

Algorithm 4 Default caching function

```
1: procedure STORING SEQUENCE
2:     get reversed sequence of pick-up and delivery nodes
3:     get tree with root node of corresponding resource combination
4:     current order becomes first order
5:     current node becomes root node
6:     while current order is among children of current node do
7:         current node becomes child node of current order
8:         current order becomes next order
9:     end while
10:    while last order is not inserted do
11:        add node as child current node
12:        current node becomes child node
13:        current order becomes next order
14:    end while
15:    Insert flag                                  ▷ Either feasible or infeasible
16: end procedure
```

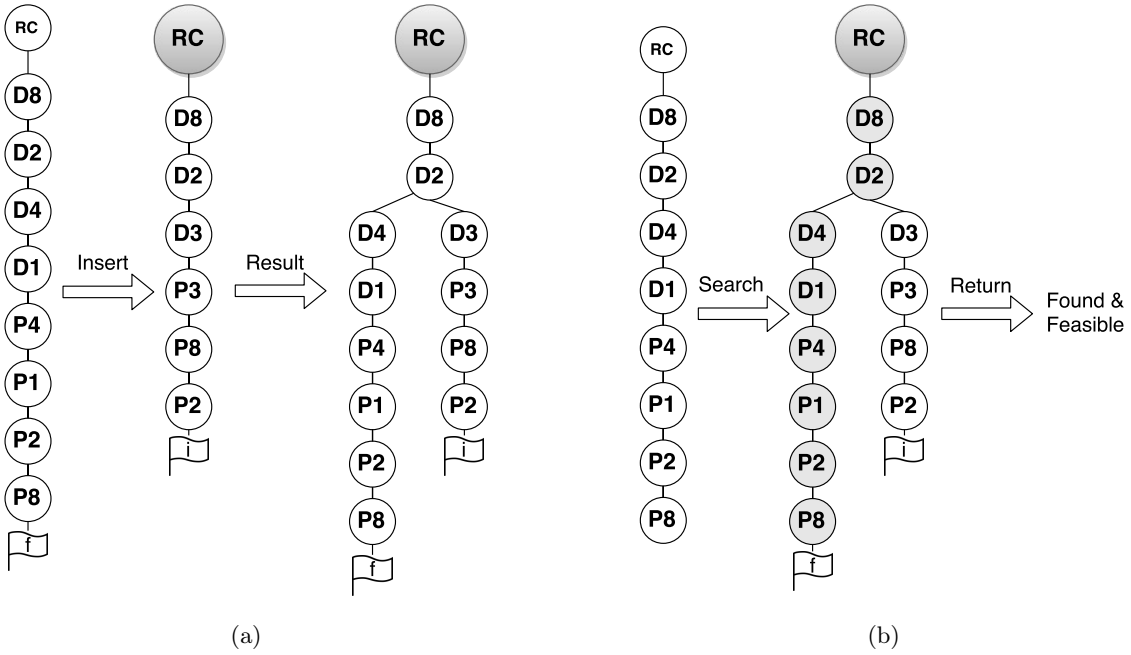


FIGURE 4.10: (a) Storing a sequence in the tree. The flag under a sequence represents if the sequence is feasible (f) or infeasible (i). (b) Searching a sequence in the tree.

Retrieving a sequence works the opposite of storing a sequence. First, the algorithm gets the root node of the corresponding resource combination. The algorithm searches for the first node of the sequence among the children of the root node. If the node exists in the children of the root node, it moves to next node of the sequence, and searches among the children of the current node of the tree. If a node is not found among the children, it returns with no solution found. If the last node in the sequence is found in the tree, it returns the flag of the current node in the tree together with the message that it found

a solution. Figure 4.10b depicts a representation of the process and Algorithm 5 shows the process of retrieving a sequence.

Algorithm 5 Default caching function

```
1: procedure RETRIEVING SEQUENCE
2:     get reversed sequence of pick-up and delivery nodes
3:     get tree with root node of corresponding resource combination
4:     current order becomes first order
5:     current node becomes root node
6:     while current order is among children of current node do
7:         current node becomes child node of current order
8:         current order becomes next order
9:     end while
10:    if current node is last node then
11:        get flag                          ▷ Either feasible or infeasible
12:        return "Found"
13:    else
14:        return "Not found"
15:    end if
16: end procedure
```

4.5.2 Superset caching function

In Section 4.5.1, we discussed a technique that only works for sequences that are an exact duplicate of each other. Another technique would be to look for an infeasible subset of orders within the sequence that is being called. This means that if a combination of three orders failed before, we know all sequences (in identical resource combinations) containing that combination of three orders will fail as well. This technique only works if we are certain that a failed subset causes the called sequence to fail. We refer to all sequences that contain a specific combination of orders (or subset), as being a superset of that combination of orders. A superset is regarded as the inverse of subset; hence the name superset caching function.

In the retail case we know that if one of the positioning-related restrictions fails on a sequence, we are certain that every sequence that contains that combination of orders (in any order), fails as well. Reshuffling the orders has no influence on the feasibility of the positioning-related restrictions, because the restrictions only focuses on the characteristics of there orders, like weight and volume. Also, if a positioning-related restriction is violated for a certain combination of orders, it is impossible that the restriction becomes feasible by adding more orders. For example, if a compartment exceeds its capacity, we cannot fix this by reshuffling the sequence or adding more orders. However, if a sequence fails by a container-related restriction, it can be counteracted by a reshuffle or adding more orders. For example, if on one of the travels the weigh balance is incorrect, we can reshuffle the sequence or add an order to fix this. Therefore, if we know the weight balance never failed during the load validation procedure, we know only positioning-related restrictions failed and we can apply this technique. The pre-checks are based solely on positioning-related restrictions, an not on container-related restrictions. This means that

the sequences that already fail during the pre-checks, can be stored in the superset caching function as well.

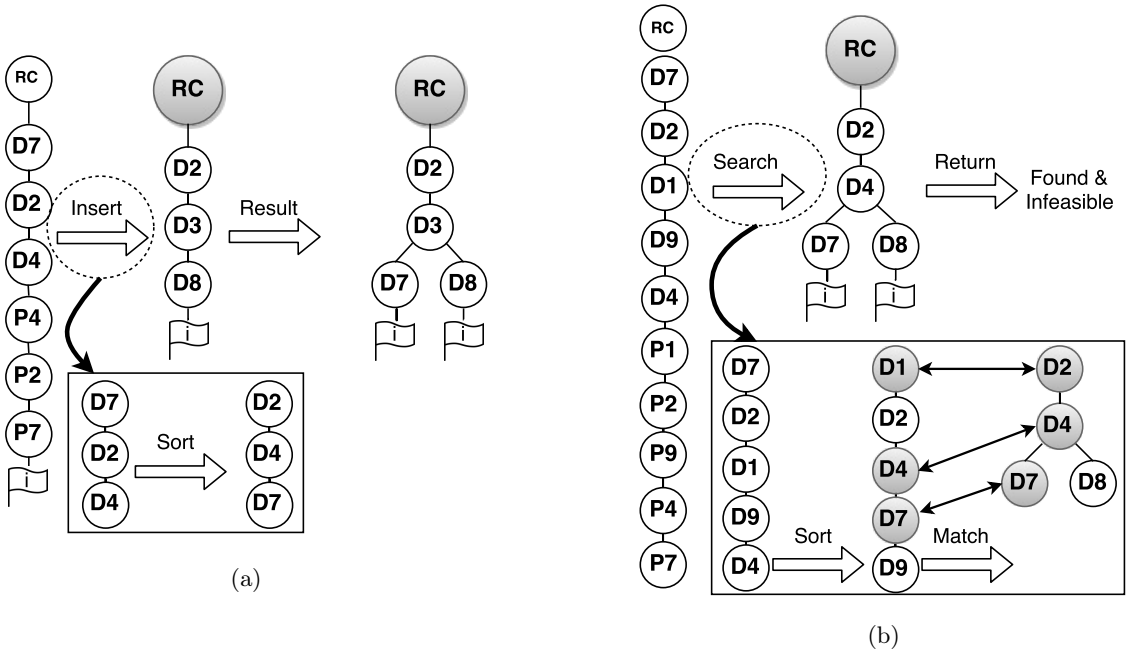


FIGURE 4.11: (a) Storing an infeasible sequence of three orders in the (superset) tree. Note that we sort the sequence and we only store the delivery sequence. (b) Searching a sequence of 5 orders in the superset tree. In this example the sequence of the call has a subset that matches with one of the branches of the tree.

