Container loading with multi-drop constraints

Søren Gram Christensen

David Magid Rousøe

Kongens Lyngby 2007
IMM-MSC-2007-37
April 10, 2007

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

In this thesis, we develop an algorithm for the container loading problem (CLP) with multi-drop constraints. The CLP is the problem of loading the maximal value of a set of boxes into a container, while ensuring that the boxes neither overlap each other nor the container walls. By adding multi-drop constraints it is further demanded, that the relevant boxes must be available, without rearranging others, when each drop-off point is reached.

The algorithm is made with a setup in mind, where the CLP will be solved as a feasibility check on a route suggested by a vehicle routing application. Therefore solutions must be obtained fast. Furthermore it is demanded, that packings must be feasible in a real world scenario. The latter is accomplished by a detailed model, which makes sure that all boxes are placed multi-drop feasible, properly supported and only rotated in feasible directions. It is also assured, that boxes that are stacked, are not damaged. Based on the model, a tree search framework has been developed. The tree search utilises different greedy methods to evaluate the potential of each node considered. To make the framework more generic, a dynamic breadth is proposed. Based on problem characteristics and the time limit imposed, it will choose the breadth of the tree, making sure that the time is utilised most profitable.

The algorithm is tested on data from the two Danish companies Aarstiderne and Johannes Fog. The obtained results show, that the proposed model and algorithm is able to solve these problems within a reasonable time, giving solutions that respect all the constraints imposed and are feasible in a real world scenario.

Keywords: Container Loading, Vehicle Routing, Multi-drop, Heuristics, Tree-

search

Resumé

I denne afhandling udvikles en algoritme til container loading problemet med multi-drop begrænsninger. Container loading problemet består i, at pakke den maksimale værdi af et sæt af bokse ind i en container. Boksene må hverken overlappe hinanden eller container væggene. Ved at tilføje multi-drop begrænsninger kræves det yderligere, at de relevante kasser er tilgængelige, uden at omrokere andre kasser, når de skal tages ud.

Algoritmen er lavet med henblik på, at skulle kunne benyttes til at kontrollere, om ruter foreslået af en ruteplanlægningsapplikation er gyldige. Dette kræver, at løsningerne bliver fundet hurtigt. Det er et yderligere krav, at pakningerne er virkelighedstro. Dette er opnået ved hjælp af en detaljeret model, der sikrer, at alle kasser er placeret så de overholder multi-drop begrænsninger, er tilstrækkeligt understøttede og kun roteres i tilladte retninger. Det er yderligere sikret, at bokse kun placeres ovenpå hinanden, hvis de kan holde til det. Baseret på denne model er der udviklet en træsøgningsalgoritme. Denne udnytter forskellige grådige metoder til at evaluere potentialet i de betragtede knuder. For at lave træsøgningen mere generel anvendelig, foreslås en dynamisk bredde, der baseret på problem karakteristika og den angivne tidsgrænse, vælger bredden, så tiden bliver udnyttet bedst muligt.

Algoritmen er testet på data fra to de danske selskaber Aarstiderne og Johannes Fog. De opnåede resultater viser, at den foreslåede model og algoritme kan løse problemerne indenfor en rimelig tidsgrænse. De fundne pakninger respekterer alle begrænsningerne og er virkelighedstro.

Nøgleord: Pakning af lastrum, Container loading, Ruteplanlægning, Multi-drop, Heuristikker, Træ-søgning

Preface

This Master Thesis was conducted at the Institute of Informatics and Mathematical Modelling (IMM), Technical University of Denmark (DTU), in co-operation with Transvision A/S in the period from the 4th of September 2006 to the 10th of April 2007. The thesis is written to achieve the degree Master of Science in Engineering at DTU.

Supervisor from IMM, DTU on this project was Associate Professor Jesper Larsen. From Transvision A/S Jakob Birkedal Nielsen supervised the project.

We would like to thank our supervisors for their help throughout the project. Also a special thank to Sune Vang-Pedersen from Transvision A/S for great inspiration.

Kgs. Lyngby, April 2007

Søren Gram Christensen s011566

David Magid Rousøe s011302

Contents

Chapter 1

Introduction

The mathematical field of *Operations Research* (OR) supplies a number of tools that are used to model, solve and improve solutions to many problems faced by companies today. These problems are often found in areas such as logistics and planning.

One such problem is packing of goods. Packing problems are met in many forms. In a smaller scale it could be arranging boxes on a pallet, on a larger scale it could be loading a fleet of trucks delivering consumer goods at supermarkets. The packing problems can be categorised into several different OR problems. One of these is the *container loading problem* (CLP). In general terms, the problem can be formulated as follows: Given a set of small boxes and a container, load the boxes into the container, maximising the total value of the load. Often the value of a box equals its volume, thus the problem is to maximise the volume utilisation of the container.

Another classical OR problem is the *vehicle routing problem* (VRP). It is the problem of visiting a number of customers in the least expensive way, when a fleet of vehicles is given. A possible extension to this problem is that the vehicles are distributing some kind of goods to the customers and the vehicles have limited space to store these goods. Then the problem becomes a *capacitated vehicle routing problem* (CVRP). In CVRP the capacity is calculated as a one dimensional measure. This is a reasonable assumption if, for instance, the problem is

to distribute oil to households. Then the volume of the tank on the truck is a natural capacity limit on how many litres of oil the truck can hold. However, the one dimensional capacity measure is often to simple. If the problem is to bring out construction products, there is no natural one dimensional measure on how many products a vehicle can hold. This is because the products can be very different in size and shape. In that case a three dimensional capacity measure is needed.

To solve a vehicle routing problem containing a three dimensional capacity measure, it is necessary to be able to solve both VRP and CLP. The combined problem is known as the *three-dimensional loading capacitated vehicle routing problem* (3L-CVRP). In this problem, a box loaded in the container must be dropped of at a certain customer on the route. In a practical setting, it is reasonable to demand, that when a customer is reached, the boxes that should be dropped off must be accessible. This adds multi-drop constraints to the CLP.

Both the VRP and the CLP have had the focus of researchers for many years, but the combined problem has only recently been investigated. It is a natural development that more complex problems can be considered today, as both computer power and the solution methods have improved drastically during the past years. 3L-CVRP is an example of this.

A possible solution procedure for 3L-CVRP, is to find a solution to the VRP and check for each of the vehicles assigned to a route, if all boxes can be loaded. This reduces the CLP to a feasibility check for the VRP solution. This means that the two problems can be solved independently, which is illustrated in Figure 1.1.

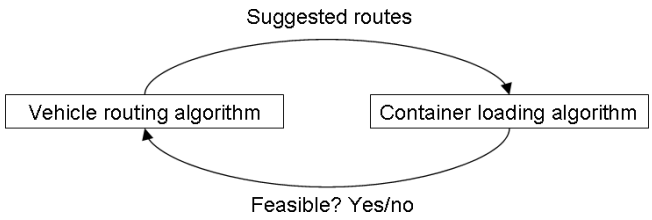


Figure 1.1: The interface between a vehicle routing and a container loading algorithm

For this setup to work in a practical application, short response time from the CLP is important. The reason for this is that algorithms solving the VRP typically generate and evaluate a large number of routes. If the feasibility check of a route cannot be done quickly, it is not possible, in a practical setting, to use the algorithm.

The Danish company Transvision A/S is a leading provider of transportation management solutions and their core product is a state-of-the-art vehicle routing application. This product is used by a range of customers, mainly situated in Northern Europe. Transvision wishes to improve their application, so that it can handle 3L-CVRP. To do this an algorithm that is able to solve the CLP is needed.

1.1 Purpose

The purpose of this work is to develop an algorithm to solve the three dimensional rectangular container loading problem with multi-drop constraints. Packings found by the algorithm must be feasible in a real world scenario. To reveal the complications met in real world scenarios industrial cases must be included. The algorithm must be usable in connection with a vehicle routing application and hence, must be able to give solutions fast.

1.2 Thesis structure

The layout of this thesis is as follows. First, we will describe the container loading problem that we will solve and place it in the family of packing problems. In the subsequent chapter, we will go through the most important literature. We will especially focus on the part of the literature which deals with heuristic solution approaches. We will also mention some problem extensions which have been introduced by others.

In Chapter 4 a mixed integer programming model, describing the basic CLP, is formulated. This is primarily done to introduce the problem, but also to illustrate the complexity of the problem. Then, in Chapter 5, we will introduce a more realistic container loading model. This is done by imposing additional constraints on the problem, such as the multi-drop constraint. In Chapter 6 we will further extend the model, so that is becomes capable of handling problems faced by two very different companies that solve the CLP with multi-drop constraints in their everyday work.

In Chapter 7 the different solution approaches we have developed are explained. This is partly a number of greedy methods and partly a tree search improvement framework. Chapter 8 shortly explains the most fundamental points from the implementation, and Chapter 9 introduces the data we have used to tune and test the proposed algorithms. Afterwards, in Chapter 10, the methods are

tuned to perform best possible. This includes discarding some of the greedy methods and afterwards finding the best settings for the introduced tree search framework.

Finally, in Chapter 11, we will report the results obtained with our algorithm. The test of the algorithm consists of two stages. First, benchmark problems from the literature are solved and secondly, the performance on specific industry problems is reported. In Chapter 12, we will discuss our results and point out possible improvements and further research topics in this field. Chapter 13 conclude on our findings.

Chapter 2

Problem description

In this chapter we will place the container loading problem in the family of cutting and packing problems, and further specify the problem which is the subject of this work.

In 1990 Dyckhoff wrote "A typology for cutting and packing problems" [9]. This article was published in The European Journal of Operations Research (EJOR) and recently received the honour of being chosen, by the editors of EJOR, among the 30 most influential papers published in the first 30 years of the journals history. Based on Dyckhoff's typology, Wäscher et al. [35] developed a new improved typology for cutting and packing problems. These typologies form the basis for the following description of packing problems.

It can be realised that there exists a close connection between basic cutting and packing problems. Packing *small items* into *large objects* can be juxtaposed with cutting the small items out of large objects. An example of small items and large objects are boxes and containers, respectively. When the problems are further specialised, the connection is less obvious. In this description, we will focus on packing problems.

2.1 General packing problems

Packing problems can be split in two basically different problem types: maximisation and minimisation. In the maximisation problem, the large object is limited and a maximal number of small items should be packed inside the large object. The minimisation problem, on the other hand, is the problem of packing a limited number of small items into a minimal number of large objects.[1] Two classical examples of one dimensional packing problems are the *knapsack problem* [27] and the *bin packing problem* [29]. These are maximisation and minimisation problems, respectively.

Several criteria are used to classify the different packing instances. These are:

Dimensionality The distinction is basically between 1, 2 or 3 dimensions, but also problems with more dimensions have been considered in the literature.

Assortment of large objects If more than one large objects is used, it is of interest whether the large objects are identical or not.

Assortment of small items There are three cases. Problems where all the small items are of same size are denoted homogeneous. Problems, with many small items of few different sizes are denoted weakly heterogeneous. Lastly, problems with many different small items of many sizes are denoted strongly heterogeneous.

Shape of small items If problems have more than one dimension, it is an important problem characteristic, what shapes the small items have. Basically they can be either regular or not. Regular shapes are rectangles, boxes, circles, balls etc. while irregular (or non-regular) items are for example the shape of a t-shirt, sofa etc. Among regular shapes a further distinction is made between rectangular items and circular items.

Different extensions can be added to the problem to make it more realistic or specialised. A condensed list, taken from Bischoff and Ratcliff [3], is shown below:

Orientation constraints Only specific rotations of the boxes are allowed. This is often indicated by a "this side up" mark on boxes.

[1] A special case of the minimisation problems is the *strip packing problem*, where all small items should be packed into one variable sized large object, minimising the size of the large object.

Fragile items Some items are fragile and hence, no items should be placed on top them.

Load bearing constraints Each box can carry only a limited weight.

Load stability The packing should be stable, so that when the container is handled, the boxes do not move. Therefore, it can be important that boxes are supported from below and from the sides.

Multi-drop The boxes are to be dropped off at different places and at each place the unloading should be possible without rearranging the boxes left in the container.

Grouping of items The same type of boxes should be grouped, to ease loading and unloading of the container.

Weight limit The weight limit on the container should not be exceeded.

Weight distribution Restrictions on the weight distribution in the container are imposed.

2.2 Container loading in a vehicle routing context

When solving 3L-CVRP, a number of customers are assigned to each vehicle. For each vehicles all boxes belonging to the customers should be loaded. This means that each instance of the packing problem consist of only one large object, but many small items, belonging to a number of customers. The large object considered is a three dimensional container, and the small items three dimensional boxes. This packing problem is also known as the *single container loading problem* (SCLP), and it is a maximisation problem.

In the typology by Wäscher et al. [35] SCLP is classified as a three-dimensional rectangular *SLOPP* and -*SKP*, whereas Dyckhoff [9] classify it *3/B/O/F*, *3/B/-O/M* and *3/B/O/R*. In Wäschers typology both SLOPP (Single Large Object Placement Problem) and SKP (Single Knapsack Problem) identifies problems with a single large object. SLOPP contains a weakly heterogeneous consignment, whereas SKP contains a strongly heterogeneous consignment. From Dyckhoffs typology 3/B/O, describes problems with a three dimensionsional (3), maximisation (B) problems (german: beladen) where one large object is considered (O). The fourth character in the classification describes the assortment of the small items. These can be either few (F), many of many different sizes (M) or many of relatively few different sizes (R).

For a solution to the SCLP to be practical applicable it should also satisfy a number of the problem extensions described in Section 2.1. First of all, it is important to obey the multi-drop constraint, as this is a natural extension when SCLP is solved in combination with VRP, because more customers are considered. In a real world scenario, SCLP is not realistic, because boxes are allowed to float in the air, hence no guarantee can be given about the stability of the boxes. Therefore extensions regarding the relation between the boxes should be included. These are stability-, load bearing strength- and rotation restriction constraints. The weight limit restriction should also be considered, but, as we are solving the CLP in a vehicle routing context, all boxes must be packed. Therefore it is more obvious to check the weight limit constraints in the VRP application.

It has been a goal throughout the project to make an algorithm which is able to solve problems faced by real world companies. Therefore, the constraints on the model have been chosen with this in mind. The primary considerations about multi-drop constraints were introduced by Transvision, who also requested constraints regarding the stability of the load and assurance of fragile boxes. The model obtained with these constraints is denoted a *realistic* model. We have further extended the problem to meet the needs of the two Danish companies Aarstiderne and Johannes Fog, who distribute organic vegetables and fruit, and construction products, respectively. These models are both denoted *specialised* models.

For a load to be multi-drop feasible, we demand that boxes should be available when they are needed. This means that no boxes which are to be dropped off later should be placed on top or in front of the boxes we need to drop off at a given time. In the realistic model, we will allow unloading from one direction only. This restriction is relaxed in one of the specialised models, so that three unloading directions are available.

The stability of a load is closely related to how much support the boxes get. We will only work with support from below. In the realistic model we will assume that all boxes must be fully supported from below. In one of the specialised models, this will be relaxed, so that the minimum amount of support can be chosen freely.

Load bearing strength constraints assure that boxes do not get damaged. It can be seen as an extended fragility principle, but gives a more realistic measure on which boxes can be stacked. This is only part of the specialised models. In the realistic model, however, we will assume that boxes can be either fragile or not.

Finally, the boxes can only be rotated in some given ways. Most often the restriction is on which side of a box should be up. This is for instance the case

if open boxes with vegetables are distributed. This constraint is implemented in the realistic model.

Chapter 3

Literature review

The combined problem of CLP and VRP has only lately been a subject of academic research in a notable degree. Because of the problem complexity, real world applications have only been documented in few cases. However, separately each of the problems have been studied thoroughly for decades.

Before attempting to solve the combined problem, the nature of both problems must be investigated thoroughly. The focus of this work is to solve CLP with multi-drop constraints, that afterwards, can be combined with a suitable solver for the VRP. Therefore, we have concentrated this review on literature concerning the pure container loading problem as well as the combined container loading and vehicle routing problem, leaving out the massive study of the pure vehicle routing problem.

The first section of this chapter presents two typologies of cutting and packing problems. The next section refers to articles where mixed integer programming (MIP) models are used to represent the CLP. In Section 3.3 known solution procedures for the problem are discussed and in Section 3.4 articles describing solution procedures to the combined packing and routing problem are discussed.

3.1 General packing and cutting problems

In 1990 Dyckhoff wrote "A typology of cutting and packing problems" [9]. The paper identifies the logical structure that is common for the many different cutting and packing problems and develops a standard notation for describing the different characteristics in these. The article gives an overview of the research done in the field until that date, and is an excellent introduction to both cutting and packing problems.

Since the time Dyckhoff published his typology, the research on the field has intensified, and in the light of this development an updated version of the typology has been written. This was done by Wäscher et al. [35]. In the typology it is stated that more than 400 articles have been published from 1995 to December 2005 dealing with cutting and packing problems "in the narrow sense". This excludes all papers describing problem extensions, such as weight limit, weight distribution, multi-drop, stability, etc. In the improved typology a more fine grained classification scheme is proposed. This supports a more precise classification of both new and old literature. References to all the literature classified in the article has been gathered in a database, which is available on the internet [11].

Both these typologies were used as a basis to classify and describe our problem in Chapter 2.

3.2 MIP models for the container loading problem

The CLP is known to be an NP-hard combinatorial optimisation problem [28]. Hence, it cannot be guaranteed to be solved to optimality in polynomial time. This is probably the reason that only few contributions exist on MIP models for the problem.

In 1993, Chen et al. [6] formulated the single- and multi-container loading problems as MIP models and solved them to optimality for small problems. This was done as a proof of concept to make sure that the models actually give feasible solutions to the problem. Validation of the models is done on problems with only 6 boxes. To solve these problems, solution times of 15 minutes are reported. They extent the model, so that it can solve the strip packing problem and problems where the weight distribution is of importance. Their final conclusion was that: "A more efficient solution procedure is needed

to solve large scale container loading problems."