We developed a method to find if a sequence has an infeasible subset as shown in Algorithm 6. We accomplish this by ordering the delivery nodes, before inserting them into one of the separate trees corresponding to the resource combinations. The way to order them is not important, as long as it is done consistent. Our method sorts the delivery nodes in ascending order based on the delivery id. The method inserts this ordered sequence into the corresponding tree, just as the default caching function does, only without reversing the sequence. When we want to check if a sequence has a subset that is infeasible, we also arrange the deliveries in the same way as the sequences that were stored. Then we look if the child nodes of the root node of the resource combination match with the nodes of the sequences. If so, we look if those nodes have children that are the same as any of the preceding nodes of the sequence. We keeping on searching, until one of the nodes in the tree has a flag that hits failure. If so, we found a subset of the sequence and know it will fail as well.

Algorithm 6 Superset caching function

```
1: procedure RETRIEVING SEQUENCE
2:     get ordered sequence of only the deliver nodes, remove pick-up nodes   ▷ Ascending order
3:     get tree with root node of corresponding resource combination
4:     current order becomes first order
5:     current node becomes root node
6:     for current order to last order do                                     ▷ Recursive function
7:         if current order is among children of current node then
8:             if child node has flag then
9:                 return: Subset found, infeasible!                          ▷ Sequence is infeasible
10:            end if
11:            current order becomes next order
12:            current node becomes child node
13:            move to line 6                                                 ▷ Recursive part of function
14:        end if
15:    end for
16:    return "Not found"                                                     ▷ Nothing found
17: end procedure
```

4.5.3 Conclusion

In this chapter, we developed our load validation algorithm. First, we discussed the solution space of the problem. This solution space got extremely large, due to the large amount of choices that influenced the configuration of the resource combination. By deducting the configuration from the allocation of the orders, we were able to reduce the solution space to a maximum c^t, with c as the number of compartments and t as the number of compartment types. The algorithm consists of multiple processes. These processes operate consecutively, which means that if one process finds an answer, the algorithm stops and the remaining processes are skipped.

Second, we developed two caching functions. These functions cache all sequences that have been calculated before, to prevent the algorithm to calculate something twice. The first caching function looks only for exact duplicates of the sequence, so it keeps track if a sequence is either feasible or infeasible. The second caching function keeps track of only infeasible sequences, and only saves the sequence if it solely failed on positioning-related restrictions. If so, all sequences that contain any combination of orders of the failed sequence, fail as well. It therefore checks for every sequence if any subset of orders failed before. This method ensures that much less sequences need to be validated by the actual load validation algorithm, than would be necessary if we only check for duplicates.

Third, we developed three pre-checks, that check the feasibility of the sequence relatively fast. We can only determine the infeasibility of sequences. This means that if the sequence is not infeasible, it does not directly mean that it is feasible. These pre-checks are developed to be fast, rather that complete. Next, we developed the tree traversal algorithm, that determines with certainty if the sequence is feasible or infeasible. The tree traversal algorithm starts at the top of the tree (root node), which represents the resource combination in which the orders are placed. Each order is inserted into the tree at the node of the previous order within the sequence. The location of the insertion (branch) represents the compartment

in which it is allocated. After each insertion, the algorithm validates the solution constructed so far, and stops after a feasible solution has been found, or all branches have been scanned. This method is easy to scale and we can increase the number of compartments and orders to any number. Even though the solution space increases quickly, it is safe to ignore a large part of the solution space, after a certain branch gets infeasible.

Chapter 5

Experiments & Results

This chapter describes the experiments to test our algorithm and the subsequent results. In Section 5.1 we discuss the datasets we use in our experiments. Section 5.2 describes the experiments to test our solution method. In Section 5.3 we analyse and discuss the results of the experiments.

5.1 Data

This section describes the data we use for our experiments with the load allocation algorithms. We have access to two datasets of two different customers of ORTEC. These customers are an Australian retailer (retailer 1) and a Russian retailer (retailer 2). Both customers use a custom made algorithm for their load validation, in contrast to the load validation algorithm from Section 1.2.3. Section 3.1 already gave a description of the characteristics of the two companies; in this section we focus on the characteristics of their datasets.

5.1.1 Dataset of retailer 1

Retailer 1 provides a planning set of their distribution operation in Australia. Because Retailer 1 operates in Australia, the travel times and distances are large. The dataset consist of 3002 orders and 266 resource combinations. The number of orders and resource combinations is much larger than retailer 1 plans in one optimization. Therefore they plan the complete dataset in parts, instead all at once. All orders originate from one depot. The average distance between the depot and the 432 delivery addresses is 232 kilometre, with a maximum travel distance of 4000 kilometre. The average travel time between the depot and all delivery addresses is 3.5 hour, with a maximum of 67 hours.

The orders consist of volumes between 1 and 24 pallet spaces. Figure 5.1 shows the distribution of the different volumes of the orders. For reasons explained later, we adjust the original dataset to two smaller datasets, one of 150 orders and one of 300 orders. The distributions of these datasets are also present in Figure 5.1. The majority of the orders has a volume of 9 pallet spaces or less. The orders do not have weight characteristics; retailer 1 assumes that the weight of the orders do not cause any problems during the realisation and therefore neglect the weight. All of the orders have the ambient product type, as the

depot only serves ambient products. All of the 266 resource combinations are single trailer trucks, but they differ in size. Five different type of trucks are available, all consisting of a single trailer. Table 5.1 shows the number of vehicles per type in the original dataset. The two largest vehicle types consist of three compartments that are completely flexible in capacity. Every compartment consist of one of the three compartment types (frozen, chilled, or ambient). This is different from the orders, which are all ambient. The smaller vehicles only consist of one ambient compartment.

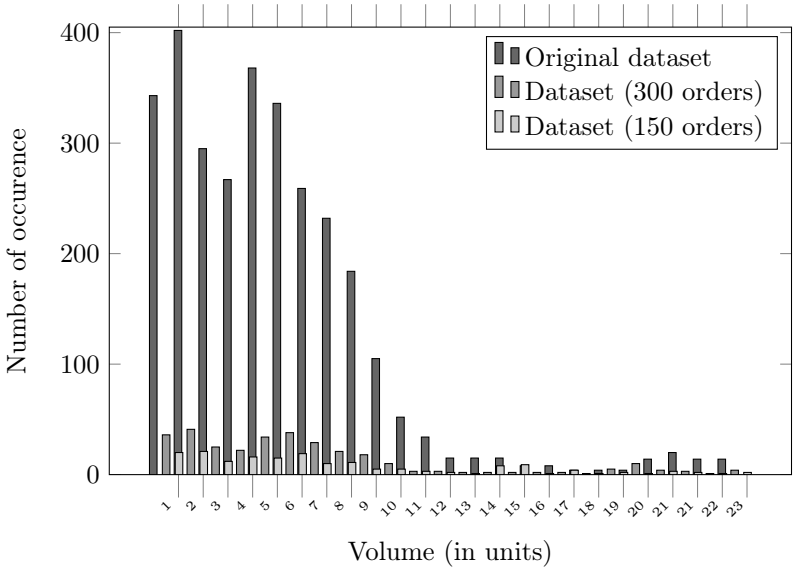


FIGURE 5.1: Occurrence of the different order volumes in the retailer 1 dataset.

Trailer size	**Occurrence**
24 pallets	141
22 pallets	13
16 pallets	59
12 pallets	36
10 pallets	17

TABLE 5.1: Occurrence of the different trailer sizes in the retailer 1 dataset.

The dataset has several active restrictions that are outside the scope of this research. These restrictions interfere with the experiments, it prevents us to properly test our developed algorithms. We remove these restrictions with the purpose to get more reliable results. First, the dataset does not allow for certain combinations of products to be delivered at the same address at the same time. This means they are allowed to be transported in the same compartment, just not unloaded at the same time. This makes it impossible to check for the contamination rules as we developed them. Second, every order, address, and resource combination contains a time window. The vehicle routing algorithm considers the time windows first, before it calls the load validation algorithm. Therefore a lot of routes are rejected before load validation is even considered, and therefore only a limited amount of calls go to the load validation algorithm. Because we are not interested in the vehicle routing algorithm, and the time windows themselves do not influence the functioning of the load validation algorithm, the time windows are removed. Third, all orders in the original dataset have the ambient product group, but because one of the requirements is the use of a contamination matrix, we define a total of three product groups (ambient, chilled, frozen). Within each product group, we define three different product types, creating a total of nine different product types. We randomly assign one of the nine product types to every order in the dataset.

In addition, we make adjustments to the original dataset in order to effectively test the functioning of our algorithms. These adjustments result in 7 different (Retailer 1) datasets. Table 5.2 gives an overview of the 7 different datasets and their characteristics. Dataset 1 is the original dataset as the customer provided it, including unloading restrictions, time windows, and orders only having one product group.

Dataset 2 to 7 have the adjustments as described in the previous paragraph, and are also smaller in size because of high computation times as we show in Section 5.2.1. The smaller datasets consist of either 150 and 13 resource combinations, or 300 orders and 25 resource combinations. The orders and resource combinations in the smaller datasets are randomly selected using an uniform distribution. Datasets 2 and 3 have fixed compartment sizes, because the ILP model does not allow for flexible compartments as we model them. With these modifications, we can compare the results of the current algorithm with the algorithm we propose. The compartments of resource combinations with three compartments are restricted to a maximum of 1/3 of the resource capacity. As a capacity of 22 pallet spaces cannot be divided into equal numbers, we increase the capacity of this resource to 30 pallet spaces.

Datasets 4 and 5 contain a modification that allows us to model flexible compartments in the ILP model. Instead of the algorithm deciding the location of the baffles, we provide the ILP with all possible configurations of the compartments. A configuration of compartments consist of compartment sizes, of which the sum is equal to the total resource capacity. For example, a resource with a capacity of three pallet spaces and two compartments, has four different configurations; [0,3], [1,2], [2,1], and [3,0]. To provide a fair comparison, as well as keeping a reasonable amount of configurations, we test the flexible compartments only on resource combinations with a maximum of three compartments and a total capacity of 2 x 5 pallet spaces. Note that the smallest resource in the original dataset consists of only one compartment, and not three compartments as we modified it. Within dataset 6 and 7, we double the number of compartments to test the use of a contamination matrix. Within the contamination matrix, every one of the nine product types contaminates with each other. This is because we want to prevent a high number of infeasible solutions, caused by the restriction of different product types being together in the same compartment. All of these adjustments are required to test all defined requirements of Section 3.3.

	Orders	Resource combinations	Compartments	Product types
Dataset 1	3002	24p:*141* |22p:*13* |16p:*59* |12p:*36* |10p:*17*	Flexible	A:1
Dataset 2	150	24p:*5* |16p:*4* |12p:*2* |10p:*2*	Fixed	A:3 |C:3 | F:3
Dataset 3	300	24p:*16* |30p:*1* |16p:*5* |12p:*3*	Fixed	A:3 |C:3 | F:3
Dataset 4	150	10p:*13*	Flexible	A:3 |C:3 | F:3
Dataset 5	300	10p:*25*	Flexible	A:3 |C:3 | F:3
Dataset 6	150	48p:*5* |16p:*4* |12p:*2* |10p:*2*	Fixed	A:3 |C:3 | F:3
Dataset 7	300	48:*16* |44p:*1* |16p:*5* |12p:*3*	Fixed	A:3 |C:3 | F:3

Table 5.2: Overview of the original dataset (dataset 1), and the six adjusted datasets used in the experiments to test the algorithms. (p = pallet spaces)

5.1.2 Dataset of retailer 2

The dataset of retailer 2 contains a planning set that represents the deliveries of a week supply to all their stores of one depot. The set consists of 641 orders that need to be divided over 55 resource combinations. As described before, retailer 2 is a large retail chain in the Russia. The average distance between two addresses is 790 kilometre, with a maximum distance of 2515 kilometre between the two furthest addresses. Travel times are on average 18 hours, with a maximum of 55 hours between the two furthest addresses. The 641 orders are distributed over an address set of 250 different addresses, including

3 depot locations. The three depot locations have mutual distances between 10 and 20 kilometre, with travel times between 30 and 65 minutes. The average distance to all other addresses from the depots, is 600 kilometre, with an average travel time of 13 hours.

The orders consist of a volume between 1 and 14 pallet spaces and of a weight between 23 kg and 9150 kg. Figure 5.2 presents this distribution of weight and volume of the order set. The distribution of orders is skewed to the right, meaning there are a larger number of smaller & lighter orders than there are big & heavy orders. Of the 641 orders, 264 have explicit compartment restrictions. Of the 55 resources, 28 are single trucks and 27 are truck-trailer combinations. All resources have only one compartment, so, a compartment and a resource are the same entity. The compartments have a weight capacity between 10.000 and 12.000 kilograms and a volume between 16 and 18 pallet places. Each resource combination has at least one resource with a compartment that is accessible for all orders.

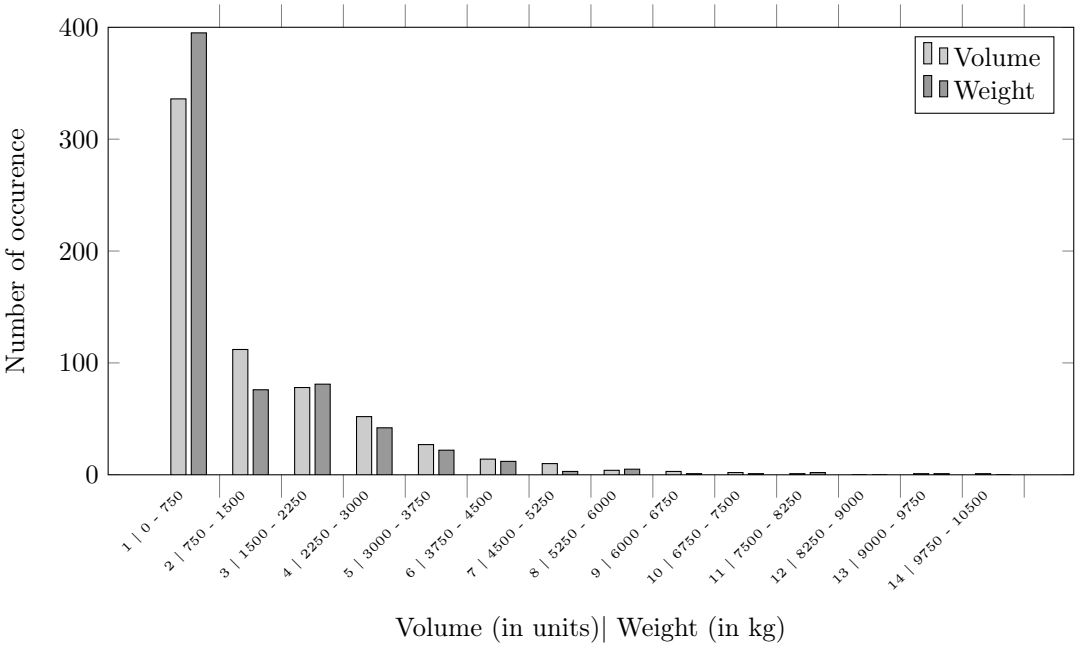


FIGURE 5.2: Order volumes and weight from the retailer 2 dataset.

5.2 Setup of experiments

This section describes the experiments we use to test our algorithm. The experiments are developed by generating optimization runs for the vehicle routing algorithm on a given set of orders and resources. During these runs, the vehicle routing algorithm frequently calls the load validation algorithm to validate the load allocation. We ignore the optimization of the vehicle routing algorithm and focus only on the speed of the load validation part. For every response of the load validation algorithm, we record information of the route (sequence), the feasibility, and speed of the algorithm and its individual parts. In the end, every experiment results in a list of load validation responses, which we use for our analyses.

The experiments run on a laptop with i7-3720QM CPU with a speed of 2600 MHz and 8,00 GB of RAM. It runs on a Windows 7 (64-bit) operating system, and the experiments run on a C++ coded program in Virtual Studio 2012. The datasets use a XML format, which contains all information necessary to

successfully run the experiments. The test saves its generated output into text files, which are analysed using Excel.

Note that the two provided datasets (Section 5.1) are not complementary to each other, which means that we cannot conduct equal experiments on them. The two distributors who provide the datasets use different models for their distribution services. The dataset of retailer 2 does not describe the product types and therefore a contamination matrix cannot be used. Additionally, the retailer 2 dataset does not use compartments to divide their resources, not to mention the flexibility constraints. However, the missing characteristics in the retailer 2 dataset, are present in the dataset of retailer 1. The retailer 2 dataset does support co-loading, weight restrictions, and two resources in a resource combination (truck and trailer). On the other hand, these characteristics lack in the retailer 1 dataset. In addition, both customers have a custom build algorithm to fit their requirements. Because we want to test the performance of the general load validation algorithm (COPSLoad) as well, we decide to test the retailer 1 dataset on solely the general algorithm. This has some consequences on the performance, as some restrictions are hard to validate using an ILP model. However, this is the only way to include the general load validation algorithm into this research using the available datasets.

We divide our experiments into three parts. First, we verify if our algorithm computes the answers we intend it to compute (Section 5.2.1). Second, we test the effectiveness and efficiency of the pre-checks and caching functions (Section 5.2.2). Third, we test the main algorithm by comparing it with the current algorithms (Section 5.2.3). Table 5.3 provides an overview of the numbered experiments we carry out.

Experiment	Number	Retailer 1	Retailer 2
Verification	1	Dataset 1	original retailer 2 dataset
	2	Dataset 3,5	original retailer 2 dataset
	3	Dataset 3	original retailer 2 dataset
Caching function	4	Dataset 3,5	original retailer 2 dataset
Pre-checks	5	Dataset 3,5	original retailer 2 dataset
Tree traversal	6	Dataset 2,3,4,5	original retailer 2 dataset
	7	Dataset 6,7	n.a.