In 1999, Scheithauer [31] presents another model of the problem, which is based on one dimensional bars. The bars represent stacks and are found as solutions to knapsack problems. The model is decomposed into a subproblem, that generates the one dimensional bars and a master problem that chooses, which combination of bars that gives the best packing. The master problem is a two dimensional packing problem. The model is solved by column generation, where the bars are generated as solutions to knapsack problems. They use the model to find bounds to both the single and multi-container loading problems based on LP-relaxations. The bound for the single container loading problem is found within 36 seconds on average, when 5 different box types are considered.

Both models only consider the basic CLP. The only constraints imposed, is that boxes should not overlap with each other or the container walls. This is seen as a major drawback of the models, as the solutions found are not valid in a realistic setting. Moreover, there is no real perspective for our purposes, as solutions to these models cannot be found fast enough. The value of the papers is that, they show that MIP models cannot be used effectively to solve the CLP. Furthermore a formal description of the most basic CLP can be derived.

3.3 Heuristics approaches for the container loading problem

As earlier mentioned, the CLP is NP-hard. For this reason, when solution procedures are developed, heuristics are often used. In the following, the most important of the different methods will be mentioned.

A general solution strategy is to divide the container into smaller pieces, packing each of these separately. The point often being to reduce the solution space from three dimensions to one or two. Pisinger [28] gives an excellent overview of these strategies.

The most common dividing procedure is *wall* building, introduced by George and Robinson [16]. A wall is constructed by making a vertical slice through the container. The depth of a slice is defined by the depth of the first box placed in the wall. As the slice is filled, the boxes will create a wall-like formation. This procedure is used in [4, 7, 24, 28]. Methods where the container is split by horizontal slices have also been developed. This is often called *layers*. Compared to the wall approaches, layer approaches are said to be produce more stable loads. Examples can be seen in [3] and [20]. In *stack* building approaches (see

[13, 17]) box stacks are constructed, leaving only a two dimensional problem of arranging the stack on the container floor. Other methods put together boxes in *blocks* before they are placed in the container. A block typically consists of one or two types of boxes, and the advantage is therefore that each block can be tightly packed. This procedure is used by [5] and [10].

A common term used, both in connection with wall and layer building algorithms, but also alone, is guillotine cuts. The term is taken from cutting theory, and it describes a cut going straight through an object. The cut will continue until another guillotine cut is met or the object is cut through. When cutting glass, it is often demanded that the cutting patterns are guillotine cuttable. In relation to container loading, guillotine cuts can be used to split the container into smaller pieces, which is utilised by Morabito and Arenales [23].

Typically, greedy construction procedures are used to make fast solutions to the overall packing problem or sub problems. On top of this, many different heuristic frameworks are used. In the recent literature metaheuristics such as tabu search [34], genetic algorithms [34] and GRASP [12] have been used. Straight forward construction heuristics and tree search heuristics are also common.

3.3.1 Construction algorithms

In 1965 Gilmore and Gomory [17] presented a paper in which they solve two- and three dimensional cutting stock problems. The cutting stock problem consist of cutting demanded material out of bigger raw-material. In the two dimensional case this is often paper, glass, steel etc. In the three dimensional case it could be small stones cut out of a large one. They realise that cutting and packing problems are very closely related, and point out that the proposed heuristic can be use to solve the single container loading problem. The algorithm basically consists of building all possible stacks by solving knapsack problems, and afterwards solving a two dimensional packing problem of arranging these stacks on the container floor. They do not solve any specific container loading problems, but mention the first known, to the knowledge of the authors, solution procedure.

In 1980 George and Robinson [16] developed the first attempt to pack a single container with a given set of boxes. They use a wall building approach, where one wall is build at a time, and the depth of the wall is defined by the first box placed in the wall. They have two different rules for choosing the first box in a wall. The first one chooses the box type with the most boxes left while the second chooses the box with the largest smallest dimension. The first rule tends to build blocks of the same box types, while the second rule is based on the

intuition, that large boxes may be hard to place later on in the packing process. Rectangular empty cuboids, denoted empty spaces, are used to indicate where in the container it is possible to place boxes. The use of empty spaces is now common when modelling the problem. The authors solve a real life problem with around 800 different boxes and up to 20 different box types. They solve 15 problems in all. All the solutions were found *by hand*, which - as the authors mention - was a time consuming process taking many hours.

In 1994, Ngoi et al. [25] presented a somewhat different representation of the problem. The container is modelled as many matrices, where the first row and column of each matrix determines the coordinates on the container floor and the rest of the elements in a matrix identify different boxes. Each matrix correspond to different layers in the container. Their heuristic uses a ranking scheme when selecting which boxes to place. The scheme favours placements where the most dimensions between the box and the empty space are equal. All combinations of boxes and empty spaces are considered. The algorithm is tested on 15 problems, and volume utilisations around 80% were found within 1 minute.

In 1995 Bischoff and Ratcliff [3] describe two different heuristics to solve the CLP. Each heuristic cater for different problem extensions. First, they consider the stability of the load to be of importance. This is accomplished by building layers from the container floor and upwards, which assures that the highest placed box is placed relatively low. Thereby the overall stability of the load is increased.
Secondly, they cater for multi-drop situations. This is accomplished by packing the boxes in the opposite sequence of the visiting order. When the container is packed, the boxes are chosen according to a ranking scheme, where box types which give the best utilisation in a vertical stack in the chosen empty space is favoured. Both heuristics are tested on a number of problems. These problems do not have any information about different drop-off points, so the test of the algorithm, which is designed for multi-drop situations, is in reality done on problems with one customer only.
Bischoff and Ratcliff remark that common test data is necessary, if different solution approaches should be compared. Therefore, 700 problems are generated, that are still used as benchmark problems today. These problems can be found in the OR-Library [1]. Both algorithms yield volume utilisations around 82%. Overall the heuristic made for multi-drop situations turns out to be the better of the two proposed algorithm. However, the two algorithms seem to compliment each other. With concern to the multi-drop constraint the results are irrelevant, as only one customer is considered.

In Davies and Bischoff's paper from 1999 [7], an algorithm that cater for weight distribution and stability is described. The algorithm build segments consisting of one or more walls. Inside a segment, the face of the previous wall is used

as starting point for the next one, instead of using a straight wall. Between the segments, guillotine cuts are used. The motivation for building segments instead of walls is that it gives a more stable load, than the straight forward wall building approach. When the packing density is increased, the boxes are more tightly packed, hence, giving more support to each other making the load more stable. The drawback is that the segments make it harder to create an even weight distribution. To make a better weight distribution the segments are rotated and/or interchanged. To identify empty spaces in the container, the representation suggested by Ngoi et al. [25] is used. When the problems mentioned in [3] are solved, volume utilisations between 82,4% and 85% are obtained. Davies and Bischoff also solve problems with up to 100 different box types. When these problems are solved the volume utilisation drop to around 73%. The time spent before finding their final solutions varies between 1 and 48 seconds.

In 2004 Bischoff [2] proposed an algorithm to solve problems where the load bearing strength of the boxes is limited. To model the empty space in the container, Bischoff propose a new matrix representation of the container. Instead of keeping one matrix for each layer, as proposed by Ngoi et al. [25], only a single matrix is used. In this matrix the first column and row represent different coordinates on the container floor. The remaining elements describe how high the boxes have been stacked in that given area. A similar matrix is used to model the load bearing strength. Five different criteria are introduced favouring different box/space combinations. In each iteration of the search procedure, a new relative ranking of these criteria are used. Using 1000 iterations the average volume utilisation for the OR-Library problems is above 90% when load bearing strength is omitted. These results are obtained in between 26,4 and 321 seconds on average. When load bearing strength is introduced the volume utilisation decreases.

Common for the solution approaches mentioned so far, is that they all, except for the latter, make one greedy solution to the problem, and do not try to improve it. Many different problem extensions have, however, already been introduced. This includes multi-drop constraints, stability measures and load bearing strength. Two distinct ways of modelling the spaces in the container and placement of the boxes have been introduced. The concept of empty spaces, introduced by George and Robinson [16], and the matrix representation by Ngoi et al. [25], which was slightly modified by Bischoff [2], are used for these purposes. In the following sections we will go through different approaches, which use some kind of improvement method to solve the CLP. This is most often done in combination with some greedy methods, which all, to some extent, looks like the construction algorithms introduced.

We will start by describing different tree search algorithms. Afterwards different

metaheuristics will be mentioned.

3.3.2 Tree search algorithms

Scheithauer [30] developed a dynamic program based on the "forward state strategy" to solve the container loading problem. If *all* box/space combinations (states) are visited, when trying to place a box (stage), an optimal solution to the problem can be found. But due to the size of the problem, only a limited number of states in every stage of the dynamic program are considered. He reports having tested with 1, 2 and 5 states in every stage. The tendency is (not surprising) that 5 states gives better results than 1 and 2. But it takes "some" minutes instead of "a few" seconds to find a solution. Problems with up to 100 boxes are considered.

In Morabito and Arenales (1994) [23] an approach to the container loading problem using guillotine cuts is proposed. They generate a tree by guillotine-cutting the container into smaller parts eventually leading to boxes. To make a heuristic solution they prune their tree. This is done by making crude upper and lower bounds, based on how many of a single box type can fit into the piece of the container, evaluated in the current node. They also give the framework of an exact solution approach to the problem, where the container should be guillotine cuttable, that is, an optimal solution for the specific problem, but it is not necessary optimal for the original container loading problem. They are able to solve the problems from the OR-Library in less than 200 seconds giving volume utilisations just around 90%.

Michael Eley's approach from 2002 [10] uses block arrangements to solve the container loading problem. The advantage of the blocks is that boxes of same type can be stored closely, loading time is reduced and stability is improved. The solution method applied is a greedy heuristic with a tree search improvement strategy. In every node of the search tree a single box is placed. The placement is evaluated by the corresponding greedy solution, and only a limited number of nodes are kept in every depth of the tree. The method is both applied to multi- and single container loading problem. For the single container loading problem volume utilisations between 88% and 89,2% were found with a time limit of 10 minutes. This was done on the test problems from the OR-Library.

An algorithm for the single container loading problem based on knapsack packing has been proposed by Pisinger in 2002 [28]. The algorithm is based on the wall building strategy, where each wall is composed of vertical and horizontal stacks, which are found solving knapsack problems. A tree search framework is applied to decide the depth of the walls. To decrease the problem complexity,

only the most promising nodes are kept in each depth in the tree. The wall depths and the stack widths are ranked by considering the dimensions of boxes not yet packed. Many different ranking strategies are tested. Pisinger reports volume utilisations around 90% for problems which are comparable to the ones found in OR-Library.

3.3.3 Genetic algorithms

In Gehring and Bortfeldt's genetic algorithm from 1997 [13], many box stacks are produced leaving the authors with a two dimensional packing problem of arranging the stacks on the container floor. The two dimensional packing problem is solved by a genetic algorithm. The solutions are represented by the packing sequence (chromosomes) of the box stacks, which are mutated and crossed with each other to get better solutions. Mutation and crossover operations consist of changing the sequence of the stacks and interchange stacks, respectively. In each iteration new box stacks are made by the set of not loaded boxes in that chromosome. Testing on the problems from OR-Library the authors get average volume utilisations between 85% and 88%. This is done in 3 minutes on average.

The same authors proposed a new genetic algorithm in 2001 [4]. Here they use a wall building approach to fill the container. After having generated a population of solutions they *cross* and *mutate* solutions to get new solutions. The mutation and crossover methods choose walls from the parent(s) which are combined into new solutions. When no more walls can be added without having the same box in the solution twice, the remaining of the container is loaded greedily, by the basic wall building algorithm. This approach achieves between 87,8% and 90,7% in volume utilisation for the OR-Library problems. 316 seconds are used on average.

3.3.4 Tabu search algorithm

In 2002 Bortfeldt et al. [5] propose a parallel tabu search to solve the CLP. The method uses a measure of support, allowing either full or partial support of the boxes from below. A greedy procedure is used to load a container and a tabu search to improve the load. Two different neighbourhood definitions are used. A big one considering all neighbours, where one "arrangement" is different from the current solution, and a small one where only the most promising part of the neighbourhood is considered. An arrangement consist of a number of boxes producing a homogeneous block. Four differently configured tabu searches are run in parallel, exchanging best solutions. Numerical result show that the sequen-

tial algorithm is effective, but parallelization does not improve solution quality significantly. The results obtained on the OR-Library problems report average volume utilisations of 91,6% and 92,2% for the sequential and parallel versions, respectively. This was achieved with a time limit of 10 minutes. The average solution time for the sequential algorithm however was around 38 seconds.

3.3.5 GRASP algorithms

Moura and Oliviera (2005) [24] propose a GRASP algorithm for the CLP. They do a straight forward wall building approach, where they, among a given percentage of the most promising boxes, choose a random one to place next. After the container is packed with all the boxes, they perform a neighbourhood search by taking out all boxes placed later than a given box i and greedily pack the rest, without including the box type of box i before at least one other box is placed. The volume utilisations achieved are on average between 88% and 90,1% for the problems from the OR-Library. This was found in 33 seconds on average.

Another GRASP approach, developed by Parreño et al. [26], was presented at the 4th ESICUP meeting 2007. The new approach uses spaces, which have the largest possible expansion. Thereby no guarantee of support is given. When a space has been selected, the authors chooses to place the block consisting of boxes of the same type, which utilise the space the best way. This method achieves an average volume utilisation of 94% in around 30 seconds.

3.3.6 Hybrid algorithms

In 2004 Mack, Bortfeldt and Gehring [21] further developed their parallel tabu search ([5]) to also include a simulated annealing. In the hybrid version of their method, they use the simulated annealing to jump around in the solution space and the tabu search to search for better solutions in the neighbourhood found by the simulated annealing. Both sequential simulated annealing, sequential tabu search, hybrid sequential search and parallel dittos are tested. By combining all these methods and take the best found solution, the authors are able to report some really good volume utilisations. For the 700 OR-Library problems an average utilisation of 93,78% are achieved in 596 seconds on average, using 64 PC's.

Based on priority rules, Lim and Zhang (2005) [20] propose an algorithm, which repeatedly produce loads. They build layers from the container floor and upwards. To build a layer, they choose the lowest unoccupied point inside the

container and identify the extension of the related surface. They fill this surface with only one box type using the G4 heuristic proposed by Scheithauer and Terno [32] to solve two dimensional pallet loading problems. After a solution is found, it is analysed and boxes not packed are given higher priority. Then a new solution is found based on the new priorities. Average volume utilisations between 91% and 92,4% are found in 707 seconds on average for the problem in the OR-Library. However, the time usage spans between 1 and 4600 seconds.

The overview of the heuristics shows, that throughout the years many different solution approaches have been tried. The trend is that metaheuristics are replacing simple construction heuristics. Tree search approaches are, however, applied with comparable results. To the basic container loading problem a single greedy solution can give volume utilisations between 80% and 85%. If an improvement method is used, this measure can be enhanced to between 90% and 94%.

3.4 Combined routing and packing

Now, we will turn the focus to approaches which have been applied to the combined VRP and CLP problem. Within this field out focus is how the two problems are interacting, how the introduction of a more complex loading constraint will affect the VRP results, and of course which loading algorithms are considered.

In 2006 Iori et al. [19] proposed an exact algorithm for the two-dimensional loading capacitated vehicle routing problem (denoted 2L-CVRP). They are fully aware of the need to pack in three dimensions, but argue that the problem typically is to load pallets that cannot be stacked, leaving only a two-dimensional packing problem. Their exact approach is based on a branch-and-cut framework that minimises the routing cost and calls a branch-and-bound method to check for feasibility in the corresponding packings. They consider 60 problems with up to 35 customers and 114 items, but are not able to guarantee optimality in five cases within the given time limit (1 CPU-day).

Gendreau et al. (2006) [15] developed a tabu search for 2L-CVRP. They propose two different packing approaches, one considering the basic two dimensional problem and one that additionally cater for multi-drops. Moreover the possible rotations of the pallets are restricted, so that only one rotation is feasible for each pallet. This is argued as a reasonable restriction as the pallets are handled with a forklift, which does not allow rotation inside the container. When constructing a solution, a box is inserted in the space where the largest surface of the box

abut another box or the container. To obtain different packings, different box sequences are tried. Before a box is inserted, it is tested whether or not the loading constraints are violated. On the tested problems it is shown that the solution quality drops on average 53%, when the multi-drop constraints are introduced. The extra solution time spent only amounts to 4% on average. Problems with between 16 and 252 customers are considered in the test. Each customer receives between 1 and 5 pallets.

Doerner et al. (2006) [8] recently solved a real-world capacitated vehicle routing problem with multi-drop constraints using metaheuristics. Because of the simple nature of their packing problem, it is reduced from a three dimensional to a one dimensional problem. No weight or stability constraints are imposed. They solve the vehicle routing problem by either tabu search or ant colony optimisation. The packing problem is solved by a dynamic program or greedy heuristic. As the authors reduce the packing constraints the paper is not really interesting in our situation.

In 2005 Gendreau et al. [14] developed a tabu search heuristic to solve the three dimensional loading capacitated vehicle routing problem (3L-CVRP). They demand that loads obey stability-, fragility-, fixed vertical orientation and multi-drop constraints.
Overall they use two distinct tabu searches, one solving the VRP and one solving the CLP, with the problem extensions mentioned. The latter tabu search is solving a strip packing problem, with a virtual container of infinite length, but otherwise the same dimensions as the real container. The problem is to find a packing with a length smaller than or equal to the real container length. The neighbourhood is defined as the interchange between boxes which are loaded inside the real container and the boxes which are not. Two different greedy procedures are introduced to make the actual packings. The first one chooses the lowest valid point in the container to pack a given box, whereas the second one try all insertions which are valid with respect to the problem extensions and chooses the best found insertion evaluated by the amount of the surface of the box that touches either another box or the container.
To evaluate their approach, VRP instances from VRPLIB [33] are used. Each customer in the VRP instances is assigned a set of boxes. Tests show that adding the extra constraints to the CLP extend the solution time needed to find their best solutions from 1500 to 2000 seconds on average. Unfortunately, it is not reported how the packing constraints affect the solution quality.