Table 5.3: Overview of the experiments. Section 5.2.1 to Section 5.3.3 explain the experiments in detail.

5.2.1 Verification

First, we use the original datasets to test if they provide realistic results (experiment 1). We present the results of this experiment in Section 5.3. However, we encounter such long computation times (+-4 hours) during the first experiments on the retailer 1 dataset, that we decide to reduce the size of the retailer 1 dataset for the remaining experiments. We reduce the size of orders and resource combinations to two sizes, namely 150 orders in 13 resource combinations and 300 orders in 25 resource combinations.

In the second experiment we test if the tree traversal algorithm provides the same answer as the original load validation algorithms. We expect both load validation algorithms to respond the same to an identical call. Because the vehicle routing algorithm is deterministic, it generates identical calls on the same dataset. This results in an equal total number of calls by the vehicle routing algorithm. If the number

of responses or the responses themselves are different, it means the algorithms give at least once an opposite answer to a call. This means one of the algorithms does not function as it was designed. We use datasets 3 and 5 from retailer 1 to check if both fixed and flexible compartment settings match the ILP model. We use the original dataset of retailer 2 to check if our load validation algorithm produces the same answers as the custom build version algorithm of retailer 2.

Third, we verify if the computation time found is consistent (experiment 3). This is to verify if our test method is consistent, and we use the right set-up. We accomplish this by running the experiment three times on the same dataset, and compare their different outcomes. We measure the mean and standard deviation of the total computation time, the correlation coefficient over all responses, and the sum of all absolute differences of each individual call.

5.2.2 Caching functions & Pre-checks

In the fourth experiment, we test the speed and functioning of the caching functions. The goal is to study how efficient and effective these caching functions are. We measure the number of calls that the caching functions prevents from going to the actual load validation algorithm, as well as the computation time. In the experiment of the caching functions, the pre-checks will be active as well. In this experiment, the pre-check will only check the sequences that were not prevented by the caching function. We add them because they do not influence the results of the caching functions, and this gives a complete picture of all three elements together. Because we also want to test how effective the pre-checks are by themselves, we run a separate experiment without the caching functions. This is experiment 5, where both datasets of retailer 1 and retailer 2 are used.

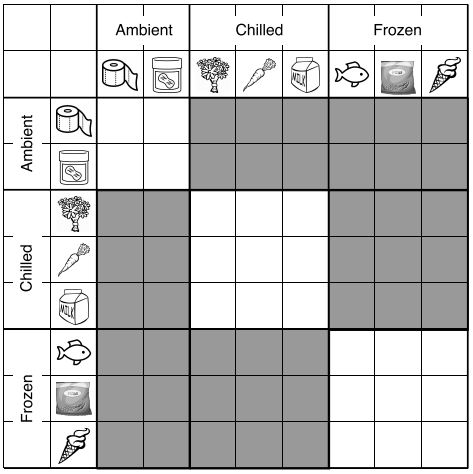
5.2.3 Tree traversal

In the following experiments we test the speed and functioning of the tree traversal algorithm. We do this partially by comparing the new results with the original algorithms used at the two customers. As described earlier, we had to make some adjustments to the retailer 1 dataset to make comparisons possible, due to the working of the ILP model. We also run the experiments on the retailer 2 dataset in a different environment, than all experiments so far. We are unable to create adequate test runs for these retailer 2 experiments on the laptop, due to a mismatch in software versions of the vehicle routing algorithm. Therefore, it is only possible to run these tests on a virtual machine. Not only was there a difference in the type of calls to the load validation algorithm, we also encounter a higher variability of the runtime between identical runs on the virtual machine. However, by taking the average of three subsequent runs, we try to reduce this variability. We consider it to only to be necessary to run it three times for the experiments on the virtual machine, as the experiments on the local machine does not show the same high variability. We finish with testing how the tree traversal performs with an active contamination matrix.

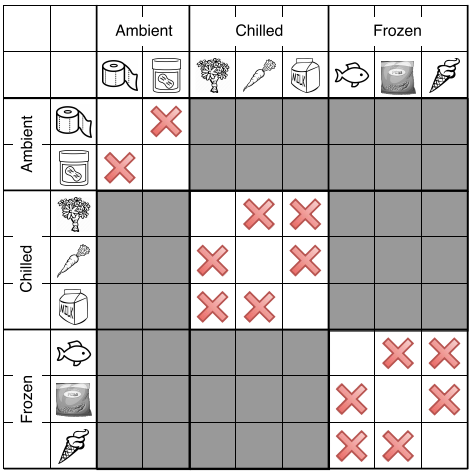
The sixth experiment tests the computation time of the original algorithms with that of the tree traversal algorithm. In this experiment, the pre-checks and caching functions are disabled to only test the tree traversal algorithm. The experiments consist of assigning all available orders into the available resource combinations. For the retailer 1 dataset, we test for both the fixed (dataset 2,3) and flexible (dataset 4,5) datasets. Note that we base our results on two measurements of two identical runs. In addition, we

compare the two compartment strategies, namely the front-first and the prediction based strategy. The minimum length of the route to activate the prediction based algorithm is set to 10 orders. We only test the prediction based strategy on the retailer 2 dataset.

The last experiment includes a fictional contamination matrix (experiment 7). Because it is not possible to include the contamination matrix into the ILP model, the test focusses only on the tree traversal algorithm. Until now, orders with the same product groups but with different product types are allowed in the same compartment (Figure 5.3a). The fictional contamination matrix adds contamination rules for all product types, meaning two orders with different product types always have to be in separate compartments. For example, two orders within the frozen product group, but with different product kinds, are not allowed in the same frozen compartment (Figure 5.3b). Because the current resource combinations have only one compartment of every product group, we double the capacity of the resource combinations. This means six compartments, two of each product group (same as compartment type), and twice the capacity of the resource combinations. The smaller resource combinations with only one compartment remain the same size.



(a) No contamination on product level, only on product group level.



(b) Every product type contaminates with each other.

FIGURE 5.3: Two contamination matrices.

5.3 Results

In this section we present the results of our experiments. Section 5.3.1 presents the results of the verification experiments. Section 5.3.2 presents the results of the experiments on pre-checks and caching functions. We finish with the results on the experiments of the tree traversal algorithm (Section 5.3.3).

5.3.1 Verification

In this section we present the results of the first three experiments. Table 5.4 shows the computation time and the longest generated route for dataset 1 one of retailer 1 and the retailer 2 dataset. The total computation time is the total of the complete run, including the vehicle routing algorithm. The minimal difference between the computation times of the different algorithms, means that the computation time

required for load validation in these particular cases is small compared to the total computation time. Later on, se show that there is a significant speed difference between the current and our load validation algorithm, but the difference is rather small compared to the total computation time of the complete optimization tool. We also see that the original dataset of retailer 1 requires a computation time of almost 4 hours. This is not a realistic value as it is too long for the customers to do on a daily basis. That the computation time is this high is not surprising, as the customer normally plan it separate parts, instead all at once. Also, the maximum length of a route only consists of two orders, which indicates that the restrictions are to constraining for the vehicle routing algorithm to produce routes of a realistic length. It is therefore sensible to remove the most restricting constraints that do not have an effect on load validation itself, as we described before. Looking at the retailer 2 dataset, the results are as expected. A total computation time of well under an hour is realistic, as well as routes up to 25 orders. We see no reason to change the dataset of retailer 2.

Dataset	Algorithm	Total computation time	Longest route
Dataset 1	TT	3.8 hr	2 orders
	ILP	4.0 hr	2 orders
Retailer 2 dataset	TT	0.8 hr	25 orders
	CA	0.8 hr	25 orders

Table 5.4: Results of experiment 1. TT = Tree Traversal; ILP = Integer Linear Programming; CA = Custom Algorithm.

Table 5.5 shows the results of the second experiment. In this experiment we show that the tree traversal algorithm calculates the same feasibility answer as the current algorithm. If there would have been a different response for even one of the calls, the number would have been different due to the functioning of the vehicle routing algorithm. In addition, we checked on a random basis if the calls were equal at the same point in time for both algorithms. This means that both calls consist of an identical resource combination and sequence. We only found identical calls. We conclude that given these datasets, the tree traversal algorithm gives the same responses as the original algorithms.

Dataset	Algorithm	Calls
Dataset 3	TT	289,597
	ILP	289,597
Dataset 5	TT	237,225
	ILP	237,225
Retailer 2 dataset	TT	253,030
	CA	253,030

Table 5.5: Results of experiment 2. TT = Tree Traversal; ILP = Integer Linear Programming; CA = Custom Algorithm.

In the third experiment we verify if the laptop we use to test our algorithm, computes stable results. Table 5.6 shows the results of three equal runs on the laptop. We do these three runs on both datasets and the corresponding algorithms. The standard deviation of the total computation time is relatively small. The correlation coefficients show that the individual responses are closely related to each other.