3.5 Summary

The literature review reveals that many different approaches to the container loading problem with many different problem extensions have been tried. However, not much focus is given to three dimensional packing with multi-drop constraints. Two articles mention solution procedures to solve the problem. The first one is proposed in [3], but was not tested on problems where multi-drop situations occur. The second approach is mentioned in [14], where the combined routing and packing problem is solved. The shared idea is to load the boxes in the opposite sequence as they should be dropped off (LIFO). The latter approach furthermore introduces the idea to check the feasibility of an insertion, before a box is placed. This was also done in [15] for a two dimensional packing problem.

Another rarely mentioned extension, is the load bearing strength constraint. This was introduced in [2], where a matrix representation of the container is used to model the load bearing strength. Most of the mentioned approaches, assure that the boxes are, 1) only rotated in feasible directions and 2) fully supported from below, which are also restrictions that we imply on our problem.

When the 3L-CVRP (or reductions of it) are solved, the two problems are solved separately. Thereby the same setup is used as proposed in Chapter 1. Only sparse documentation concerning solution quality is available, but the introduction of loading constraints on the vehicle routing problem, seems to make it harder to solve.

The problem, which is treated in this thesis, is highly restricted compared to most problem considered in the literature. To be able to handle this, a high degree of control of the different box placements is needed. This kind of control is given in the tree search frameworks presented by Scheithauer [30] and Eley [10], where every box/space combination is a choice in the tree. When explicitly choosing each box placement, feasibility checks, like the ones used by Gendreau et al. [14], can be carried out, assuring that the different additional constraints we impose on our problem, are not violated. This is not straight forward in the tree search approaches by Morabito and Arenales [23] or Pisinger [28]. The same applies for the genetic algorithms proposed by Bortfeldt and Gehring [4, 13]. Here especially the multi-drop constraints becomes hard to cater for. Their solutions are represented by either a vector of box stacks or a vector of walls, which are permutated, thereby many boxes already loaded together, are moved around between the different solutions, which in many cases would conflict with the multi-drop constraint. Both the tabu search approach by Bortfeldt et al. [5] and the GRASP approach by Moura and Oliveira [24], gives the opportunity to make feasibility checks before each box is inserted.

Chapter 4

The basic single container loading problem

In this chapter a mixed integer programming model, describing the basic single container loading problem, is introduced. The problem can be formulated as follows: Given a set of boxes and a container, find the subset of the boxes with maximum total volume, which can be placed in the container without overlapping each other.

The model is inspired by the work of Chen et al. [6], who consider the multi-container loading problem. Their model is extended to describe two other packing problems, namely strip packing and the problem of choosing the smallest among many containers, wherein all considered boxes must be loaded. The different models introduced by Chen et al. describe problems where the size or the amount of containers should be minimised. Our problem is of a slightly different nature. We want to load as many of the boxes as possible into a container - if possible all. In a feasible solution, with regards to the VRP, all boxes must be loaded into the container. This is an optimal solution to the SCLP. If all boxes cannot be loaded, valuable information can still be gathered from the solution. The volume utilisation will give information about how dense the container is packed and the volume of the boxes not loaded indicates how much the consignment should be reduced, to allow all boxes to fit in the container. As a result we want to solve a problem where the total volume of the packed

boxes are maximised. We have one fixed sized container and a limited number of boxes. A model describing this is introduced in the following.

4.1 Mathematical model

To be able to relate the placement of the different boxes to each other, we place the container in a cartesian co-ordinate system. The container is placed with the back-left-lower corner in the origin of the co-ordinate system. The length of the container is placed along the x-axis, the width along the y-axis and the height along the z-axis.

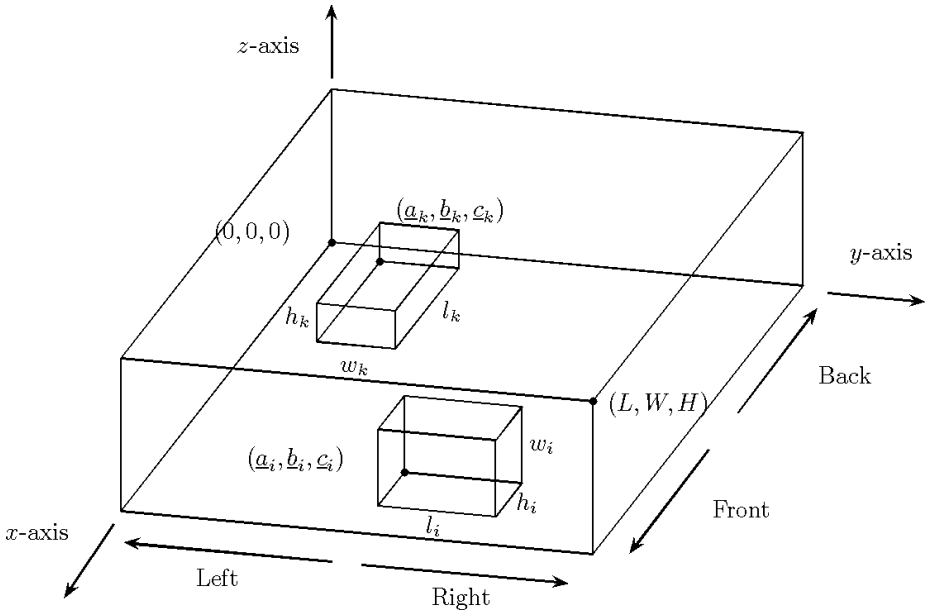
Given the set of boxes B, the list below gives the parameters and variables used in the model:

N	Total number of boxes $\|B\|$
l_i, w_i, h_i	Parameters indicating the length, width and height of box i
(L, W, H)	The container dimensions
(a_i, b_i, c_i)	Continuous variables. The coordinate describing the placement of box i.
$\ell_i^x, \ell_i^y, \ell_i^z$	Binary variables indicating whether the length of box i is parallel to the x- ($\ell_i^x = 1$), y- ($\ell_i^y = 1$) or z-axis ($\ell_i^z = 1$)
w_i^x, w_i^y, w_i^z	Binary variables indicating whether the width of box i is parallel to the x- ($w_i^x = 1$), y- ($w_i^y = 1$) or z-axis ($w_i^z = 1$)
h_i^x, h_i^y, h_i^z	Binary variables indicating whether the height of box i is parallel to the x- ($h_i^x = 1$), y- ($h_i^y = 1$) or z-axis ($h_i^z = 1$)
le_{ik}	Binary variable indicating if box i is placed on the left side of box k
ri_{ik}	Binary variable indicating if box i is placed on the right side of box k
be_{ik}	Binary variable indicating if box i is placed behind box k
fr_{ik}	Binary variable indicating if box i is placed in front of box k
ab_{ik}	Binary variable indicating if box i is placed above box k
un_{ik}	Binary variable indicating if box i is placed underneath box k
p_i	Binary variable indicating if box i is placed in the container
M	Large number used in Big-M constraints

The variables $le_{ik}, ri_{ik}, be_{ik}, fr_{ik}, ab_{ik}$ and un_{ik} are only defined for $i < k$, since that fully describes the placement relationship between box i and k. In Appendix A, a complete list of symbols used in this thesis can be found.

We have introduced 9 sets of binary variables to describe the rotations of the

boxes. This gives a over-representation of the variables necessary to describe the possible rotation. A reduced model is described in Section 4.1.1. For now we have chosen to include the variables to make the model easier to interpret.



$$
\begin{array}{rclrcl}
(\ell_i^x, \ell_i^y, \ell_i^z) & = & (0,1,0) & (\ell_k^x, \ell_k^y, \ell_k^z) & = & (1,0,0) \\
(w_i^x, w_i^y, w_i^z) & = & (0,0,1) & (w_k^x, w_k^y, w_k^z) & = & (0,1,0) \\
(h_i^x, h_i^y, h_i^z) & = & (1,0,0) & (h_k^x, h_k^y, h_k^z) & = & (0,0,1)
\end{array}
$$

$$le_{ik} = 0,\ ri_{ik} = 1,\ be_{ik} = 0,\ fr_{ik} = 1,\ ab_{ik} = 0,\ un_{ik} = 0,$$

Figure 4.1: Two boxes placed in a container, all variables concerning these boxes are shown.

In Figure 4.1 two boxes are placed in a container. It can be seen that the origin of the coordinate system coincide with one of the corners of the container, namely the back-left-lower corner. The container has dimensions L, W, H shown in the front-right-upper corner of the container. The two boxes i and k $(i < k)$ are placed in (a_i, b_i, c_i), respectively (a_k, b_k, c_k). Box i is placed with its length l_i along the y-axis, its width w_i along the z-axis and the height h_i along the x-axis. This is also indicated by the binary variables ℓ_i^y, w_i^z and h_i^x. It can be seen that box i is placed right of and in front of box k, which is indicated by the binary variables ri_{ik} and fr_{ik}.

The objective of the basic single container loading problem is to maximise the

volume of the boxes packed in the container. Now the basic single container loading problem can be formulated as:

$$\max \quad \sum_{i \in B} (l_i w_i h_i) p_i \tag{4.1}$$

Subject to:

$$a_i + l_i \cdot \ell_i^x + w_i \cdot w_i^x + h_i \cdot h_i^x \leq a_k + (1 - be_{ik}) \cdot M \quad \forall i,k,i<k \tag{4.2}$$

$$a_k + l_k \cdot \ell_k^x + w_k \cdot w_k^x + h_k \cdot h_k^x \leq a_i + (1 - fr_{ik}) \cdot M \quad \forall i,k,i<k \tag{4.3}$$

$$b_i + l_i \cdot \ell_i^y + w_i \cdot w_i^y + h_i \cdot h_i^y \leq b_k + (1 - le_{ik}) \cdot M \quad \forall i,k,i<k \tag{4.4}$$

$$b_k + l_k \cdot \ell_k^y + w_k \cdot w_k^y + h_k \cdot h_k^y \leq b_i + (1 - ri_{ik}) \cdot M \quad \forall i,k,i<k \tag{4.5}$$

$$c_i + l_i \cdot \ell_i^z + w_i \cdot w_i^z + h_i \cdot h_i^z \leq c_k + (1 - un_{ik}) \cdot M \quad \forall i,k,i<k \tag{4.6}$$

$$c_k + l_k \cdot \ell_k^z + w_k \cdot w_k^z + h_k \cdot h_k^z \leq c_i + (1 - ab_{ik}) \cdot M \quad \forall i,k,i<k \tag{4.7}$$

$$le_{ik} + ri_{ik} + be_{ik} + fr_{ik} + ab_{ik} + un_{ik} \geq p_i + p_k - 1 \quad \forall i,k,i<k \tag{4.8}$$

$$a_i + l_i \cdot \ell_i^x + w_i \cdot w_i^x + h_i \cdot h_i^x \leq L \quad \forall i \tag{4.9}$$

$$b_i + l_i \cdot \ell_i^y + w_i \cdot w_i^y + h_i \cdot h_i^y \leq W \quad \forall i \tag{4.10}$$

$$c_i + l_i \cdot \ell_i^z + w_i \cdot w_i^z + h_i \cdot h_i^z \leq H \quad \forall i \tag{4.11}$$

$$\ell_i^x, \ell_i^y, \ell_i^z, w_i^x, w_i^y, w_i^z, h_i^x, h_i^y, h_i^z, p_i \in \mathbb{B} \quad \forall i \tag{4.12}$$

$$le_{ik}, ri_{ik}, be_{ik}, fr_{ik}, ab_{ik}, un_{ik} \in \mathbb{B} \quad \forall i,k,i<k \tag{4.13}$$

$$a_i, b_i, c_i \in \mathbb{R}_+ \quad \forall i \tag{4.14}$$

The objective function (4.1) maximises the volume of loaded boxes. Equations (4.2)-(4.7) assure that the loaded boxes do not overlap. Equation (4.2) for instance, assures that, if a box i is placed behind another box k, then the following must hold:

$$a_i + l_i \cdot \ell_i^x + w_i \cdot w_i^x + h_i \cdot h_i^x \leq a_k$$

where $l_i \cdot \ell_i^x + w_i \cdot w_i^x + h_i \cdot h_i^x$ is the size of the dimension of box i placed along the x-axis.

Equation (4.8) assures that all placed boxes are considered in the constraints (4.2)-(4.7). If both box i and k are placed in the container ($p_i + p_k = 2$), then

they are not allowed to overlap. This is assured by forcing at least one of the six variables $le_{ik}, ri_{ik}, be_{ik}, fr_{ik}, ab_{ik}, un_{ik}$ to one.

Equations (4.9)-(4.11), make sure that the boxes do not overlap with the container walls and finally constraints (4.12)-(4.14) describe each type of variables used.

4.1.1 Reducing the model

As described in Chen et al. [6] there are many redundant variables in the model. If the model should be solved this redundancy should be removed.

It is clear that a given box can be placed with its length along only one dimension, either x, y or z. Thereby:

$$\begin{aligned} \ell_i^x + \ell_i^y + \ell_i^z &= 1, \quad \forall i \\ w_i^x + w_i^y + w_i^z &= 1, \quad \forall i \\ h_i^x + h_i^y + h_i^z &= 1, \quad \forall i \end{aligned} \tag{4.15}$$

must hold. It is also clear that only one box side can be placed along the same dimension. Thereby also:

$$\begin{aligned} \ell_i^x + w_i^x + h_i^x &= 1, \quad \forall i \\ \ell_i^y + w_i^y + h_i^y &= 1, \quad \forall i \\ \ell_i^z + w_i^z + h_i^z &= 1, \quad \forall i \end{aligned} \tag{4.16}$$

must hold.

The relations in Equations (4.15) and (4.16), can be used to eliminate some of the variables used in the model. As the equation system described in (4.15) and (4.16) has rank five, one of the equations is fully described by the others. Therefore, one set of equations can be eliminated, leaving five sets of equations. One among many solutions is:

$$\begin{aligned} \ell_i^y &= 1 - \ell_i^x - \ell_i^z, \quad \forall i \\ h_i^z &= 1 - h_i^x - h_i^y, \quad \forall i \\ w_i^x &= 1 - h_i^x - \ell_i^x, \quad \forall i \\ w_i^y &= \ell_i^x + \ell_i^z - h_i^y, \quad \forall i \\ w_i^z &= h_i^x + h_i^y - \ell_i^z, \quad \forall i \end{aligned} \tag{4.17}$$

This can be used to reformulate constraints (4.2) to (4.7) and (4.9) to (4.11) as

follows:

$$\begin{aligned}
a_i + l_i \cdot \ell_i^x + w_i \cdot (1 - h_i^x - \ell_i^x) + h_i \cdot h_i^x &\leq a_k + (1 - be_{ik}) \cdot M \\
&\quad \forall i, k, i < k \quad (4.18)\\
a_k + l_k \cdot \ell_k^x + w_k \cdot (1 - h_k^x - \ell_k^x) + h_k \cdot h_k^x &\leq a_i + (1 - fr_{ik}) \cdot M \\
&\quad \forall i, k, i < k \quad (4.19)\\
b_i + l_i \cdot (1 - \ell_i^x - \ell_i^z) + w_i \cdot (\ell_i^x + \ell_i^z - h_i^y) + h_i \cdot h_i^y &\leq b_k + (1 - le_{ik}) \cdot M \\
&\quad \forall i, k, i < k \quad (4.20)\\
b_k + l_k \cdot (1 - \ell_k^x - \ell_k^z) + w_k \cdot (\ell_k^x + \ell_k^z - h_k^y) + h_k \cdot h_k^y &\leq b_i + (1 - ri_{ik}) \cdot M \\
&\quad \forall i, k, i < k \quad (4.21)\\
c_i + l_i \cdot \ell_i^z + w_i \cdot (h_i^x + h_i^y - \ell_i^z) + h_i \cdot (1 - h_i^x - h_i^y) &\leq c_k + (1 - un_{ik}) \cdot M \\
&\quad \forall i, k, i < k \quad (4.22)\\
c_k + l_k \cdot \ell_k^z + w_k \cdot (h_k^x + h_k^y - \ell_k^z) + h_k \cdot (1 - h_k^x - h_k^y) &\leq c_i + (1 - ab_{ik}) \cdot M \\
&\quad \forall i, k, i < k \quad (4.23)
\end{aligned}$$

$$\begin{aligned}
a_i + l_i \cdot \ell_i^x + w_i \cdot (1 - h_i^x - \ell_i^x) + h_i \cdot h_i^x &\leq L \quad \forall i \quad (4.24)\\
b_i + l_i \cdot (1 - \ell_i^y - \ell_i^z) + w_i \cdot (\ell_i^x + \ell_i^z - h_i^y) + h_i \cdot h_i^y &\leq W \quad \forall i \quad (4.25)\\
c_i + l_i \cdot \ell_i^z + w_i \cdot (h_i^x + h_i^y - \ell_i^z) + h_i \cdot (1 - h_i^x - h_i^y) &\leq H \quad \forall i \quad (4.26)
\end{aligned}$$

where equation (4.18)-(4.23) corresponds to (4.2)-(4.7) and equation (4.24)-(4.26) corresponds to (4.9)-(4.11). Finally equation (4.12) should be changed to:

$$\ell_i^x, \ell_i^z, h_i^x, h_i^y, p_i \in \mathbb{B} \qquad \forall i \tag{4.27}$$

since the number of variables is reduced.

By doing this reformulation the number of variables is reduced by $5N$.

The model as formulated in equations (4.1), (4.18)-(4.23), (4.8), (4.24)-(4.26), (4.27), (4.13) and (4.14) describes the basic single container loading problem. It has $7N\frac{N-1}{2} + 3N$ constraints, $6N\frac{N-1}{2} + 5N = (3N + 2)N$ binary variables and $3N$ continuous variables.