This means, if one response has a high computation time in the first run, it is likely that the computation time of responses in the other two runs are high as well.

Dataset	Algorithm	Computation time	Correlation coefficients
Dataset 3	TT	μ(25292 ms)σ(609 ms)	0.95,0.95,0.97
	ILP	μ(301090 ms)σ(22399 ms)	0.96,0.95,0.96
Retailer 2 dataset	TT	μ(63992 ms)σ(577 ms)	0.98,0.97,0.98
	CA	μ(43799 ms)σ(114 ms)	0.97,0.97,0.99

Table 5.6: Results of experiment 3. TT = Tree Traversal; ILP = Integer Linear Programming; CA = Custom Algorithm.

5.3.2 Caching functions & Pre-checks

This section presents the results of experiments 4 and 5. Tables 5.7 and 5.8 show the results of experiment 4. The difference in datasets represent the use of flexible compartments. The computation times were measured with a precision of 4 digits, but even then both caching functions measured sometimes a computation time of zero. The table shows the average response, the longest response, and the total computation time. In addition, we calculate the total computation time of the longest 1% of the response. We sort the responses on the duration of the total computation time and take the summation of the top 1%. This gives an indication of the skewness of the responses, so we know if the responses are constant. Note that the average response time of the individual processes is calculated based on whether or not it has been active. Therefore, the total average response time differs from the sum of the other four.

Both tables look similar, and show no major differences. We see that the pre-checks take a relative large amount of time, compared to the other processes. However, if we look at the total computation time, it is around 3 to 4 times faster than solely using the tree traversal algorithm.

	Default CF	Superset CF	Pre-Checks	Tree traversal	Total
Average response	0.00 ms	0.01	0.09 ms	0.2 ms	0.02 ms
Longest response	0.079 ms	0.37 ms	3.5 ms	1.1 ms	4.0 ms
Total comp. time of process	$.17 * 10^3$ ms	$2.9 * 10^3$ ms	$2.9 * 10^3$ ms	$.8 * 10^3$ ms	$6.8 * 10^3$ ms
Total comp. time of longest 1%	$0 * 10^3$ ms	$0.1 * 10^3$ ms	$0.3 * 10^3$ ms	$0 * 10^3$ ms	$2.4 * 10^3$ ms

Table 5.7: Results of experiment 4 on the dataset 3 of retailer 1. The computation time of the individual processes have been measured. In this experiment the resource combinations have fixed compartments.

	Default CF	Superset CF	Pre-Checks	Tree traversal	Total
Average response	0.01 ms	0.01	0.24 ms	0.19 ms	0.03 ms
Longest response	0.1 ms	0.52 ms	8.6 ms	5.0 ms	11.3 ms
Total comp. time of process	$1.6 * 10^3$ ms	$3.1 * 10^3$ ms	$16.9 * 10^3$ ms	$13.1 * 10^3$ ms	$34.7 * 10^3$ ms
Total comp. time of longest 1%	$0 * 10^3$ ms	$0.1 * 10^3$ ms	$2.2 * 10^3$ ms	$1.5 * 10^3$ ms	$7.1 * 10^3$ ms

TABLE 5.8: Results of experiment 4 on the dataset 5 of retailer 1. The computation time of the individual processes have been measured. In this experiment the resource combinations have flexible compartments.

Figure 5.4 presents the results of experiment 4, the distribution of different response possibilities. It shows that only 1.6 and 2.5% of the load validation calls are exactly checked by the load validation algorithm. The majority of sequences were stopped by the caching functions. Between 76.8 and 87% were duplicate sequences or sequences that were identifiable as infeasible. Between 11 and 20.6% were sequences that were valid before.

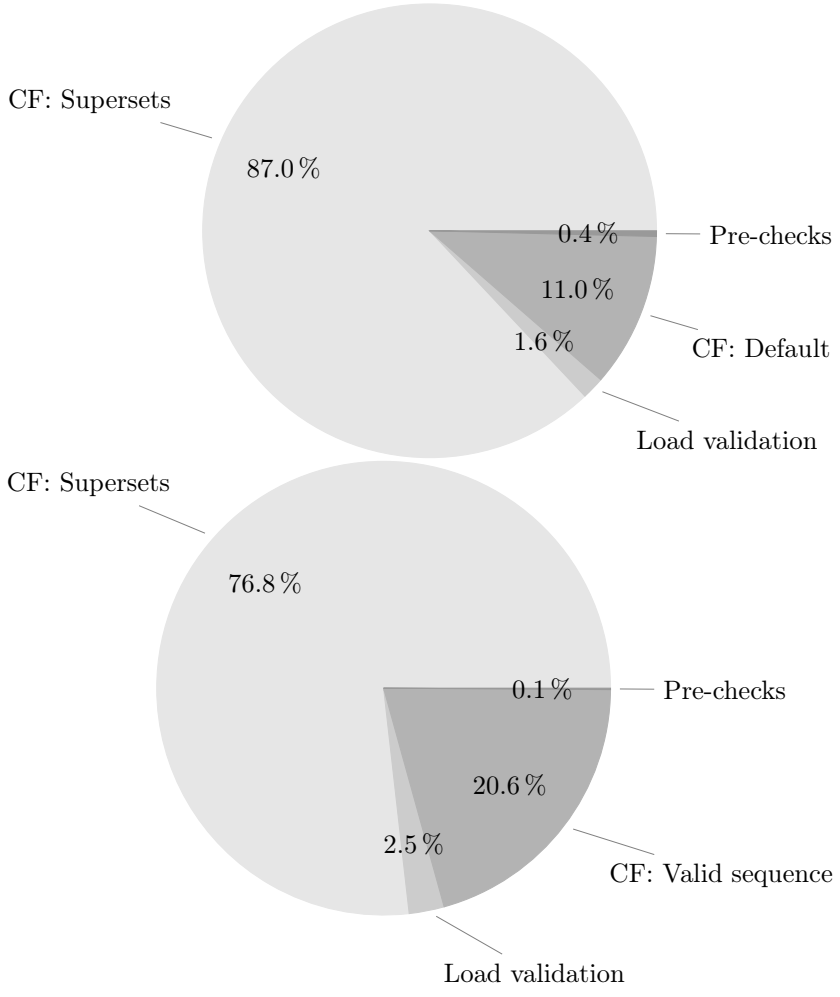


FIGURE 5.4: Results of experiment 4. It shows the percentage breakdown of the different functions. The top graph shows the results of the large retailer 1 dataset with fixed compartments (dataset 3), the bottom graph shows the results of the large retailer 1 dataset with flexible compartments (dataset 5). All Caching Functions (CF) cancel more then 97.5% of the calls to the load validation algorithm.

The same experiment is done on the retailer 2 dataset. We test the pre-checks and caching functions. The pre-checks have an average response time of 0.11 ms. Of the 238,000 calls to the pre-checks, none of the calls gave a negative response. The average response time of the default and superset caching function are 0.023 and 0.022 ms respectively. Note that, the average response time of the default caching function is based on only the active calls, because the superset caching function is located before the default caching function. Table 5.9 presents the results of the average and longest response, as well as the total computation time and the total computation time of the longest 1%.

Figure 5.5 presents the distribution of different response possibilities. It shows that only 1/3 of the load validation calls are exactly checked by the load validation algorithm. The other 2/3 are sequences that are blocked by the caching function. The major part (60.9%) are identical sequences that have been calculated before. The superset caching function retained 5.8% of the calls.

	Default CF	Superset CF	Pre-Checks	Tree traversal	Total
Average response	0.023 ms	0.022 ms	0.12 ms	0.11 ms	0.31 ms
Longest response	8.5 ms	8.5 ms	117 ms	79 ms	117 ms
Total comp. time of process	$7.1 * 10^3$ ms	$8.3 * 10^3$ ms	$31 * 10^3$ ms	$28 * 10^3$ ms	$74 * 10^3$ ms
Total comp. time of longest 1%	$.48 * 10^3$ ms	$.68 * 10^3$ ms	$6.2 * 10^3$ ms	$3.3 * 10^3$ ms	$8.2 * 10^3$ ms

Table 5.9: Results of experiment 4 on the retailer 2 dataset.

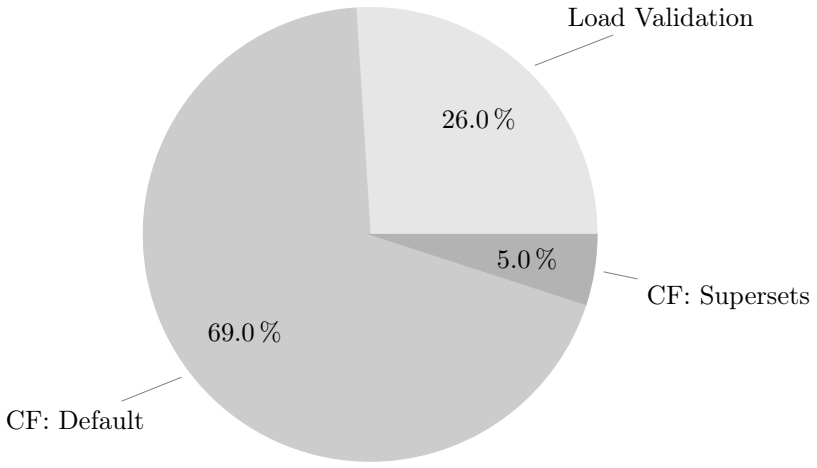


Figure 5.5: Percentage breakdown of the different functions. All Caching Functions (CF) cancel 2/3 of the calls to the load validation algorithm.

In experiment 5 we disable the caching functions, and only focus on the pre-checks. We do this because the pre-checks start after the caching function, meaning that we do not know for sure if the pre-check would also prevent the call from going to the tree traversal algorithm. We get different results from both algorithms. The retailer 2 dataset remains at 0 responses answered by the pre-checks. We conclude that the developed pre-checks operate on different constraints than those that are active for the retailer 2 dataset.

The retailer 1 datasets have a high percentage of responses, that end at the pre-checks. For dataset 3 we see that 82.9% of the calls get prevented by the pre-checks, and 76.8% when we look at dataset 5. This means the pre-checks are actually very helpful, but they get overshadowed by the power of the caching functions.

5.3.3 Tree traversal

In this section, we present the results of experiment 6 we ran on the retailer 1 datasets. Table 5.10 and 5.11 show the results of the comparison of the ILP algorithm with the tree traversal algorithm.

Table 5.10 presents the results of datasets 2 and 3. The algorithm received 41,344 and 289,597 calls in dataset 2 and 3 respectively. We see that the average response time of the tree traversal algorithm is for both experiments steady around 0.09 ms, significantly slower than the ILP model. We see that the longest 1% of response only cause around 9% of the total computation time and the longest response is under 3 ms. This makes the results of the tree traversal algorithm more constant than the ILP model. The total computation time of the tree traversal algorithm is around 35 times faster than the ILP model.

Table 5.11 presents the results of datasets 4 and 5. The algorithms received 46,828 and 237,225 calls in datasets 4 and 5 respectively. The average response time is similar in both experiments, while the total computation time is around four times as large for both the ILP and tree traversal. The average response time of the tree traversal algorithm is around 50 times faster than the ILP algorithm. We see that 26% of the total time in the ILP algorithm for the larger test set is caused by 1% of the slowest responses.

In addition, we measure the number of feasibility checks. The feasibility check is the part of the algorithm that checks, given an allocation of orders over the compartments, if it is feasible or not. For the tree traversal algorithm, this is at least the number of orders, and maximal the number of orders to the power of compartments. The computation power of the computer does not affect this indicator, making it a better indicator for comparison. However, we cannot use it on the ILP algorithm, as functions differently. The ratio between the smaller dataset (150 orders) and the large dataset (300 orders), is between 4 and 6 for both the feasibility check and the computation time. The results of the the experiment with fixed compartments look similar to that of the experiment with flexible compartments. The only difference is that the ILP runs faster on a dataset with fixed compartments than with flexible compartments. This can be explained by the fact that the algorithm only has to consider one configuration, instead of multiple in the dataset with multiple configurations. This means it has to construct a different ILP model for every configuration, which consumes more time.

	ILP	Tree traversal	ILP	Tree traversal
	dataset 2		*dataset 3*	
Average response	2.4 ms	0.09 ms	1.0 ms	0.09 ms
Longest response	216 ms	2.9 ms	219 ms	2.4 ms
Total comp. time load validation	$100 * 10^3$ ms	$3.8 * 10^3$ ms	$300 * 10^3$ ms	$26 * 10^3$ ms
Total comp. time of longest 1%	$14 * 10^3$ ms (14% of total)	$0.34 * 10^3$ ms (9% of total)	$78 * 10^3$ ms (26% of total)	$2.3 * 10^3$ ms (9% of total)
Total number of feasibility checks	n.a.	$128 * 10^6$	n.a.	$1103 * 10^3$

Table 5.10: Results of experiment 6. To compare only the tree traversal algorithm, no pre-checks and caching function are used. In this experiment, the algorithm uses the front-first compartment strategy, and the resource combinations have fixed compartment capacities.

	ILP	Tree traversal	ILP	Tree traversal
	dataset 4		*dataset 5*	
Average response	5.2 ms	0.10 ms	4.5 ms	0.09 ms
Longest response	271 ms	1.5 ms	454 ms	2.3 ms
Total comp. time load validation	$244 * 10^3$ ms	$4.4 * 10^3$ ms	$1058 * 10^3$ ms	$21 * 10^3$ ms
Total comp. time of longest 1%	$26 * 10^3$ ms (11% of total)	$0.4 * 10^3$ ms (9% of total)	$187 * 10^3$ ms (18% of total)	$1.9 * 10^3$ ms (9% of total)
Total number of feasibility checks	n.a.	$149 * 10^3$	n.a.	$718 * 10^3$

Table 5.11: Results of experiment 6. To compare only the tree traversal algorithm, no pre-checks and caching functions are used. In this experiment, the algorithm uses the front-first compartment strategy, and the resource combinations have flexible compartment capacities.

Figure 5.6 shows the average response time by the load validation algorithm, against the size of the route. We use both the ILP algorithm and the tree traversal algorithm, and run these algorithm on the large dataset with non flexible compartments. First, we see that all calls had routes with a maximum length of 8 orders. Second, we see that the graph of the ILP algorithm has a steeper slope compared to the tree traversal algorithm. This means that the computation time for longer routes, increases more using the ILP model than using the tree traversal algorithm. Therefore, an increased route length affects the computation time of the ILP algorithm much more than that of the tree traversal algorithm. Note that the scale of the y-axis is logarithmic.

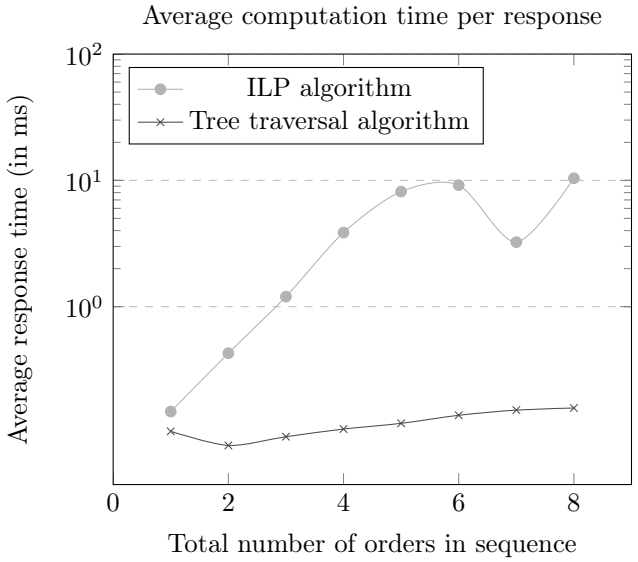


FIGURE 5.6: Results based on the retailer 1 dataset. The figure shows the average computation time per response for given lengths of the route.

In the following graphs and figures, we present the results of experiment 6 we run on the retailer 2 dataset. Table 5.12 presents the results of the first experiment, where we ran both algorithms without any pre-checks or caching function. One optimization run results in 238,000 calls. Both algorithms produce a similar list of responses. The current algorithm has an average computation time of 0.52 ms on a total computation time of 123 seconds. The tree traversal algorithm has an average computation time of 0.35 ms on a total computation time of 82 seconds.

The longest response is the computation time of the load validation call that took the longest to check. The longest response in the current algorithm was around 34 times higher than the longest response of the tree traversal algorithm. When we evaluate the longest 1% of the responses of both algorithms, we see that the top 1% of the current algorithm is responsible for 58% of the total computation time of the load validation part, while the 1% of the longest calls of the tree traversal algorithm only takes 6% of the computation time. Last, we compare the total number of load assignments validated, or feasibility checks. Although we cannot use this indicator on the ILP model, it is possible to use on the custom algorithm of retailer 2. The current algorithm validates a total of 223 million responses, while the tree traversal algorithm only validates 8.82 million load allocations. The custom algorithm of retailer 2 uses total enumeration, causing this large difference in the number of load allocations.

Figure 5.7 shows the average response time by the load validation algorithm, against the size of the route. We see that the tree traversal algorithm somewhat converges, which means a linear relationship exists between the length of the route and the computation time of the algorithm. The current algorithm shows the same relationship up to routes of 15 orders. For longer routes, the average computation time starts to increase rapidly.