4.1.2 Deciding the value of M

When solving MIP models like the one described, it is always important to keep M as small as possible. If M is chosen arbitrarily large, the relaxations used by the MIP solver will be weak, leading to excessive branching and thus increased computation time.

To make the constraints (4.18)-(4.23) valid, M must be larger than the largest dimension of the box i which is considered in the constraint. This means that M becomes a set of values, one for each box. M_i can formally be defined as:

$$M_i = \max(l_i, w_i, h_i)$$

The index used on M should be the same as the index used on the left side of the inequalities in Equations (4.2)-(4.7) and (4.18)-(4.23). Equation (4.2) then becomes:

$$a_i + l_i \cdot \ell_i^x + w_i \cdot w_i^x + h_i \cdot h_i^x \leq a_k + (1 - be_{ik}) \cdot M_i \quad \forall i, k, i < k$$

whereas Equation (4.3) becomes:

$$a_k + l_k \cdot \ell_k^x + w_k \cdot w_k^x + h_k \cdot h_k^x \leq a_i + (1 - fr_{ik}) \cdot M_k \quad \forall i, k, i < k$$

4.2 Summary

In an industrial application it would not be unusual that 100 boxes should be packed. In that case the mathematical model has 34950 constraints, 300 continuous variables and 30200 binary variables. When Chen et al. introduced their models, which have the same characteristics as the model introduced in this chapter, they solved problems with 6 boxes, which took around 15 minutes. This clearly shows that using MIP models to solve our problem is not a possible approach.

Another problem with the model, is that many necessary constraints are neglected. In this basic model the only restriction is, that boxes do not overlap each other and do not protrude the container wall. This would allow the boxes to hover free in the air and furthermore not guarantee that fragile boxes are placed safely. These restrictions should at least apply to an industrial application, to assure that the content of the boxes is not damaged during transport. A further demand is that boxes can be restricted in their possible rotations. This is also not part of the model, but the constraint is straight forward to incorporate, as the only necessary change is to eliminates some of the binary variables. Moreover, in our specific setup, it is important also to include the multi-drop constraint, which are also not part of the introduced MIP model.

Chapter 5

A realistic single container loading problem

In the previous chapter a formal mixed integer programming model of the basic single container loading problem was derived. Because of the large amount of binary variables and constraints it proofed not to be a reasonable solution approach, considering our problem setup. Therefore no attempt was made to make the model more realistic.

In the light of this a heuristic solution approach is chosen. To accommodate this, a new mathematical model is introduced in this chapter. The modelling of the boxes and the container will be thoroughly described, this include properties regarding these entities. An important concept considered, is how to handle the space inside the container. Beside the restrictions contained in the basic model, some further extensions are considered. These will make the model more applicable in a real world scenario. The extensions are multi-drop- and stability constraints as well as limitations on the possible rotations of the boxes. First of all, however, we will introduce necessary general modelling notation.

5.1 General notation

A central part of the modelling takes advantage of, that the boxes and the container are placed in a cartesian coordinate system. As already mentioned all objects we consider will have a cuboid form, as they are rectangular boxes. A *cuboid* can be defined as follows:

Definition 5.1 (Cuboid) A *cuboid*, p, is a three dimensional object defined by the point closest to the origin of the co-ordinate system (the minimum):

$$\underline{p} = (\underline{p}_x, \underline{p}_y, \underline{p}_z)$$

and the point furthest away from the origin of the co-ordinate system (the maximum):

$$\overline{p} = (\overline{p}_x, \overline{p}_y, \overline{p}_z)$$

such that $\underline{p} < \overline{p} \Leftrightarrow \underline{p}_x < \overline{p}_x \wedge \underline{p}_y < \overline{p}_y \wedge \underline{p}_z < \overline{p}_z$ and $\underline{p}_x, \underline{p}_y, \underline{p}_z \geq 0$ and $\overline{p}_x, \overline{p}_y, \overline{p}_z > 0$.

The cuboid occupy all space, in the interval from its minimum to its maximum, in all three dimensions.

It is necessary to be able to relate and compare cuboids placed in the coordinate system. To do this, it is needed to compare points and intervals in both one, two and three dimensions. As these relations are used very often in the modelling, we have chosen to introduce some extra notation to ease the descriptions later on.

A very fundamental check is to see if two cuboids overlap.

Definition 5.2 (Overlap) Two cuboids p and q are said to *overlap* if and only if:

$$\begin{aligned} \underline{p}_x < \overline{q}_x \wedge\ \underline{q}_x < \overline{p}_x \quad \wedge \\ \underline{p}_y < \overline{q}_y \wedge\ \underline{q}_y < \overline{p}_y \quad \wedge \\ \underline{p}_z < \overline{q}_z \wedge\ \underline{q}_z < \overline{p}_z \end{aligned}$$

We will for short denote this $p \boxdot q$.

This means that cuboids are overlapping, if they overlap in the intervals between their minimum and maximum, in all three dimensions of the coordinate system. For each dimension we introduce the operators $\boxdot_x$, $\boxdot_y$ and $\boxdot_z$ which correspond

to overlap in the x-dimension, y-dimension and z-dimension, respectively. For instance, two cuboids p and q overlap in the x-dimension if:

$$p \boxdot_x q \Leftrightarrow \underline{p}_x < \overline{q}_x \wedge \underline{q}_x < \overline{p}_x$$

.

The following definition is also needed.

Definition 5.3 (Overlap-equal) Two cuboids p and q *overlap-equal* if:

$$\begin{aligned} \underline{p}_x \leq \overline{q}_x \wedge\ \underline{q}_x \leq \overline{p}_x \quad \wedge \\ \underline{p}_y \leq \overline{q}_y \wedge\ \underline{q}_y \leq \overline{p}_y \quad \wedge \\ \underline{p}_z \leq \overline{q}_z \wedge\ \underline{q}_z \leq \overline{p}_z \end{aligned}$$

We will denoted this $p \boxplus q$.

This means that the cuboids either overlap or abut each other. For each dimension we introduce the operators $\boxplus_x, \boxplus_y$ and $\boxplus_z$ which correspond to overlap-equal in the x-dimension, y-dimension and z-dimension, respectively.

5.2 Boxes

The boxes are the small items that we want to place inside the container and they are per definition cuboids. Their size is denoted (l_i, w_i, h_i) (length, width, height) as seen in Chapter 4. Each box i is also characterised by a mass, which we will denote m_i. We will assume that the geometrical and gravitational centres coincide and that the mass is homogeneously distributed in the boxes. Furthermore boxes can be fragile. If this is the case then no other boxes should be placed on top of them. This is indicated by the parameter f_i for box i. If box i is fragile then $f_i = 1$, otherwise $f_i = 0$.

When multi-drop constraints are introduced, it becomes important to know which boxes should be delivered at each customer. This imposes a so called *sequence number* on each box. The boxes that should be dropped off at the first customer get sequence number 1, boxes with sequence number 2 should be dropped off at the second stop and so on. We call the sequence number of box i d_i.

When solutions are constructed, we need to make decisions regarding how a box should be rotated. To a box i a set of feasible rotations $fr \in FR_i$ exists.

Without any rotation restrictions, the boxes can be rotated in six different ways, see Figure 5.1. Sometimes, however, this is not desirable due to the goods contained in a box. This is often illustrated with a "this side up" mark on the boxes, which means that the given box is restricted in possible rotations. In this

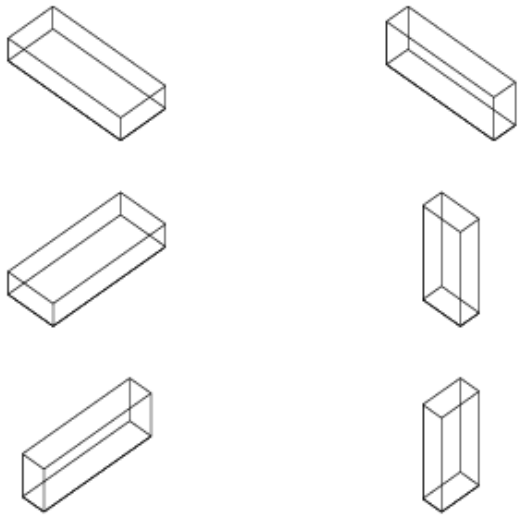


Figure 5.1: Possible rotation of a box. On the figure the box with initial dimensions (l, w, h) is rotated to give the following 6 possibilities; $(l, w, h), (l, h, w), (w, l, h), (w, h, l), (h, l, w)$ and (h, w, l).

model only the vertical rotation, that is, the dimension that is placed vertical, can be limited. This means that for a box where only one side can be the upside, there exists two possible rotations. If two different sides can be the upside, four feasible rotations exist and finally if all three sides can be the upside, the six rotations shown in Figure 5.1 are all feasible.

Finally, the notion of a *box type* must be introduced. Two boxes have the same box type, if all the above attributes, except the sequence number, are equal. This means that they must have the same size, fragility, mass and rotation restrictions. We will say that box i has box type bt_i.

When a box is placed inside the container, the placement and rotation of the box is chosen. When this is done the *extension* along the x-, y- and z-axis is given by Δa_i, Δb_i and Δc_i, respectively.

This means that the *minimum* of a box i is given by its placement (a_i, b_i, c_i), which is also denoted: $\underline{i} = (\underline{a}_i, \underline{b}_i, \underline{c}_i)$. The *maximum* of a box i is given by its placement plus extension, $(a_i + \Delta a_i, b_i + \Delta b_i, c_i + \Delta c_i)$, which is also denoted: $\overline{i} = (\overline{a}_i, \overline{b}_i, \overline{c}_i)$.

5.2.1 The batch - a set of boxes

When a set of boxes is considered it will be denoted a batch, B. To a batch there exist some properties, namely how many boxes of each box type there exist in the batch, and information about the smallest dimension of a box present in the batch.

The set $\mathcal{B}$ holds for every box type bt, the number k_{bt} of boxes of that box type. The smallest dimension among all the boxes in the batch is also at all times known. The smallest dimension is denoted sm.

When we load boxes into the container, we will work with two distinct batches, namely the batch containing already placed boxes B_{placed} and the batch with not placed boxes $B_{nPlaced}$. Notice that when no boxes are placed $B_{placed} = \emptyset$ and $B_{nPlaced} = B$. At all times

$$B_{placed} \cap B_{nPlaced} = B \quad \wedge \quad B_{placed} \cup B_{nPlaced} = \emptyset$$

must hold.

5.3 The container

A feasible solution to the CLP consists of a number of boxes placed inside the container. Thereby, the container defines the natural physical bound of where the boxes can be placed. The container is a cuboid and its size can be described by its three interior dimensions: Length L, width W and height H. The container will always be placed such that its minimum coincides with the origin of the co-ordinate system and the length expands along the x-axis, width along the y-axis and height along the z-axis.

When a box is placed in the container, part of the space in the container is occupied, and the remaining possible space to place the next box is reduced. This calls for a way to manage the space inside the container, so that the possible positions to place boxes are known.

In the literature two distinct ways to handle the space inside the container are described. Ngoi et al. [25] introduced a matrix representation of the container, which was modified by Bischoff [2]. Here the first row and column describe different areas inside the container and the remaining elements, the space left from the container floor or the placed boxes to the container roof. Another approach was pointed out by George and Robinson [16]. They used what is called *empty spaces* to handle the space inside the container. We have chosen the latter approach, which is also the most common. In the following a detailed description of how we handle the space inside the container is given.

5.3.1 The space inside the container

A way to look at the problem is that we want to fill out the empty space inside the container. Each time a box is placed in the container, the empty space is reduced. Following this thought, the container at first can be considered as a single empty space the size of the container. When more boxes are placed it will be divided in smaller cuboid spaces.

Definition 5.4 (Empty space) An *empty space* s is a cuboid inside the container. It has minimum in $\underline{s} = (\underline{x}_s, \underline{y}_s, \underline{z}_s)$ and maximum in $\overline{s} = (\overline{x}_s, \overline{y}_s, \overline{z}_s)$. The space inside this area *must be empty*. Empty spaces will for short also be denoted *spaces* in the following.

The size of the space s is denoted $(\Delta x_s, \Delta y_s, \Delta z_s)$ and found as:

$$\begin{aligned} \Delta x_s &= \overline{x}_s - \underline{x}_s \\ \Delta y_s &= \overline{y}_s - \underline{y}_s \\ \Delta z_s &= \overline{z}_s - \underline{z}_s \end{aligned}$$

Corollary 5.5 *No placed boxes overlap with valid empty spaces. This means that a space $s \in S$ is valid if:*

$$\forall i \in B_{placed} \quad \neg(i \boxdot s)$$

This situation should be attained, for all spaces, every time a box is placed in the container.

When a box is added to the container, it is placed in a space. This makes the space occupied and therefore it is invalid (Corollary 5.5). Instead of this space, new empty spaces should be made, surrounding the box, but still being

bounded by the old space. Therefore an updating procedure for the empty spaces is needed. This is described in Section 5.3.2.

In principle, a box can be placed anywhere in an empty space as long as it is fully contained by the space. However, to reduce the solution space, we will only consider the situation where a box is placed in the point in the empty space closest to the origin of the coordinate system. This is the minimum of the empty space, and follows the concept of a normalised packing, see [10]. It can be shown that to any packing, an equally good normalised packing exists. This was done by Martello et al. [22], when packing a single bin in the three dimensional bin packing problem and by Herz [18] for the two dimensional stock cutting problem. This means that we have a discrete representation of the possible places, where a box can be placed.

Definition 5.6 (Placement of boxes in empty spaces) Boxes are always placed in the minimum of an empty space.

To have a feasible packing, it is however not enough to assure that the place, where a box is placed, is not occupied by other boxes. In a real world scenario, the box must be placed, either on the container floor or on top of other boxes. This gives the box support from below, and hence increases the stability of the cargo. When a box is placed, we therefore further demand that it has full support. A box has full support when the *base surface* of the box has full contact with the surface below. The surface below could either be other boxes or the container floor.

Observation 5.7 *When working with full support, the height of empty spaces is always given by the container roof. Thereby all spaces have the same* $\overline{z}$.

This observation makes it easier to update the empty spaces as it means, that there is no need to consider changes to $\overline{z}$.

Full support is a reasonable constraint, as it assures that boxes will never float in the air and it stabilises the load. The constraint could of course be relaxed to allow a solution with less then 100% support for all boxes. This is done in Chapter 6. In the next section it will be shown that full support can be assured by simple considerations of how to update the spaces.

5.3.2 Updating the empty spaces

A number of strategies can be considered when modelling the empty spaces, but there are two basically different approaches. For both methods a choice can be made, whether full support for all boxes placed in the container, is guaranteed. The difference between the methods is that either the spaces are allowed to overlap or not. Models where no overlap is allowed are used by for instance, Moura and Oliveira [24], Bortfeldt et al. [5] and George and Robinson [16]. Models allowing overlapping spaces has been tried by Eley [10], Bischoff and Ratcliff [3] and Parreño et al. [26]. Figure 5.2 shows an example of a box

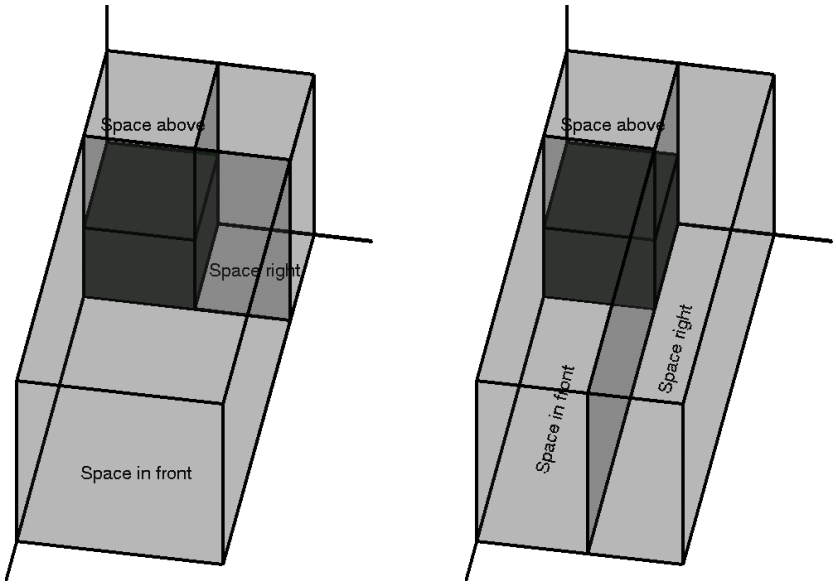


Figure 5.2: The empty spaces generated inside the space where the box is placed. No overlap is allowed. The left picture shows the empty spaces generated, if walls against the back of the container is build. The right picture shows empty spaces that lead to walls along the container sides.

placed in a container with empty spaces that do not overlap. Figure 5.3 shows the same situation with overlapping spaces. In the first case it is clear that a choice has to be made regarding which situation is wanted. It is not possible to get the spaces shown on the left and right graph at the same time. The main problem with overlapping spaces, however, is, that a box placed in a space will possibly affect other spaces, leading to a more complicated update procedure. Thereby disadvantages exists to both methods. We find that a rule of non-overlapping spaces will limit the solution space too much, thereby removing

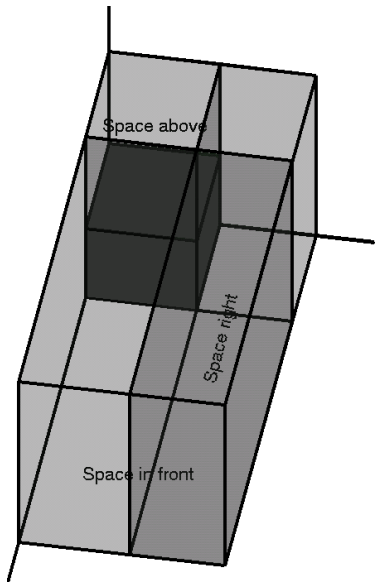


Figure 5.3: The empty spaces generated inside the space where the box is placed. Overlap is allowed.

good solutions. Thus, we will allow spaces to overlap.

5.3.2.1 Generating new spaces

When a box is placed in the container, the empty spaces must be recalculated and adapted. At least one empty space is made infeasible, namely the empty space in which the box is placed. The infeasible space must be deleted and instead new valid empty spaces inside the old space must be generated. All new empty spaces must of course respect Definition 5.4, otherwise they would not be valid.

Lets say that we place a box i, with minimum $(\underline{a}_i, \underline{b}_i, \underline{c}_i)$ and maximum $(\overline{a}_i, \overline{b}_i, \overline{c}_i)$, in a space $s \in S_{old}$, with minimum in $(\underline{x}_s, \underline{y}_s, \underline{z}_s)$ and maximum in $(\overline{x}_s, \overline{y}_s, \overline{z}_s)$. In general three new empty spaces are generated. One above, one in front and one right of the box. The resulting new spaces is shown in Table 5.1. Here space $t = 1$ is placed in front of, $t = 2$ right of and $t = 3$ above the box. The space above is only made if the box i is not fragile, thereby no boxes can be placed on top of fragile boxes. The maximum of space 1 and 2 are only limited by the old space, s. This is not the case for space 3, which is limited by the placed

	min			max		
t	$\underline{x}_t$	$\underline{y}_t$	$\underline{z}_t$	$\overline{x}_t$	$\overline{y}_t$	$\overline{z}_t$
1	$\overline{a}_i$	$\underline{y}_s$	$\underline{z}_s$	$\overline{x}_s$	$\overline{y}_s$	$\overline{z}_s$
2	$\underline{x}_s$	$\overline{b}_i$	$\underline{z}_s$	$\overline{x}_s$	$\overline{y}_s$	$\overline{z}_s$
3	$\underline{x}_s$	$\underline{y}_s$	$\overline{c}_i$	$\overline{a}_i$	$\overline{b}_i$	$\overline{z}_s$

Table 5.1: The three resulting spaces constructed after an insertion is made.

box in both the x- and y-dimension. This is done to always assure full support in the empty spaces. After the procedure the set of spaces S_{new} is given by $S_{new} = \{(S_{old} \setminus s) \cup N\}$, where N is the set of valid empty spaces from Table 5.1. Two spaces are overlapping, namely space 1 and 2. This can also be seen on Figure 5.3.

5.3.2.2 Remove box overlap with other spaces

Using overlapping spaces has the consequence that when we place a box, we have to investigate the remaining spaces to see if there are other spaces made invalid by the newly placed box. If a space s is made invalid by a placed box i, the space is deleted and instead new spaces are generated inside s, but not invalidated by the placed box. This means that four different spaces, seen in relation to the box, theoretically can be created. These could be placed in front of, behind and on both sides of the box. The calculations used to make the new spaces are given in Table 5.2. Here space $t = 1$ is behind, $t = 2$ in front of,

	min			max		
t	$\underline{x}_t$	$\underline{y}_t$	$\underline{z}_t$	$\overline{x}_t$	$\overline{y}_t$	$\overline{z}_t$
1	$\underline{x}_s$	$\underline{y}_s$	$\underline{z}_s$	a_i	$\overline{y}_s$	$\overline{z}_s$
2	$\overline{a}_i$	$\underline{y}_s$	$\underline{z}_s$	$\overline{x}_s$	$\overline{y}_s$	$\overline{z}_s$
3	$\underline{x}_s$	$\underline{y}_s$	$\underline{z}_s$	$\overline{x}_s$	b_i	$\overline{z}_s$
4	$\underline{x}_s$	$\overline{b}_i$	$\underline{z}_s$	$\overline{x}_s$	$\overline{y}_s$	$\overline{z}_s$

Table 5.2: The four spaces that can be made when a placed box i overlap with a space s, which is different from the space the box is placed in.

$t = 3$ and $t = 4$ on each side of the box. Again only empty spaces that respect Definition 5.4 are made. An illustration of new empty spaces surrounding a placed box is shown in Figure 5.4. On the left figure it is seen that a box has been placed, which invalidate a different space than the one it is placed in. On the right figure the new spaces, which are generated from the invalidated space,

are shown. The new spaces are all fully contained within this. Three spaces

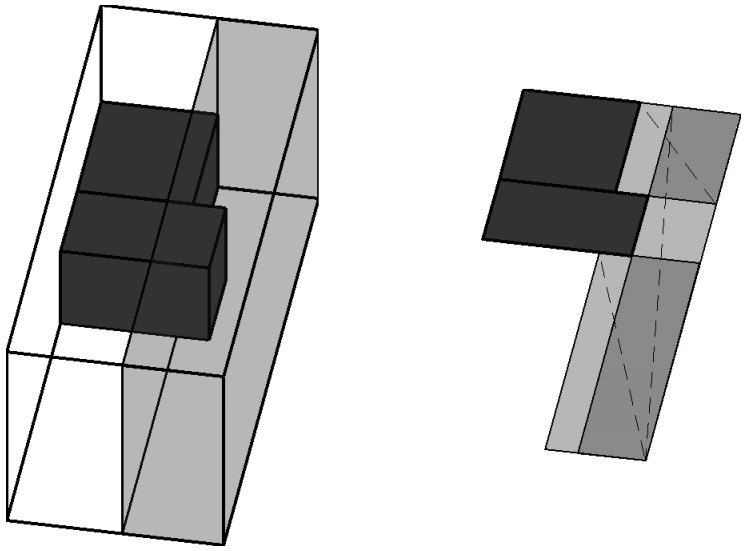


Figure 5.4: Illustration of new spaces surrounding a placed box.

are generated, one behind, one in front and one right of the placed box. The spaces are indicated with the dashed lines, going from their minimum to their maximum. The dark grey indicates where the spaces overlap each other. Three of the four cases from Table 5.1 occur.

It could be argued, that a new space above the invalid empty space should be made. But, this space is guaranteed to be a *subset* (see Section 5.3.2.3) of space 3 from Table 5.1, created when the box was inserted in the first place, and it is therefore not created.

It can be realised, that the method to remove box space overlaps, can be generalised to include the special case, where the space considered is the space where the box is placed. This is done by including the space made above the placed box from Table 5.1 to the spaces made by Table 5.2.

5.3.2.3 Removal of subsets

When updating overlapping empty spaces, it is possible that two spaces cover the same areas of the container. Thereby a space can be considered a *subset* of

another space. The concept of subsets is also used by others, see for instance Bischoff and Ratcliff [3].

Definition 5.8 (Space subsets) An empty space s is a *subset* of another space t if:

$$\underline{s} \geq \underline{t} \quad \wedge \quad \overline{s} \leq \overline{t}$$

This means that space s is a subset of space t if, all points in s are contained in t, meaning that the minimum of s is greater than the minimum of t and that maximum of s is smaller than the maximum of t. If this happens, we do not need space s anymore because it is completely described by t.

5.3.2.4 Amalgamation of the empty spaces

The way the spaces are constructed assures full support for the boxes, but sometimes the method limits the spaces to much. The method has a secondary effect of creating stacks where a box on top of another at most will be as large as the one below. This is not always desirable, as it can give an ineffective use of the space in the container.

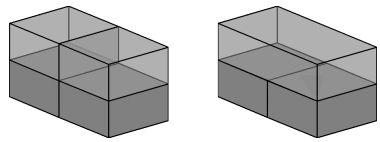


Figure 5.5: Two boxes with the same height placed beside each other. On the left picture the individual spaces that are generated after the spaces updating procedure. On the right picture a new combined space is shown.

Consider the situation where two equally tall boxes are placed beside each other, it would be possible to place a box on top of both boxes, still meeting the demand for full support. An example is shown on Figure 5.5. In the way the spaces are generated, this would not be allowed, as there would be two empty spaces, one on top of each of the boxes, but neither overlapping with the space over the other box (left). In this case it should be possible to generate a space that expands over both boxes (right). This operation is called *amalgamation* of empty spaces and was introduced by George and Robinson [16].

We are able to amalgamate two spaces s and t when the following expression is true.

$$\underline{z}_s = \underline{z}_t \wedge \left[\Big(s \boxdot_x t \wedge (\underline{y}_s = \overline{y}_t \vee \underline{y}_t = \overline{y}_s) \Big) \vee \Big(s \boxdot_y t \wedge (\underline{x}_s = \overline{x}_t \vee \underline{x}_t = \overline{x}_s) \Big) \right] \quad (5.1)$$

This is the case if the two spaces s and t have the same floor-height ($\underline{z}_s = \underline{z}_t$). Moreover it should be true that their borders touch ($\underline{x}_s = \overline{x}_t \vee \underline{x}_t = \overline{x}_s \vee \underline{y}_s = \overline{y}_t \vee \underline{y}_t = \overline{y}_s$) and finally the base dimension that does not touch should be overlapping ($s \boxdot_y t \vee s \boxdot_x t$, respectively).

From the square bracketed expression in Equation (5.1), we see that two different things can lead to the amalgamation of two empty spaces, namely overlap in the x- or in the y-dimension. Both situations can be seen in Figure 5.6, where

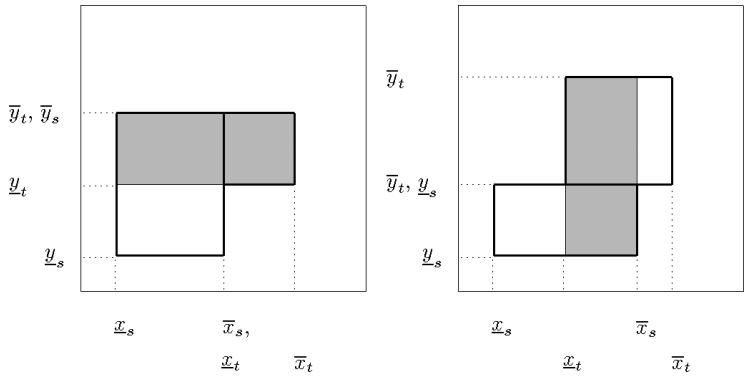


Figure 5.6: Amalgamation of the two spaces s and t. On the left, amalgamation in the x-direction is shown. On the right, amalgamation in the y-direction is shown.

the grey rectangles indicate the new space obtained by amalgamating the two old spaces drawn with bold lines.

If we are able to amalgamate two spaces, and overlap between these are in the x-dimension, the new amalgamated space will have greater extension in the y-dimension. We will say that we *amalgamate in the y-direction*. If two spaces can be amalgamated and overlap between these exists in the y-dimension, we will say that we *amalgamate in the x-direction*.

Amalgamate in x-direction

Two spaces will be amalgamated in the x-direction if the following hold:

$$\underline{z}_s = \underline{z}_t \wedge ((s \boxdot_y t \wedge (\underline{x}_s = \overline{x}_t \vee \underline{x}_t = \overline{x}_s))$$

If this is the case, the new empty space u is given as:

$$\begin{aligned}
\underline{x}_u &= \min(\underline{x}_s, \underline{x}_t) \\
\underline{y}_u &= \max\left(\underline{y}_s, \underline{y}_t\right) \\
\underline{z}_u &= \underline{z}_s \\
\overline{x}_u &= \max(\overline{x}_s, \overline{x}_t) \\
\overline{y}_u &= \min(\overline{y}_s, \overline{y}_t) \\
\overline{z}_u &= \overline{z}_s
\end{aligned}$$

This is shown in Figure 5.6, left.

Amalgamate in y-direction

Two spaces will be amalgamated in the y-direction if the following hold:

$$\underline{z}_s = \underline{z}_t \wedge ((s \boxdot_x t \wedge (\underline{y}_s = \overline{y}_t \vee \underline{y}_t = \overline{y}_s))$$

If so, the new empty space u is given as:

$$\begin{aligned}
\underline{x}_u &= \max(\underline{x}_s, \underline{x}_t) \\
\underline{y}_u &= \min\left(\underline{y}_s, \underline{y}_t\right) \\
\underline{z}_u &= \underline{z}_s \\
\overline{x}_u &= \min(\overline{x}_s, \overline{x}_t) \\
\overline{y}_u &= \max(\overline{y}_s, \overline{y}_t) \\
\overline{z}_u &= \overline{z}_s
\end{aligned}$$

This is shown in Figure 5.6, right.

When two spaces are amalgamated to make a new empty space, both are still valid. But to keep the number of empty spaces low subsets (see Definition 5.8) are removed. This would be the case in Figure 5.6, left, where the rightmost space is fully contained in the new grey space.

On the container floor, spaces are only limited by the container size and other placed boxes, as full support exists. Therefore, amalgamation will never expand these. Hence, when updating empty spaces, amalgamation should only be considered for spaces above the container floor.

5.3.2.5 Removal of too small empty spaces

As more and more boxes are placed in the container, the spaces will gradually be reduced in size. If a space is smaller than all boxes not yet loaded, then it should be discarded. However, if there is a possibility to amalgamate the space later on in the loading process it should be kept. The fewer spaces we have to consider when finding a solution, the easier. The strictest check would compare all three dimensions of the space with the dimensions of the boxes. To simplify the procedure of removing too small empty spaces, we have chosen only to consider one dimension of the boxes and spaces. Therefore only the smallest dimension in the batch of not placed boxes, sm, is compared with the dimensions of the spaces.

space] A space $s \in S$, can be deleted if:

$$\left\{ \begin{array}{l} \forall t \in S \setminus s : \Delta z_s < sm \vee \\ \quad \Big((\Delta x_s < sm \vee \Delta y_s < sm) \ \wedge \ \neg \big((s \boxplus_x t) \wedge (s \boxplus_y t) \wedge \underline{z}_s \geq \underline{z}_t \big) \Big) \end{array} \right\} \tag{5.2}$$

where $\boxplus$ is the overlap-equal operator defined in Definition 5.3. From Definition 5.2, it is seen that spaces can generally be deleted if the extension in one dimension is smaller than the smallest available box dimension, sm. However, if it is possible to expand the size of the space in that direction it should not be deleted anyway. It is only possible to expand spaces by amalgamation, which again only is possible in the x- and y-direction. Therefore if the expansion of the space in the z-dimension is smaller than sm the space is always deleted. If it is in the x- or y-dimension, we further demand that no other spaces t exists which fulfill both $s \boxplus_x t$, $s \boxplus_y t$ and $\underline{z}_s \geq \underline{z}_t$, meaning that it should overlap-equal both in the x- and y-dimensions and be placed lower.

5.4 A valid box placement

The procedure for placing a box in the container is to match it with an empty space. As mentioned earlier a box is always placed in the minimum of the space.

When a box is placed in a space, two things should be considered, to make the placement valid. It must be, feasible regarding the space and multi-drop feasible.

5.4.1 Feasible regarding a space

The box is feasible regarding the space, when the box, with the chosen rotation, can be placed in the space without protruding the boundaries of the space.

More formally this can be expressed as: Given the box i and the space s, there is room for i in s if:

$$\Delta a_i \leq \Delta x_s \ \wedge \ \Delta b_i \leq \Delta y_s \ \wedge \ \Delta c_i \leq \Delta z_s \tag{5.3}$$

If the box was allowed to protrude the space one of two things would happen. Either the box would overlap other boxes placed in the container or the placed box would not be stable. This is certain because of the way the spaces are generated.

If there were no other constraints on the problem than fitting the boxes into the container this could do as a check. In our situation we also need to check if the multi-drop constraint is satisfied.

5.4.2 Multi-drop feasibility

When the container arrives at a customer, naturally the boxes belonging to the customer have to be taken out. If all boxes go to the same customer this is a trivial exercise. If there are more customers on the route, care has to be taken when the container is loaded, if unloading the container should be easy. A given box should be available and easy to take out when the container arrives at the customer who requested the box. This is handled by the multi-drop constraint.

To make sure that these demands are respected, we require that no boxes with higher sequence number is placed above the given box or in the *passage* from the box to the entrance/exit of the container.

In Figure 5.7 two different situations are depicted. The unloading direction is shown with an arrow. The red boxes belongs to customer 2, while the blue box belongs to customer 1. This means that the blue box should be unloaded before the red ones. On the left graph, the blue box is placed behind one of the reds. Thereby, it is no longer possible to take it out, and that placement is not

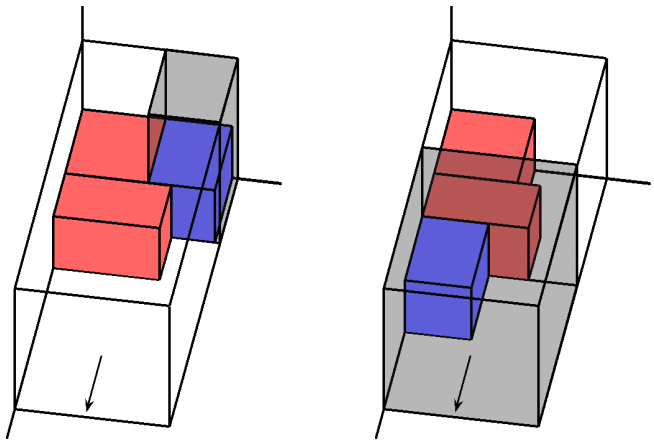


Figure 5.7: Not valid (left) and valid (right) placement of a box.

multi-drop feasible. On the right graph, the blue box is placed in front of the red ones which is fully legal.

5.4.2.1 Boxes placed on top of each other

If a box is placed above another box, either fully or partly, we say that they overlap in the $\{x, y\}$-plane. If this is the case for boxes i and j then:

$$i \boxdot_x j \wedge i \boxdot_y j$$

is true.

Lets assume that two boxes are overlapping in the $\{x, y\}$-plane and without loss of generality that box i is placed under box j. For the multi-drop constraints to hold, we demand that the sequence number d_i of box i is greater than or equal to the sequence number d_j of box j, i.e. $d_i \geq d_j$. This means that if two boxes overlap in the $\{x, y\}$-plane then the box on top should be the one with the lowest sequence number.