In a separate run, we look at the two different compartment strategies. The results are only based on the 1/3 of responses, as the rest was terminated prematurely by the caching functions. The total amount of feasibility checks with the front-first strategy is 1.37 million against 1.4 million of the prediction-based strategy. This means the prediction-based strategy has a positive effect of 2.2 percent on the efficiency of the algorithm.

	Current algorithm	Tree traversal
Average response	0.52 ms	0.35 ms
Longest response	1290 ms	37.80 ms
Total computation time load validation	$123 * 10^3$ ms	$82.5 * 10^3$ ms
Total computation time of longest 1%	$71 * 10^3$ ms (58% of total)	$5.7 * 10^3$ ms (6% of total)
Total number of feasibility checks	$223 * 10^6$	$8.82 * 10^6$

TABLE 5.12: Results of experiment 6, ran on the retailer 2 dataset. To compare only the tree traversal algorithm, no pre-checks and caching function are used. In this experiment the algorithm uses the front-first compartment strategy.

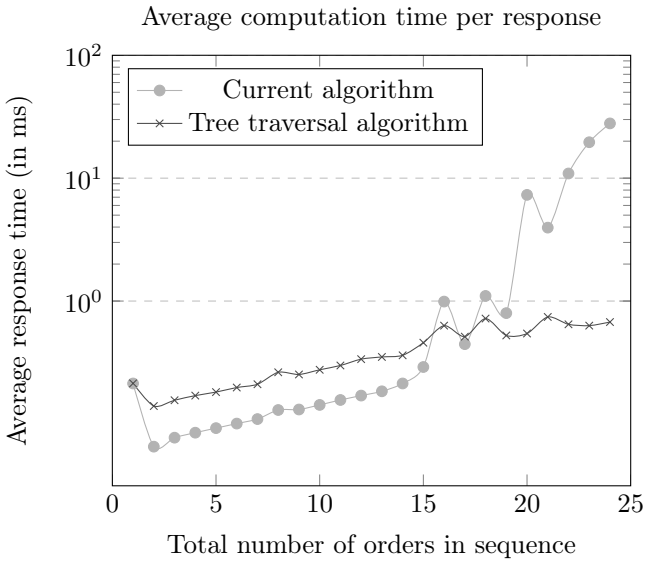


FIGURE 5.7: Results of experiment 6, ran on the retailer 2 dataset. The figure shows the average computation time per response for given lengths of the route. We see that it takes more time to validate longer routes.

Table 5.13 presents the results of the last experiment. A contamination matrix has been added to datasets 9 and 10. The number of responses by the load validation algorithm are 49,920 and 351,794 respectively. The average response time is almost equal for both the small and large set, while the total response time is around 7 times larger in the larger dataset. The total computation time of the longest 1% does not show a skewness.

	Dataset 6	Dataset 7
Average response	0.13 ms	0.12 ms
Longest response	6.4 ms	14.9 ms
Total computation time load validation	$6.6 * 10^3$ ms	$44.6 * 10^3$ ms
Total computation time of longest 1%	$.791 * 10^3$ ms (12% of total)	$5.2 * 10^3$ ms (12% of total)

TABLE 5.13: Results of experiment 7. This experiment include a fictional contamination matrix in which every product type contaminates with every other product type of its own product group. The small set consist of 150 orders and 13 resources, while the large set consist of 300 orders and 25 resources.

Chapter 6

Conclusions and Recommendations

In Section 1.6, we stated the four research questions together with the goal of this research. In this chapter we discuss the answers to these questions and whether or not we reached our goal. So far, Chapters 2 to 5 each answered one of the research questions. In Chapter 2, we presented what is currently known in the literature in relation to load allocation. In Chapter 3, we discussed what the requirements of the retail case are. We designed our load validation algorithm in Chapter 4. Lastly, we analysed its performance in Chapter 5. Section 6.1 presents the conclusions based on the results presented in Chapter 5. In Section 6.2 we present our recommendations on what we think should be done with the findings of this research.

6.1 Conclusions

We developed a load validation algorithm that determines in a split second if it is feasible to load a certain sequence of goods inside a resource combination. In this section we discuss the conclusions of this research based on findings described in the previous chapters.

First, we defined the requirements of a typical retail customer of ORTEC. Based on two major customers of ORTEC, we created a list of 23 requirements on which the algorithm should be functional. These customers can be seen as typical examples of a retail distributor. We looked at future trends, but we did not encounter any potential developments that would influence the requirements. We updated the load classification model of Bortfeldt and Wäscher [3] and sorted the list of requirements using this model. We classified 2 requirements as "container-related", while the other 21 requirements were related to the "positioning" of the order. The difference is that for the two container-related requirements, the internal sequence of orders affects the feasibility of the sequence, whereas for the 21 positioning requirements only the characteristics of the orders affect the feasibility. This means that the sequence of the pick-ups and the deliveries influences load validation. We use this in our advantage, by creating the tree traversal algorithm.

Second, we developed a load validation algorithm that is suitable to quickly validate if the load allocation constraints are met. The algorithm shows to be faster than the current practices. Besides being faster, it is important to have a reliable response time. We see that the tree traversal algorithm does not have the outliers as the current algorithms have. The average response time of a call using the tree traversal

algorithm was on all datasets of retailer 1 around 0.09 ms, and on the dataset of retailer 2 around 0.35 ms. Both the longest response, as well as the total computation time of the longest 1%, are reasonable when compared to the average response and the total response time. Also, when the complexity increases due to longer routes, we see that the tree traversal algorithm has approximately a linear relationship, while the other algorithms has exponentially increasing computation times with increasing route length.

The pre-checks and cache functions gave mixed results. The pre-checks did not gave the desired results as we hoped to have in the retailer 2 dataset, it never prevented any call. On the other hand, within the retailer 1 dataset we reached up to a percentage of 82.9% for the dataset with fixed compartments. However, this percentage was only 0.4% if the caching functions were active. The computation time of the pre-checks itself was not extremely high, but compared to the cache functions and the load validation algorithm it is too high. The performance of the cache functions were beyond expectations. The amount of responses that the cache function can prevent is extremely high, reaching between the 75% and 97.5% of the calls. The total time increase of adding the caching function was between 1/4 and 1/2 of the tree traversal time.

Third, we wanted to know how well the tree traversal algorithm performs under different circumstances. We differentiated between flexible and fixed compartments, and the inclusion of a contamination matrix. The two available datasets forced us to test different sets of requirements on both datasets. The dataset of retailer 1 did not support co-loading, weight restrictions, and two resources in a resource combination (truck and trailer). The dataset of retailer 2 did not contain product types, a contamination matrix, compartments, and baffles. So, we were unable to test all 23 requirements at once on only one of the datasets. Still, we see similar results on all of them. More than 3/4 of the calls are prevented using the caching functions, and the tree traversal algorithm remains approximately linear when increasing the complexity (length) of the routes. For all datasets, the average response time is always around a few milliseconds, while the sum of all response times is always under one hundred seconds.

6.2 Recommendations

This section presents our recommendations based on the results of this research. First, we recommend to implement the caching functions into the current software of ORTEC. The caching functions show to be effective. Because the caching functions are based on the principle of not calculating something twice, it is easy to implement, even if load validation is done with different algorithms than the tree traversal algorithm. It does not matter for the default caching function what load validation algorithm ORD uses. For the superset caching function it is a bit more complicated, as we need to identify for all current restrictions if they can be added to the default or superset caching function. When using the tree traversal algorithm, we can keep track which restrictions were triggered and if a failed sequence can be stored in the superset caching function. This is the preferred method, as the superset caching function considers all subsets of a response if it has failed before, making it more effective. However, the ILP cannot keep track which of the restrictions caused load validation to fail, as it does not operate like a search method. This means further research is required to properly implement the superset caching function. Second, we recommend to first do additional research before implementing the pre-checks into ORD. The results vary, between not effective and very effective. We think it is key to determine in which cases a pre-check is useful and when it is not, so they are only active when necessary.

Third, we recommend to further research the use of the tree traversal algorithm, into the load validation algorithm (COPSLoad) of ORTEC Routing and Dispatch (ORD). The results show large improvements related to the total computation time, as well as more constant response times. The algorithms in this research have been programmed into the current software environment of ORTEC and have been tested on actual client datasets. The experiments themselves were tested in the same way as customers would use the software. This makes it valid to conclude that the positive results will actually lead to improvements for ORTEC's retail customers. However, it is not directly clear if the tree traversal algorithm will be a success for all customers. We know that ORTEC has a large set of customers hauling liquid substances, something not necessary effective to validate using tree traversal. Also, the measurements on how the tree traversal algorithm performed under different requirements sets should in our opinion be expanded, to get a better insight on which restrictions influence the algorithm in what way.

For further research, we recommend to find even better datasets in which all restrictions can be active. Also, we were very dependent on the functioning of the vehicle routing algorithm for the quality of our calls. A better understanding on how to tune the vehicle routing algorithm is recommended. In our introduction, we already present the idea of a better cooperation between the vehicle routing algorithm and the loading algorithm. We decided to keep it out of our scope, but we recommend for further research to include it. Additionally, we see many opportunities to improve the caching functions. If we can make even better deductions if a sequence is going to be infeasible, this would greatly increase the speed. An idea would be to keep track of the depth of the tree during load validation. If we know that the sequence always fails before adding the 5th node in a sequence of 25, we know that the first 5 nodes are the critical part. Also, we took every unique resource combination as an unique root node, but we can group them based on their characteristics. This would mean we do not need 266 different root nodes, but only the number of different resource combinations. Finally, the caching functions should be moved earlier into the whole validation function. Now it is placed at the beginning of the load validation part, but we could move it as one of the first checks to perform, so also before all other validations like time. If we know that a sequence is going to fail on the load restrictions, we should reject this solution as fast as possible.

Bibliography

[1] Birger Funke, Tore Grünert, and Stefan Irnich. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of heuristics*, 11(4):267–306, 2005.

[2] Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.

[3] Andreas Bortfeldt and Gerhard Wäscher. Constraints in container loading–a state-of-the-art review. *European Journal of Operational Research*, 229(1):1–20, 2013.

[4] Nitin Chaturvedi, Mirko Martich, Brian Ruwadi, and Nursen Ulker. The future of retail supply chains. *Operations as a competitive advantage in retail*, pages 59–67, 2013.

[5] Maria Battarra, Michele Monaci, and Daniele Vigo. An adaptive guidance approach for the heuristic solution of a minimum multiple trip vehicle routing problem. *Computers & Operations Research*, 36 (11):3041–3050, 2009.

[6] Koen Demkes. Automated tuning of an algorithm for the vehicle routing problem. 2014.

[7] Thijs van Dijk. Tuning the parameters of a loading algorithm. 2014.

[8] Hanne Pollaris, Kris Braekers, An Caris, Gerrit K Janssens, and Sabine Limbourg. Vehicle routing problems with loading constraints: state-of-the-art and future directions. *OR Spectrum*, pages 1–34, 2014.

[9] Harald Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

[10] Eberhard E Bischoff and MSW Ratcliff. Issues in the development of approaches to container loading. *Omega*, 23(4):377–390, 1995.

[11] Yi Tao and Fan Wang. An effective tabu search approach with improved loading algorithms for the 3l-cvrp. *Computers & Operations Research*, 55:127–140, 2015.

[12] Søren Gram Christensen and David Magid Rousøe. Container loading with multi-drop constraints. *International Transactions in Operational Research*, 16(6):727–743, 2009.

[13] David Pisinger. Heuristics for the container loading problem. *European journal of operational research*, 141(2):382–392, 2002.

[14] Yong Wu, Wenkai Li, Mark Goh, and Robert de Souza. Three-dimensional bin packing problem with variable bin height. *European Journal of Operational Research*, 202(2):347–355, 2010.

[15] Leonardo Junqueira, Reinaldo Morabito, and Denise Sato Yamashita. Mip-based approaches for the container loading problem with multi-drop constraints. *Annals of Operations Research*, pages 1–25, 2012.

[16] John A. George and David F. Robinson. A heuristic for packing boxes into a container. *Computers and Operations Research*, 7(3):147–156, 1980. doi: 10.1016/0305-0548(80)90001-5. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-0019208595&partnerID=40&md5=3003a2e5f7b0df002381ec04c92a96b5`. cited By 141.

[17] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing*, 20(3):368–384, 2008.

[18] Hermann Gehring and A Bortfeldt. A genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 4(5-6):401–418, 1997.

[19] Xueping Li, Zhaoxia Zhao, and Kaike Zhang. A genetic algorithm for the three-dimensional bin packing problem with heterogeneous bins. In *proceedings of the 2014 Industrial and Systems Engineering Research Conference*.

[20] Andrea Lodi, Silvano Martello, and Daniele Vigo. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, 141(2):410–420, 2002.

[21] Andrew Lim, Hong Ma, Chaoyang Qiu, and Wenbin Zhu. The single container loading problem with axle weight constraints. *International Journal of Production Economics*, 144(1):358–369, 2013.

[22] Sara Ceschia and Andrea Schaerf. Local search for a multi-drop multi-container loading problem. *Journal of Heuristics*, 19(2):275–294, 2013.

[23] Fabiano Do Prado Marques and Marcos Nereu Arenales. The constrained compartmentalised knapsack problem. *Computers & operations research*, 34(7):2109–2129, 2007.

[24] Aline AS Leão, Maristela O Santos, Robinson Hoto, and Marcos N Arenales. The constrained compartmentalized knapsack problem: mathematical models and solution methods. *European Journal of Operational Research*, 212(3):455–463, 2011.

[25] Manuel Iori and Silvano Martello. Routing problems with loading constraints. *Top*, 18(1):4–27, 2010.

[26] Hanne Pollaris, Kris Braekers, An Caris, Gerrit K Janssens, and Sabine Limbourg. Capacitated vehicle routing problem with sequence-based pallet loading and axle weight constraints. *EURO Journal on Transportation and Logistics*, pages 1–25, 2014.

[27] Rahma Lahyani, Leandro C Coelho, Mahdi Khemakhem, Gilbert Laporte, and Frédéric Semet. A multi-compartment vehicle routing problem arising in the collection of olive oil in tunisia. *Omega*, 51:1–10, 2015.

[28] Manuel Iori, Juan-José Salazar-González, and Daniele Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 41(2):253–264, 2007.

[29] Emmanouil E Zachariadis, Christos D Tarantilis, and Chris T Kiranoudis. Integrated distribution and loading planning via a compact metaheuristic algorithm. *European Journal of Operational Research*, 228(1):56–71, 2013.

[30] Tino Henke, Maria Grazia Speranza, and Gerhard Wäscher. *The Multi-Compartment Vehicle Routing Problem with Flexible Compartment Sizes*. Univ., Fak. für Wirtschaftswiss., 2014.

[31] Lijun Wei, Zhenzhen Zhang, Defu Zhang, and Andrew Lim. A variable neighborhood search for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 243(3):798 – 814, 2015. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2014.12.048. URL `http://www.sciencedirect.com/science/article/pii/S0377221714010662`.

[32] Stephen CH Leung, Xiyue Zhou, Defu Zhang, and Jiemin Zheng. Extended guided tabu search and a new packing algorithm for the two-dimensional loading vehicle routing problem. *Computers & Operations Research*, 38(1):205–215, 2011.

[33] Michel Gendreau, Manuel Iori, Gilbert Laporte, and Silvaro Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, 51(1):4–18, 2008.

[34] Christos D Tarantilis, Emmanouil E Zachariadis, and Chris T Kiranoudis. A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. *Intelligent Transportation Systems, IEEE Transactions on*, 10(2):255–271, 2009.

[35] Qian Wang, Qingkai Ji, and Chun-Hung Chiu. Optimal routing for heterogeneous fixed fleets of multicompartment vehicles. *Mathematical Problems in Engineering*, 2014, 2014.

[36] Abdellah El Fallahi, Christian Prins, and Roberto Wolfler Calvo. A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem. *Computers & Operations Research*, 35 (5):1725–1741, 2008.

[37] Qingfang Ruan, Zhengqian Zhang, Lixin Miao, and Haitao Shen. A hybrid approach for the vehicle routing problem with three-dimensional loading constraints. *Computers & Operations Research*, 40 (6):1579–1589, 2013.

[38] Jonathan Reynolds and Malin Sundström. Digitalisation, retail transformation and change: what will european consumers want from their future shopping centre experience? 2014.

[39] Gian Luca Marzocchi and Alessandra Zammit. Self-scanning technologies in retail: Determinants of adoption. *The Service Industries Journal*, 26(6):651–669, 2006.

[40] Helen Rogers, Talat Abd El Hakam, Evi Hartmann, and Marina Gebhard. Rfid in retail supply chains: Current developments and future potential. In *Logistics Management*, pages 201–212. Springer, 2015.

[41] Gary M Gaukler, Ralf W Seifert, and Warren H Hausman. Item-level rfid in the retail supply chain. *Production and Operations Management*, 16(1):65–76, 2007.

[42] John Fernie and Leigh Sparks. 1 retail logistics: changes and challenges. *Logistics and Retail Management: Emerging Issues and New Challenges in the Retail Supply Chain*, page 1, 2014.

[43] Jorgen Christensen. Moving freight with better trucks. *International Transport Forum*, 2011.

[44] Guenther Fuellerer, Karl F Doerner, Richard F Hartl, and Manuel Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers & Operations Research*, 36(3): 655–673, 2009.

[45] Emmanouil E Zachariadis, Christos D Tarantilis, and Christos T Kiranoudis. A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 195(3):729–743, 2009.