5.4.2.2 A passage between the entrance and a box

In this model we work with one entrance to the container. Without loss of generality we consider the surface with coordinates: $(L, 0, 0) - (L, W, H)$ to be

the entrance. In Figure 5.7 this would be the surface indicated by the arrow.

To be able to unload box i when the container arrives at the customer who has requested the box, there must be a passage from the entrance of the container to the box.

Definition 5.9 (Passage) There exist a *passage* between the entrance of the container and the box i if:

$$\forall j \in B_{placed} \setminus i \mid a_i \geq a_j \vee d_i \geq d_j \vee \ \neg(i \boxdot_y j)$$

The opposite situation, namely when there does not exist a passage is intuitively easier to understand. There does not exist a passage if:

$$\exists j \in B_{placed} \setminus i \mid a_i < a_j \wedge d_i < d_j \wedge (i \boxdot_y j)$$

meaning that box i has lower sequence than some other box which is placed in front of it.

On the left picture in Figure 5.7 there does not exist a passage between the blue box and the entrance of the container, because one of the red boxes is in the way.

Definition 5.10 (Multi-drop feasible packings) A solution to the container loading problem respects the multi-drop constraints if: For all boxes i there exists a *passage*, and no boxes j placed on top of i with overlapping $\{x, y\}$-plane, have higher sequence number.

To maintain a multi-drop feasible packing, each time a box is placed, it must compared to all other placed boxes, to assure they are still multi-drop feasible.

It could be considered to put the information about allowed sequence numbers directly on the spaces. This would decrease the number of checks needed when a box is inserted. Such a decision would, however, increase the complexity of the space handling and was thus not carried out.

5.5 Summary

In this chapter we have introduced a mathematical model, describing a realistic single container loading problem. Besides describing the boxes and the container, the model consist of two main elements, updating empty spaces and checking for valid placements. This model offers the basic operations needed to create heuristic solutions to the problem.

When updating empty spaces a number of operations must be carried out. This include, the generation of new empty spaces, amalgamation and removal of subsets and too small spaces. A number of conditions should be checked before a box is inserted in an empty space. First of all enough room should be present and secondly we demand that the insertion of a box does not make any boxes multi-drop infeasible. The generation of empty spaces assures that fragile boxes are placed safely and that full support for the boxes is guaranteed. Thereby all the demands we have for a realistic model is achieved.

Chapter 6

Industrial insight

The model presented in Chapter 5 can handle basic constraints, which are necessary to include in the CLP, if it should be used in connection with a real world vehicle routing application. When dealing with problems faced by specific companies or industries, it must be expected, that the model should be adapted to handle different or further constraints. In this chapter, we will analyse and show the possibilities of incorporating such extensions, in the proposed model. This is done by modelling two very different problems from companies facing the challenge of single container loading with multi-drop constraints in their everyday business.

6.1 Aarstiderne A/S - Distribution of vegetables

Aarstiderne A/S is a Danish company delivering organic vegetables, fruit and meat to the door step of 35.000 families all over Denmark. The delivery is on a weekly basis and the customers often change their requests. Aarstiderne offer around 30 distinct products with different selections of fresh vegetables, fruit or meat. Additionally, grocery products, such as cereals, wine and beer, are offered. The different products are all packed in boxes, of which there exists 10 different types. The different products are designed to meet the needs of

different sized families. Therefore, typically one or two boxes are demanded by each customer. In a typical packing problem there are between 100 and 200 boxes going to between 70 and 150 different customers.

Aarstiderne A/S both handle the packing of the groceries in the boxes as well as the overall distribution of the boxes. The boxes are distributed to depots around the country, and from each of these depots the boxes are delivered to the customers. It is this last delivery we have focussed on, as this is where the multi-drop constraint becomes most important.

Today, the vehicles are loaded, by the drivers. In the depot, the boxes with the same products are stacked together, see Figure 6.1. This means that the drivers have to walk around collecting the boxes needed on the route, as opposed to a situation, where the boxes are ranked in front of the vehicle slots. As such,

Figure 6.1: The distribution centre in Bjæverskov, Denmark

every driver has his own way of loading the vehicle. However, some common characteristics exist. The loads are to the greatest extent made so that, all the different products are available, at all times on the route. This is the way the multi-drop constraint is handled today. To minimise time used loading the vehicle, most drivers load the same products close together. We have not focused on this point in our model. The extra time used for the driver to pack a vehicle should be compared to the time saved unloading the vehicle on the route. Finally, when the driver drop of boxes at the customers, empty boxes

from the last delivery should be picked up. This means that we have a delivery and pick-up system. It is, however, not a strict constraints that all empty boxes must be brought back, and moreover, is it significantly easier to bring back the empty boxes, as they can be placed inside each other, taking up much less room. Therefore, we have chosen to neglect this issue.

6.1.1 Changes made to the model

The change made to the model is motivated by the fact, that a box only gives support to boxes placed above it, through its four corners. This is because Aarstiderne use open wooden boxes with no lid on. Two different boxes are depicted in Figure 6.2. If two boxes are placed on top of each other and the box on top has smaller base dimension than the one below, it would mean that the box above would not be supported and thereby damage the contents of the box below. Therefore, to require full support, boxes can only be stacked when all four corners of the boxes are resting directly above corners of a box below. As long as the corners of the box are supported from below, there is no limit on (except for the container size) how many boxes can be stacked on top of each other. If high stacks have no support on the sides from other boxes, extendable poles are put up between the floor and ceiling in the container to make the load stable.

Figure 6.2: A picture of the "basic box" (left) and "fruit box" (right) with the boxes they are packed in.

To make sure that boxes are properly stacked, extra care should be taken when they are inserted on top of other boxes. This changes both the way we manage spaces but also the definition of a valid box placement.

The following procedures are changed:

- A valid box placement
- Update empty spaces:
  - Amalgamation
  - Remove subsets

It is important to notice that when placing a box on the container floor, support is always guaranteed. This means that the following changes only concerns boxes placed on top of other boxes. In the same way a change in the amalgamation rule is only needed above the container floor.

6.1.1.1 A valid box placement

To be able to guarantee that boxes are supported in all four corners when placed, we check at insertion time whether this condition is true. For box i in space s the check is:

$$\begin{aligned} \Delta a_i &= \Delta x_s \wedge \\ \Delta b_i &= \Delta y_s \wedge \\ \Delta c_i &\leq \Delta z_s \end{aligned} \tag{6.1}$$

This check is used whenever a box is *not* placed on the floor. The check presumes that all spaces not starting at the floor are sized so that the corners have support. It should of course still be checked if a placement is multi-drop feasible as defined in Section 5.4.2, page 46.

6.1.1.2 Update empty spaces

Since only boxes stacked directly on top of each other are valid, there is no longer any need to amalgamate empty spaces when the spaces are above the container floor. Therefore, the amalgamation is discarded for the spaces that are not starting on the container floor. Recall that amalgamation is not done on the container floor. This means that whenever working with problems like the ones from Aarstiderne, amalgamation should not be part of the procedures needed to update the empty spaces.

Since amalgamation is discarded and we use a new valid box placement check, when a box is placed above ground, no spaces will be subsets of other spaces, provided that they are above the container floor. Hence, there is no need to try to remove subsets of these.

6.2 Johannes Fog A/S - distribution of construction products

Johannes Fog A/S deals with some very different products than Aarstiderne. They handle construction products, such as gypsum boards, rockwool, concrete, mortal, wooden planks, windows and doors. This makes the consignments extremely diverse. The products are delivered to around 50 customers every day, most of them in the morning and the rest in the afternoon. A typical load consist of 10-20 different products going to between 1 and 5 customers.

Johannes Fog use trucks, that can be unloaded from the sides as well as the back. Therefore, we need to be able to reflect this relaxation of the multi-drop constraint when solving the problem. Moreover when solving this problem, it is necessary to relax the constraint demanding full support. Full support does not allow enough flexibility, compared to a real world scenario, when loading construction products. A five metre long wooden plank for example, would still be stable if for instance only four metres were supported. Instead of full support, it is demanded that at least a fixed fraction of each box should be supported. Finally, the diversity of the consignments makes it important to know which boxes can be stacked, when loading the container. It may be feasible to pack a small cardboard box on top of gypsum board, whereas it is not feasible to place a load of concrete on top of rockwool. This introduces need for a measure of how much weight each box can carry.

To sum up, these modifications to the model must be made:

- The multi-drop constraint must reflect that more unloading directions exist.
- Reduced support is introduced to allow boxes to overhang each other.
- Load bearing strength is used to make sure that boxes are stacked properly.

6.2.1 The multi-drop constraint

Figure 6.3 shows a truck operated by Johannes Fog A/S. From the picture it is

Figure 6.3: A Johannes Fog truck.

evident that boxes can be loaded and unloaded from three sides of the container. As the products generally are handled with crane and forklift, it is not possible to unload the boxes vertically without having access from a side.

To model this situation we need to change Definition 5.10, regarding multi-drop feasible packings. Since more unloading directions exist, the four parameters X^+, Y^+ and Y^- are introduced, as properties on the container. They indicate whether the container surfaces $(L,0,0)-(L,W,Z), (0,W,0)-(L,W,Z)$ and $(0,0,0)-(L,0,Z)$ are unloading surfaces.

In the realistic model from Chapter 5, we used the concept of a *passage* to reveal if each box was multi-drop feasible. This passage was define for unloading direction X^+, which must be generalised to encompass the other unloading directions. For a load to be multi-drop feasible, we demand that a passage exists, for all placed boxes, in at least one of the possible unloading directions. In the case from Johannes Fog three unloading directions are feasible, namely Y^-, Y^+ and X^+. The passage for unloading direction X^+ is given in Definition 5.9. Then unloading directions for Y^- and Y^+ are given in Definitions 6.1 and

6.2.

Definition 6.1 (A passage in Y^-) There exist a passage for box i in the Y^- direction if:

$$\forall j \in B_{placed} \setminus i \mid b_i \leq b_j \vee d_i \geq d_j \vee^{\neg} (i \boxdot_x j)$$

Definition 6.2 (A passage in Y^+) There exist a passage for box i in the Y^+ direction if:

$$\forall j \in B_{placed} \setminus i \mid b_i \geq b_j \vee d_i \geq d_j \vee^{\neg} (i \boxdot_x j)$$

With the extra passage definitions, we can redefine a multi-drop feasible packing.

Definition 6.3 (Multi-drop feasible packings) A solution to the CLP respects the multi-drop constraints if: For all boxes i, there exist a passage in a possible unloading direction, and no boxes j on top of i with overlapping $\{x, y\}$-plane, have higher sequence number.

For Johannes Fog Definition 6.3, tells that in a feasible load a passage should exist, for all boxes, in at least one of the directions Y^-, Y^+ or X^+. To keep information about which unloading directions are available for a placed box, we introduce the boolean variables x^+, y^+ and y^-. These are kept as properties on each box. When boxes are placed these values are updated, in relation to the possible unloading directions.

Notice, that a generalisation to also incorporate X^- is straight forward.

6.2.2 Reduced support of the boxes

This is by far the most complex change to the model we will introduce. It affects a large number of operations performed on the empty spaces. A model were reduced support is considered - and guaranteed - have been proposed by Bortfeldt et al. [5], but they consider non-overlapping spaces.

The idea is to adjust the spaces so that they overhang boxes below, as the empty spaces define where boxes can be placed. This allows the boxes to overhang each other, reducing the demand for support.

When a box is placed in a space, we always place it in the minimum of the space according to Definition 5.6, page 37. If the space above a box is allowed to overhang all sides of the box below, a box placed in the overhanging space

could end up without any support at all. To avoid this, we demand that spaces begin in a point, which is supported from below. This will guarantee that a small box placed in a large space is fully supported. The problem is illustrated in Figure 6.4. In the situation on the left picture the space overhangs the box

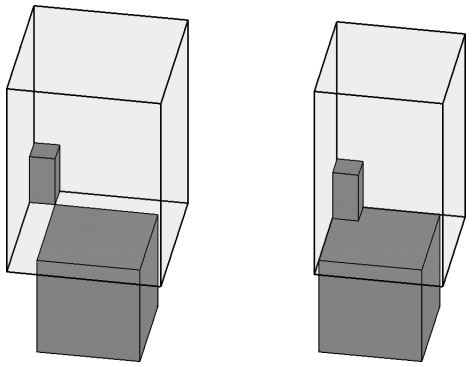


Figure 6.4: A small box placed in a large space. In the left figure the space overhang the minimum of the box below. On the right it starts where support from below exists.

below on all sides. This makes it possible to place a box with no support at all, which is not reasonable. On the right figure, the same box is placed in the space starting in a point with support, thereby guaranteing support for the box. This way of assuring support for the boxes is also used by Bortfeldt et al. [5]. A disadvantage is that many types of stable loads, with reduced support, cannot be constructed. However, it allows the remaining of the introduced model to be unchanged.

To control how much support boxes must have, the non-negative parameter γ is introduced. This parameter controls how large a part of the space that does not need support. If $\gamma = 1$, the supported and the not supported parts of the space have the same size. If $\gamma = 0$, no overhang is allowed. In a practical setting $\gamma < 1$, as this implies that the geometric centre of the box is supported from below. Recall that we assume, that the geometrical and gravitational centres coincide (see Section 5.2, page 33). This is a reasonable assumption when packing for Johannes Fog. In nearly all their products the mass is evenly distributed. This is sound for wood, stones, rockwool, gypsum etc.

The possible extension of a space depends on the amount of support from boxes below. We define support as continuous contact with a box below or the container floor in a direction. To determine the allowed size of the overhanging empty space, we need to know the extension of support from below. We have constructed the empty spaces such that support always exist in the minimum

of the spaces.

Therefore we introduce two variables α and β, holding knowledge about the maximum support from below in a space in the x- and y-dimension, respectively. Notice that, if $\gamma = 0$ then $\alpha = \overline{x}$ and $\beta = \overline{y}$. The initial empty space (representing the empty container) has full support, by definition. This could be relaxed to allow overhang out of the container. This however, is not considered in this work.

The introduction of α and β changes the definition of an empty space, found in Definition 5.4, page 36. Besides what is stated there, a space s has the support parameters (α_s, β_s) and:

$$\underline{x}_s < \alpha_s \leq \overline{x}_s \wedge \underline{y}_s < \beta_s \leq \overline{y}_s$$

6.2.2.1 Updating the empty spaces

If we place a box i in a space s with support parameters (α_s, β_s), the resulting new spaces can be seen in Table 6.1.

	min			**max**		
t	$\underline{x}_t$	$\underline{y}_t$	$\underline{z}_t$	$\overline{x}_t$	$\overline{y}_t$	$\overline{z}_t$
1	$\overline{a}_i$	$\underline{y}_s$	$\underline{z}_s$	$(1+\gamma)\cdot\alpha_s - \overline{a}_i \cdot \gamma$	$\overline{y}_s$	$\overline{z}_s$
2	$\underline{x}_s$	$\overline{b}_i$	$\underline{z}_s$	$\overline{x}_s$	$(1+\gamma)\cdot\beta_s - \overline{b}_i \cdot \gamma$	$\overline{z}_s$
3	$\underline{x}_s$	$\underline{y}_s$	$\overline{c}_i$	$\min(\underline{x}_s + \Delta a_i \cdot (1+\gamma), \overline{x}_s)$	$\min(\underline{y}_s + \Delta b_i \cdot (1+\gamma), \overline{y}_s)$	$\overline{z}_s$

	support	
t	α_t	β_t
1	α_s	β_s
2	α_s	β_s
3	$\min(\overline{a}_i, \alpha_s)$	$\min(\overline{b}_i, \beta_s)$

Table 6.1: The new empty spaces when overhang is allowed.

For the spaces to be valid, we demand that the new generated empty spaces are inside the old empty space. If this was not the case, unstable loads could be constructed. For space 1 and 2 in Table 6.1 this trivially holds - which is elaborated shortly. For space 3 this should be assured, which is done by comparing the maximum of the old and the new overhanging space and choosing the smallest of these. If this was not done, the space above could be expanded to be larger than the old space and unstable arrangements like the one on Figure 6.5 could be constructed.

It can easily be realised that space 1 and 2 in Table 6.1 are inside the old

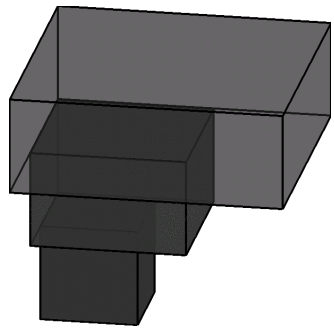


Figure 6.5: An unstable stack.

space. This is true because all dimensions of the boxes are positive, hence, the amount of support from below becomes smaller which makes the new empty space smaller. This is shown in Figure 6.6. The largest rectangle in the figure is the original empty space which is supported from $(\underline{x}_s, \underline{y}_s)$ to (α_s, β_s) indicated with dark grey. In this space a box is placed, indicated by light grey. The new resulting empty spaces, space 1 and space 2 (calculated as shown in Table 6.1), are shown like light grey transparent surfaces. For instance, the supported area in space 2 is given by the area with minimum in $(\underline{x}_s, \overline{b}_i, \underline{z}_s)$ and maximum

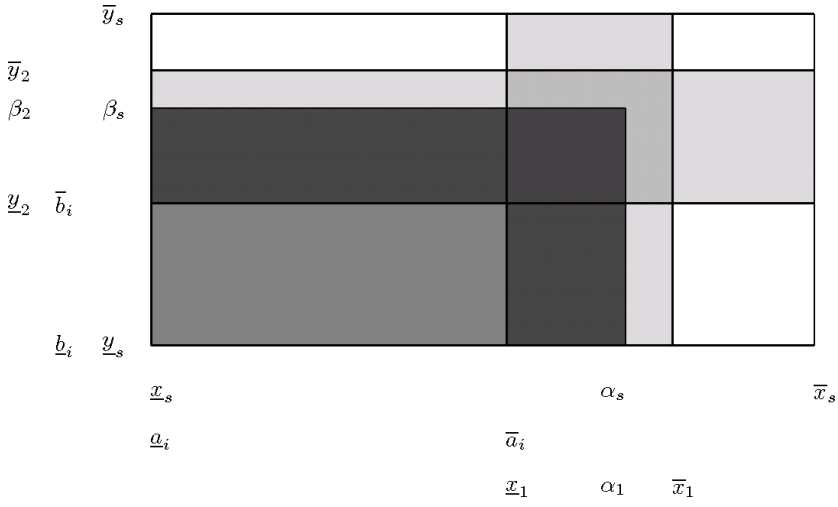


Figure 6.6: The new smaller empty spaces , space 1 and space 2 when a box is placed in a space with overhang.

in $(\overline{x}_s, \overline{y}_2, \overline{z}_s)$. The new space with overhang gets a smaller expansion in the y-dimensions since less support is present. The maximum in the y-dimension $\overline{y}_2$ is given by:

$$\overline{y}_2 = \beta_s + (\beta_s - \overline{b}_i) \cdot \gamma = (1 + \gamma) \cdot \beta_s - \overline{b}_i \cdot \gamma$$

For space 2 nothing changed in the x-dimension compared to the old empty space, since there is still the same support in that dimension.

Similar changes are made to space 1.

In the case when a box is placed so that it is overhanging another box, at least one space is made with no support at all. Lets look at a small example.

Example 6.1 *Given the space with minimum in* $(0, 0, 4)$*, maximum in* $(10, 10, 10)$ *and support parameters* $(\alpha, \beta) = (8, 8)$ *and the box* $(l, w, h) = (9, 5, 3)$*. There is obvious room for the box in the space, where it is placed with the preferred rotation - which is as it is. The new spaces will according to Table 6.1 become:*

	min			***max***			***support***	
t	$\underline{x}_t$	$\underline{y}_t$	$\underline{z}_t$	$\overline{x}_t$	$\overline{y}_t$	$\overline{z}_t$	α_t	β_t
1	*9*	*0*	*4*	*7,75*	*10*	*10*	*8*	*8*
2	*0*	*5*	*4*	*10*	*8,75*	*10*	*8*	*8*
3	*0*	*0*	*7*	*10*	*6,25*	*10*	*8*	*5*

obviously, space 1 is invalid as $\overline{x}_1 < \underline{x}_1$*, therefore space 1 is discarded and not added to the set of new spaces. This is exactly the space that is made where no support exist.*

Also notice, that the space above is limited in the x*-dimension by the space wherein the box is placed. If this was not the case the space above would have had maximum in* $(11, 25; 6, 25; 10)$*, which we do not allow since the new space then would protrude the old one, and arrangements like the one on Figure 6.5 could emerge.*

As can be seen from Example 6.1, empty spaces can be made with negative expansion - because they have no support at all. These spaces are of course not included in the set of new empty spaces, following Definition 5.4, page 36.

Remove overlap with other empty spaces

As when overhang was not allowed, a box can invalidate more empty spaces than the space where it is placed.

Compared to the situation when no overhang was allowed, more new spaces can be made. The four spaces surrounding the placed box must still be made. Besides this a new space under the box can also be made.

The five new empty spaces are given by the expressions in Table 6.2.

	min		
t	$\underline{x}_t$	$\underline{y}_t$	$\underline{z}_t$
1	$\underline{x}_s$	$\underline{y}_s$	$\underline{z}_s$
2	$\overline{a}_i$	$\underline{y}_s$	$\underline{z}_s$
3	$\underline{x}_s$	$\underline{y}_s$	$\underline{z}_s$
4	$\underline{x}_s$	$\overline{b}_i$	$\underline{z}_s$
5	$\underline{x}_s$	$\underline{y}_s$	$\underline{z}_s$

	max		
t	$\overline{x}_t$	$\overline{y}_t$	$\overline{z}_t$
1	$\underline{a}_i$	$\overline{y}_s$	$\overline{z}_s$
2	$\min(\alpha_s \cdot (1+\gamma) - \overline{a}_i \cdot \gamma, \overline{x}_s)$	$\overline{y}_s$	$\overline{z}_s$
3	$\overline{x}_s$	$\underline{b}_i$	$\overline{z}_s$
4	$\overline{x}_s$	$\min(\beta_s \cdot (1+\gamma) - \overline{b}_i \cdot \gamma, \overline{y}_s)$	$\overline{z}_s$
5	$\overline{x}_s$	$\overline{y}_s$	$\underline{c}_i$

	support	
t	α_t	β_t
1	$\min(\alpha_s, \underline{a}_i)$	β_s
2	α_s	β_s
3	α_s	$\min(\beta_s, \underline{b}_i)$
4	α_s	β_s
5	α_s	β_s

Table 6.2: The spaces made in other spaces when inserting a box and reduced support is allowed.

As in section 5.3.2.1, page 39, it can be realised that removing overlaps between the placed box and other spaces can be generalised to contain all spaces. It is seen that space 1 and 2 from Table 6.1 are the same as space 2 and 4 in Table 6.2. Thereby only the space above box i is missing in Table 6.2. This space is constructed in all spaces, where $\underline{i} = \underline{s}$. This is necessary, because two spaces having the same minimum can have different support.

6.2.2.2 Amalgamation

The empty spaces can of course still be amalgamated, but also here changes are needed.

In the space s we define the supported cuboid $\mathcal{S}_s = \{(\underline{x}_s, \underline{y}_s, \underline{z}_s), (\alpha_s, \beta_s, \overline{z}_s)\}$, which is the subset of s where support from below is guaranteed.

Two empty spaces s and t can be amalgamated if:

$$\underline{z}_s = \underline{z}_t \wedge \Big[\Big(\mathcal{S}_s \boxdot_x \mathcal{S}_t \wedge (\beta_s = \underline{y}_t \vee \underline{y}_s = \beta_t)\Big) \vee \Big(\mathcal{S}_s \boxdot_y \mathcal{S}_t \wedge (\alpha_s = \underline{x}_t \vee \underline{x}_s = \alpha_t)\Big)\Big]$$

This tells us that two spaces can be amalgamated if the part of an empty space with support abut on the minimum of another empty space. Besides this the part of the spaces with support should overlap and finally, the empty spaces must have the same floor height.

As in Section 5.3.2.4, 42, amalgamation of empty spaces can be split in two distinct cases: Amalgamation in the x- and in the y-direction. Here only amalgamation in the x-direction will be described further, but similar rules can be applied in the y-direction.

Amalgamate in x-direction

Two empty spaces s and t can be amalgamated in the x-direction if:

$$\underline{z}_s = \underline{z}_t \wedge \Big(\mathcal{S}_s \boxdot_y \mathcal{S}_t \wedge (\alpha_s = \underline{x}_t \vee \underline{x}_s = \alpha_t)\Big)$$

A case where this happens is shown in Figure 6.7. Here the new amalgamated empty space is shown with the blue colour. The new empty space r is given in

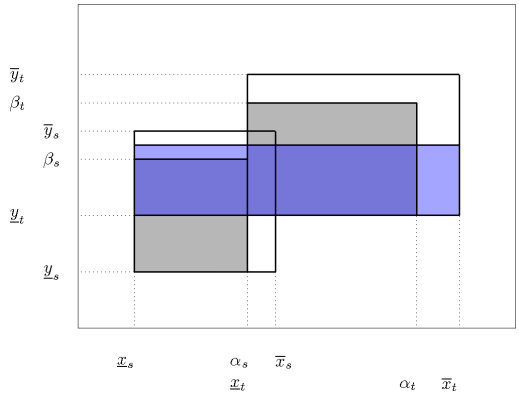


Figure 6.7: New empty spaces when amalgamating two empty spaces with overhang. Amalgamation in the x-direction

Min.	$\underline{x}_r$	=	$\min(\underline{x}_s, \underline{x}_t)$
	$\underline{y}_r$	=	$\max\left(\underline{y}_s, \underline{y}_t\right)$
	$\underline{z}_r$	=	$\underline{z}_s$
Max.	$\overline{x}_r$	=	$\max(\overline{x}_s, \overline{x}_t)$
	$\overline{y}_r$	=	$\beta_r \cdot (1+\gamma) - \underline{y}_r \cdot \gamma$
	$\overline{z}_r$	=	$\min(\overline{z}_s, \overline{z}_t)$
Support	α_r	=	$\max(\alpha_s, \alpha_t)$
	β_r	=	$\min(\beta_s, \beta_t)$

Table 6.3: The resulting empty space after amalgamation in the x-direction.

Table 6.3. Even though the new space r has larger support in the x-direction, it is not expanded beyond the borders of the original empty spaces. This is not done because we do not know if other boxes are placed here. Moreover, we do not know how the support is underneath the boxes which are under the spaces we amalgamate. If we expanded the amalgamated space, we may end up with less support than we desire, as in Figure 6.5.

6.2.2.3 Remove subsets

The definition of subsets is now changed. We have introduced two parameters to control the support from below α, β, which should be reflected.

Definition 6.4 (Space subsets) An empty space s is a *subset* of another space t if it respects Definition 5.8 and:

$$\alpha_s \leq \alpha_t \wedge \beta_s \leq \beta_t$$

This means the support as well as the size of the space s must be a subset of the space t. Note that this definition states the same as Definition 5.8 if $\gamma = 0$ and thereby $\alpha = \overline{x} \wedge \beta = \overline{y}$.

6.2.3 Load bearing strength

The concept of *load bearing strength* (LBS) was first investigated by Bischoff [2]. The basic idea is that all boxes to some extent are fragile. Hence, we need to guarantee that no boxes, are put under more pressure from other boxes, than allowed. Load bearing strength is a measure for how much weight a given box can carry without being damaged.

To model this feature we need to know, for every box $i \in B$ what its load bearing strength is. This is denoted LBS_i. In our model there exist one LBS-value for every box, no matter how the box is rotated. This differs from the work of Bischoff [2] who use three LBS, one for each of the different surfaces.

First we will explain how the load bearing strength constraints are defined if *no* overhang is allowed. In Section 6.2.3.2 this will be extended to include overhang.

6.2.3.1 Load bearing strength without overhanging boxes

To calculate if a given box is load bearing strength feasible, we need to find the sum of the weight of the boxes placed above the given box. These boxes could either be placed directly on top of the given box, or add to the weight through other boxes.

More precisely a box i is considered to be placed on top of box j if:

$$i \boxdot_x j \quad \wedge \quad i \boxdot_y j \quad \wedge \quad \underline{z}_i > \underline{z}_j \tag{6.2}$$

meaning that, they overlap in the x- and y-dimension and i is placed higher than j. Since more than one box can be placed directly on top of the same box[1] it is not enough to find *all* boxes placed above another box and calculate the above boxes joint weight. Instead it is necessary to check for every point on the container floor, if the boxes placed above it are load bearing strength feasible. In practice not every point need to be checked, only one point for every different *box stack*. In Figure 6.8 two box stacks exists. One consisting of boxes 1 and 3, and one consisting of boxes 2 and 3.

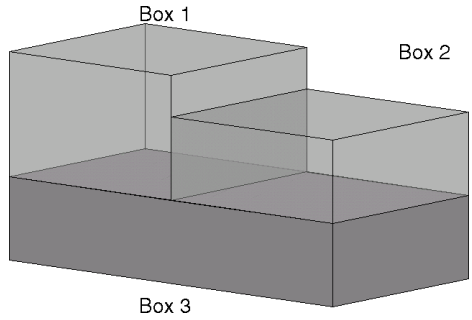


Figure 6.8: Two boxes placed directly on top of the same box.

[1] Box i is placed directly on top of box j if: $i \boxdot_x j \wedge i \boxdot_y j \wedge \underline{z}_i = \overline{z}_j$

Definition 6.5 (Box stack) A box stack bs is bound to a point on the container floor. This point is called the *reference point*, $r = (r_x, r_y, r_z)$, of the box stack. A box stack is composed by all boxes, which overlap the reference point in their x- and y-dimensions. More formally a box i is a member of a box stack bs if:

$$\underline{a}_i < r_x < \overline{a}_i \quad \wedge \quad \underline{b}_i < r_y < \overline{b}_i$$

The set of box stacks is denoted BS.

Notice, that since the box stack is bound to a reference point, two boxes placed in the same height cannot be members of the same box stack. Box 1 and 2 on Figure 6.8 can for instance not be in the same box stack. Now every box i is a member of at least one box stack bs. For every box stack $bs \in BS$, a check should be made, regarding the other boxes in the stacks. The check is:

$$LBS_i \geq \sum_{\substack{j \in bs \\ \underline{c}_i < \underline{c}_j}} \frac{m_j}{\Delta a_j \Delta b_j}, \quad \forall i \in bs \tag{6.3}$$

If this holds for all box stacks $bs \in BS$ the packing is load bearing strength feasible.

6.2.3.2 Load bearing strength with overhanging boxes

When we work with overhang two major problems arise, both illustrated on Figure 6.9.

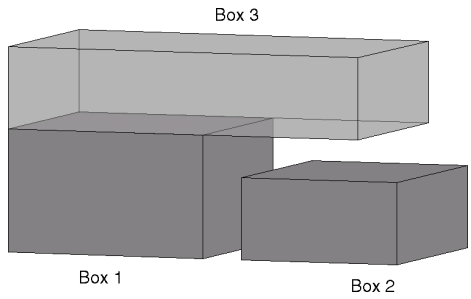


Figure 6.9: One box (box 3) placed overhanging another box (box 1). Under the overhanging part of the box 3, box 2 is placed. There is no contact between box 2 and box 3.

1. Boxes placed above one another (as given in Equation (6.2)), does not necessarily support each other. In Figure 6.9, box 3 is placed above box 2, but there is no contact between them.

2. If a part of a box i does not get support from below, then the boxes that do support it, becomes strained with more weight, proportional to the part of i not supported. In Figure 6.9 box 3 is placed on top of box 1, but it does not have full support, therefore all weight is placed on box 1.

Since boxes can be situated on top of each other without having contact, the concept of box stacks needs to be changed.

Definition 6.6 (Box stack) A box stack bs, with reference point, $r = (r_x, r_y, r_z)$, is composed of all boxes where

$$\underline{a}_i < r_x < \overline{a}_i \ \wedge \ \underline{b}_i < r_y < \overline{b}_i \ \wedge \ \left(\underline{c}_i = 0 \vee (\exists j \in bs, j \neq i | \underline{c}_i = \overline{c}_j)\right)$$

is true.

Definition 6.6 tells us that boxes are only members of a box stack, if they have support in the reference point by either another box in the box stack or the container floor.

When a box is overhanging it means that the supported area is smaller than the base area of the box. Thereby Equation (6.3) is no longer sufficient to describe the weight resting on other boxes. The change, however, is relatively small. Now the supported area S_j^{area} is used instead of the base area. The new load bearing strength check becomes:

$$LBS_i \geq \sum_{\substack{j \in bs \\ \underline{c}_i < \underline{c}_j}} \frac{m_j}{S_j^{area}}, \quad \forall i \in bs \tag{6.4}$$

where S_j^{area} is given as the supported area of box j and $bs \in BS$.

Notice that the changes made to the check introduced when we work with overhanging spaces, work perfectly well when we do not allow overhang. If no boxes overhang others, the supported area always equals the base area of a box. Likewise, when no overhang exists, we are sure that when two boxes are placed on top of each other, there will also be contact. Therefore, the latter described checks, fully replace the earlier described ones. If full support is guaranteed, it should, however, be utilised.

6.3 Summary

In this chapter it is shown how the model introduced in Chapter 5, can be modified to handle the extra complexity needed, when problems from the real world are to be solved. The complexity of the changes needed, depends on which part(s) of the model they affect. To be able to make meaningful loads for Aarstiderne, it was rather simple to change the check done whenever a box was inserted on top of another box. On the other hand it is far more complex to make changes in the way the empty spaces are handled. This has been illustrated by the changes needed to be able to solve the problems from Johannes Fog. The model introduced to the Johannes Fog problems, is more a generalisation, than a specialisation of the realistic model. The multi-drop constraint is relaxed to allow unloading in more directions, it is however, still possible only to allow unloading in one. Instead of only consider fragile boxes, all boxes can be thought of as relatively fragile, using the load bearing strength concept. Finally, also the minimum amount of support from below can be changed.

Besides allowing us to model the specific problems from Aarstiderne and Johannes Fog, the additions to the model can be combined to solve a wide variety of problems met in different industries. An example could be the problem from Aarstiderne, which resembles problems met in the brewery industry. Instead of vegetables it is softdrink and beers which is distributed. The properties of the problems are, however, comparable.

Chapter 7

Algorithms

Now, models have been introduced to describe different realistic container loading problems with multi-drop constraints. This gives us the necessary tools to develop a solution procedure. Our demands to the algorithm is that it should produce solutions fast and respect the introduced constraints. Moreover, it should be kept in mind, that the algorithm must be capable of finding solutions to problems of very different size and nature.

The main algorithm proposed in this chapter, consist of two parts. A tree search framework is constructed to guide the search for good solutions. To evaluate the nodes in the tree search, greedy packings based on different strategies are developed. The basic tree search will provide a structured framework to search through the solution space. To improve the search process, different tweaks are suggested.

First the different greedy methods are described. Afterwards the tree search framework is introduced.

7.1 The greedy procedures

Previously researchers have worked with constructive algorithms that build walls (see e.g. [16, 28]), layers (see e.g. [3, 20]), stacks (see e.g. [3, 13]) or blocks (see e.g. [5, 10]). Of these methods, we choose to disregard building layers. If only one unloading direction is considered, the allowed sequences in each layer, would depend on boxes in other layers. The other methods seen in the literature all have favourable features, and we have chosen to try approaches, which favours both walls, stacks and blocks.

All the greedy methods proposed follow the same framework. The only difference is the strategy for matching boxes and spaces.

The generic framework of the methods is summarised in Algorithm 7.1. The main difference between them is how line 4 is performed.

Algorithm 7.1: GREEDY

```
1: repeat
2:    Choose subset of the not placed boxes B_nPlaced with desired sequence
      number(s)
3:    while More boxes can be placed do
4:       Find feasible box/rotation/space combination
5:       Insert found box in the found space
6:       Update empty spaces and boxes
7:    end while
8: until stop
```

Before entering the while loop, the option to work on a limited number of customers at a time, is given. This can help control the packing sequence more thoroughly. If we for instance choose to load boxes from one customer at a time - starting with the customer visited last and ending with the one visited first - this will indeed be a LIFO (Last-In-First-Out) packing. This follows the ideas introduced by Bischoff and Ratcliff [3] and Gendreau et al. [14] to cater for multi-drop constraints. The minimum number of boxes present in the subset of $B_{nPlaced}$ used when running the inner loop of the greedy algorithms is called κ. The point of splitting the batch in smaller subsets should be not too mix up boxes from different customers. Loading one customer at a time, however, could be too strict.

The algorithm stops if: 1) all boxes are packed, 2) no boxes in the current subset of $B_{nPlaced}$ fit in the left empty spaces or 3) there are no more empty spaces.

7.1.1 Choose the first feasible box/rotation/space combination

The next two methods have very simple strategies in line 4 of Algorithm 7.1. They depend on that the boxes in $B_{nPlaced}$ are sorted according to some rule. We have chosen 3 different, namely:

1. First by sequence then by largest dimension
2. First by sequence then by largest surface
3. First by sequence then by largest volume

The strategies are chosen with focus on placing the largest boxes early, as these are the potentially hard boxes to place. A smaller box will more easily fit in the container later in the loading process. The motivation for not only looking at volume, is that it might be harder to place boxes with one or two very large dimensions, than a big but even sided one.

Also the sequence in which the different feasible box rotations are tried is important. We are considering 6 different sequences. The first one favours the biggest dimension placed along the x-dimension, the middle dimension along the y-dimension and the smallest dimension along the z-dimension. This is denoted XYZ. The second favours the biggest dimension along the x-dimension, the middle one along the z-dimension and the smallest along y-dimension, which is denoted XZY. The same principle make up the last four rotation sequence, which are denoted YXZ, YZX, ZXY and ZYX.

Finally the empty spaces in the set S are also sorted. This is done according to their position in the container, starting with the spaces with low $\underline{x}, \underline{y}$ and $\underline{z}$ in that sequence.

7.1.1.1 Find a box to the chosen space (SB)

The procedure chooses the first space and then tries to find a box that can be placed in it, starting with the most promising box according to the chosen box sort rule.

Algorithm 7.2: FIND BOX/ROTATION/SPACE COMBINATION (SB)

```
1: for all Spaces do
2:   for all Box types do
3:     for all Feasible rotations do
4:       if Box is valid in space then
5:         return Box/rotation/space combination
6:       end if
7:     end for
8:   end for
9: end for
```

If the first chosen box with the first feasible rotation does not fit in the space the next rotation is tried. When no more rotations exist for the box the next box type is tried. If no boxes at all fit in the space the next space is tried starting all over with the boxes. The procedure to find the box and space is shown in Algorithm 7.2. As spaces are sorted by x-dimension first, this procedure favours wall constructions.

7.1.1.2 Find a space to the chosen box (BS)

In the second greedy algorithm all the spaces and boxes are ranked according to the same rules as in the first procedure.

Algorithm 7.3: FIND BOX/ROTATION/SPACE COMBINATION (BS)

```
1: for all Box types do
2:   for all Feasible rotations do
3:     for all Spaces do
4:       if Box is valid in space then
5:         return Box/rotation/space combination
6:       end if
7:     end for
8:   end for
9: end for
```

The difference between the two procedures is that we choose a box, then a rotation and then searches for a space where it can be placed. This tends to favour placements of the same box types in blocks. In Algorithm 7.3 it is shown how this works.

7.1.2 Choosing the most promising box/rotation/space combination

The methods described in this section rank all boxes with all possible rotations against all spaces available. This is done based on some criteria, which we will go into more details with, in the subsequent sections. As we rank all spaces against all boxes, there is no need to sort neither the spaces nor the boxes. However, for each chosen box/rotation/space combination more calculations are needed. This makes them more time-consuming than the first two methods described. The framework to find a box and a space for the next two procedures is shown in Algorithm 7.4.

Algorithm 7.4: FIND BOX/ROTATION/SPACE COMBINATION (ST & VL)

```
1: for all Box types do
2:   for all Feasible rotations do
3:     for all Spaces do
4:       Evaluation the box/rotation/space combination
5:     end for
6:   end for
7: end for
8: return Best box/rotation/space combination
```

7.1.2.1 The Bischoff & Ratcliff multi-drop method (ST)

This method is taken directly from the work of Bischoff and Ratcliff from 1995 [3]. We have chosen to implement it, as it, to the knowledge of the authors, is the only really well described method that considers multi-drop situations reported in the literature.

For every box/rotation/space combination we calculate 4 statistics used to find the best. If the placement of box i with rotation fr is valid in space s, we calculate:

$$K^{*1}_{bt_i,s} = \min\left(\left\lfloor \frac{\Delta z_s}{\Delta c_i} \right\rfloor, k_{bt_i}\right)$$

where $k_{bt_i} \in \mathcal{B}$ is the number of boxes of box type bt_i left yet to be placed. The other part of the minimum expression calculates the maximum number of this box type that is possible to stack in space s. This means that $K^{*1}_{bt_i,s}$ represent the number of boxes of box type bt_i which can be placed in a stack in space s.

Example 7.1 *Lets assume that we have 3 boxes of the same type yet to be packed ($k_{bt_i} = 3$). Furthermore, lets say that the boxes have $\Delta c_i = 4$ in the chosen rotation and the space we are evaluating has height $\Delta z_s = 17$ and in all dimensions can contain the box. Then:* $K^{*1}_{bt_i,s} = \min\left(\left\lfloor \frac{17}{4} \right\rfloor, 3\right) = \min(4, 3) = 3$.

Now the criteria can be formulated as follows:

Main criterion: Largest potential space utilization:

$$u_{i,s} = K^{*1}_{bt_i,s} \cdot \left(\frac{l_i w_i h_i}{\Delta x_s \Delta y_s \Delta z_s} \right)$$

1. **Tie breaker:** Smallest lengthwise protrusion: $p_{i,s} = \underline{x}_s + \Delta a_i$
2. **Tie breaker:** Largest box volume: $v_i = l_i w_i h_i$
3. **Tie breaker:** Lowest value of $\underline{y}_s$

The main criterion favours a box type that gives the best volume utilisation in the space, when we can place $K^{*1}_{bt_i,s}$ of them. The first tie breaker favours box/rotation/space combinations that does not expand in the x-dimension of the container, trying to pack the boxes as close to the back wall as possible. The second tie breaker tries to place big boxes first, as they may be difficult to place later on. The only purpose of the last tie breaker is to make a unique selection when the other rules does not give one. It is these criteria that are used in line 4 of Algorithm 7.4, to evaluate a box/rotation/space combination.

Of course the second tie breaker only concerns a box and hence, does not need to be recalculated for every box/rotation/space combination. The same apply to the third tie breaker, which only concerns the space and hence only need to be evaluated when a new space is looked upon. This does not apply to the main criterion or the first tie breaker as they depend on both the space, the box and the chosen box rotation.

It is important to notice, that the calculation of $K^{*1}_{bt_i,s}$ only gives a limited lookahead in each iteration. This means that after placing one box, there is no guarantee that the same box type will be placed on top of it.

7.1.2.2 Ranking the boxes based on their volume usage in the entire space (VL)

This method is almost the same as the previous, except for the main criterion. Instead of only looking at the volume usage in a stack, we look at the volume usage in the entire space. This can easily be changed by changing the definition of $K^{*1}_{bt_i,s}$ to the number of boxes which can be placed in the space. To reflect the number of boxes that can be placed in the space, we calculate how many boxes can be placed in each direction in the space and multiply these. The new $K^{*2}_{bt_i,s}$ is given by:

$$K^{*2}_{bt_i,s} = \min\left(\left\lfloor\frac{\Delta x_s}{\Delta a_i}\right\rfloor\left\lfloor\frac{\Delta y_s}{\Delta b_i}\right\rfloor\left\lfloor\frac{\Delta z_s}{\Delta c_i}\right\rfloor, k_{bt_i}\right)$$

which replaces $K^{*1}_{bt_i,s}$ in the main criterion. Otherwise, we use exactly the same criteria as in the previous method.

7.1.2.3 Minimise the volume of generated unusable space (EL)

The third method that evaluates the combination of spaces and boxes, seeks to minimise unusable space generated by the insertion of a box. This method is an implementation of Eley's greedy heuristic [10].

Algorithm 7.5: FIND BOX/ROTATION/SPACE COMBINATION (EL)

```
1: foundBox = false
2: for all Box types do
3:    for all Feasible rotations do
4:       for all Spaces do
5:          Evaluate the box/rotation/space combination
6:       end for
7:    end for
8:    if A valid box/rotation/space combination is found then
9:       return Best box/rotation/space combination
10:   end if
11: end for
```

In this method, we actually only look at one box at a time, namely the box not yet packed with largest volume. The box is inserted in as many different ways as there are valid rotations for it in each available empty space. For each insertion, we look at the resulting empty spaces and check whether boxes can be

inserted into them. If we produce spaces that are too small to contain a box, the evaluation of the box insertion is worsened by the volume of the unusable space. If two insertions yield the same evaluation, we favour the box insertion, which is placed with euclidian distance closest to the origin in the container. If both insertions are in the same space, this will also evaluate to the same value, and the insertions are considered equally good. Then the first box/rotation/space combination is chosen. The pseudo code for this insertion method is shown in Algorithm 7.5.

7.2 The improvement framework

As mentioned in the literature review, many different approaches have been tried by others. These approaches rank from tree searches to many different kinds of metaheuristics. The most promising metaheuristics for the CLP are based on either tabu search, GRASP or genetic algorithms. Both tabu search and GRASP need an appropriate neighbourhood. As the complexity of a packing is very large, it is not trivial to remove a box from the container and guarantee that the packing stays feasible. In fact, if this is to be done without a complicated check, the boxes must be removed in the opposite sequence as they were packed. If not, a complete repacking is necessary. What other researchers have done, is to make a greedy solution and use the packing sequence as a representation for the solution. Then a new solution can be found by interchanging two boxes in the packing sequence (see e.g. [14]) or by removing a box temporarily, insert another box, and then making the removed box available to the algorithm again (see e.g. [24]). Other approaches that for instance greedily insert boxes in blocks, make all the block choices at every insertion point, different neighbouring solutions to their problem (see e.g. [5]). The drawback of all these definitions is that the neighbourhoods quickly become very large, which leads to large computation times for both the local search algorithm in GRASP and the neighbourhood search in the tabu search.

With concerns to genetic algorithms, two different approaches have been proposed by Bortfeldt and Gehring [4, 13], one is based on stacks and the other on walls. The disadvantage of the proposed genetic algorithms is that many boxes are moved around, when new solutions are constructed. This makes it hard to maintain multi-drop feasibility.

The last structured improvement method we have seen introduced with success is the tree-search approach, which we have chosen as our algorithm of choice.

7.2.1 Tree search

In order to guide the search for good solutions, a tree search improvement framework is developed. The framework gives the opportunity to test different loading sequences in a constructive way. In the proposed framework each box insertion is a choice in the tree. This follows the ideas from Scheithauer [30] and Eley [10]. Thereby, many different box insertion can be tried after each box is placed, instead of only choosing one greedily. Two different strategies, on how to choose the space wherein a box should be inserted, are tested. Either the space can be chosen by the greedy method, as proposed by Eley, or the space choice can be made part of the tree, as proposed by Scheithauer.

To improve the solution speed an extension of the tree search is proposed, taking advantage of the greedy solutions generated during the procedure. This is called the accelerated tree search. Furthermore, we also propose a dynamic breadth that will adjust the breadth of the search tree in each iteration.

7.2.1.1 The basic framework

The general idea in the tree search is to construct a tree consisting of *partial solutions*, where in each node of the tree, a combination of box, rotation and space is chosen carefully.

A *partial solution* consists of a set of valid empty spaces and two batches, B_{placed} and $B_{nPlaced}$. Additionally in a partial solution, it has not been shown, that no more boxes can be placed. The initial state of the algorithm is an empty container. The empty container is the partial solution, where no boxes are placed at all. This partial solution is the root of the search tree.

In each node of the search tree it is desirable to have both a primal and a dual bound on the solutions. The primal bound is rather easy to obtain, as it can be found as a solution to the problem found by the greedy procedure (Algorithm 7.1). Getting a good (tight) dual bound, on the other hand, is not a trivial task. Examples of dual bounds to the CLP are seen in the literature (see. e.g. [23, 31]), but because of our time perspective and the extra constraints introduces in our problem, it is not considered.

It is not possible to fully branch on all partial solutions, due to the many possibilities there exists when a new box should be inserted. An example of the size of the tree is seen in example 7.2.

Example 7.2 *If we have 20 different box types in our problem and all of them have 6 feasible rotations. The first decision alone leads to 120 possibilities. When the next box is inserted there are* $120^2 = 14400$ *different nodes in the fully expanded tree.*

Because the number of nodes in the fully expanded search tree grows very fast, as illustrated in Example 7.2, and we have a limited amount of time available to run the algorithm, it is not possible to search through all branches in the tree. Instead, we define a *breadth*, λ, of the search tree. At every depth, the tree can at most be λ wide. The choice of λ will of course affect the running time of the algorithm, and if there is a tight upper bound on the time available, it is crucial to choose λ correctly. If λ is chosen too high, the algorithm will terminate, while still being in the top of the tree. This means that many boxes are not placed by the tree search, but only placed greedily. If, on the other hand, λ is chosen too low, the algorithm will simply terminate without spending all time available, and fewer packing combinations will be tried. In either way the quality of the solutions decrease.

The basic framework of the tree search, where the spaces are chosen greedily, is shown in Algorithm 7.6. The algorithm begins with the partial solution

Algorithm 7.6: TREE SEARCH

```
1:  Partial solution := empty container
2:  repeat
3:    for all Partial solutions do
4:      for all Box types do
5:        for all Feasible rotations do
6:          Tentatively insert the box in the container using GREEDY()
7:          Add solution to partial solutions and evaluate by inserting the
            remaining boxes using GREEDY()
8:        end for
9:      end for
10:   end for
11:   Delete identical solutions
12:   Keep best λ partial solutions
13: until optimal solution found or #partial solutions = 0
    or time > timeLimit
```

containing the empty container and a batch of boxes to place. Besides this, only λ, is given. The algorithm is constructed to be able to work with any greedy algorithm, finding a feasible solution to the problem. In each iteration of the procedure, the job is to insert a box in the partial solutions. This is done

by trying to insert all possible box types not yet loaded with all their feasible rotations. The place to insert the box is in the basic approach chosen by the greedy method. This is done in line 6. When a box is inserted into a partial solution, we need to measure the quality of the solution. This is done by calling the greedy procedure with the rest of the not loaded boxes, which is seen in line 7. From the best found greedy solution, information about which boxes are loaded and the volume utilisation, is stored. From all other greedy solutions only the volume utilisations are stored.

Every different insertion gives a new partial solution. From these we first delete solutions that are identical (see Section 7.2.1.2) and then keep the best λ. This gives up to λ new partial solutions, which can be expanded in the same way. When no more boxes can be inserted into a partial solution the partial solution is no longer partial, but full. Therefore it is deleted (of course it is remembered if it the best one yet) to leave room for other partial solutions in which more boxes can be inserted. The algorithm terminates if an optimal solution is found, no more time is available or no more partial solutions exist.

7.2.1.2 Deleting identical solutions

Different packing sequences can lead to the same partial solution because of the multiple symmetry of the container loading problem, meaning that different paths in the tree can lead to identical solutions. This can lead to situations where all stored solutions are replaced by the same good solution. To avoid this, a rule to remove identical partial solutions is used. A similar rule was proposed by Eley [10].

Two partial solutions are considered identical if boxes with the same box type and sequence number are placed at the same position. This can happen even though the boxes are not loaded in the same sequence.

7.2.1.3 The space choice

In the basic framework, the empty space, in which a box is inserted, is chosen by the greedy procedure. This follows the concept of the tree search proposed by Eley. Instead of leaving this choice to the greedy procedure it could be a good idea to have the space choice as a part of the branching scheme. This means that a box insertion is now tried for every different box type, with all feasible rotations in *all valid empty spaces*. This follows the ideas of Scheithauer. This means that an extra loop, running over empty spaces should be inserted

in the inner loop of Algorithm 7.6. Thereby the space choice in line 6 becomes faster, because this choice is already made, however more partial solutions are generated, which means that more greedy solutions should be made. Therefore, the overall complexity of the algorithm increases, when insertion in all spaces is tried.

7.2.2 The accelerated tree search

In the basic framework one box is placed in the container in each iteration, given by the loop in line 2-13 (Algorithm 7.6).

When a greedy solution is found, all information except the volume utilisation, is thrown away. The accelerated tree search tries to analyse the greedy solution, and possibly use a part of it in the partial solution.

Algorithm 7.7: ACCELERATED TREE SEARCH

```
1:  Partial solution := empty container
2:  repeat
3:    for all Partial solutions do
4:      for all Box types do
5:        for all Feasible rotations do
6:          Tentatively insert the box in the container using GREEDY()
7:          Add solution to partial solutions and evaluate by inserting the
            remaining boxes using GREEDY()
8:        end for
9:      end for
10:   end for
11:   Delete identical solutions
12:   Keep best λ partial solutions
13:   Extend partial solutions by selected part of ancillary greedy solutions,
      see Algorithm 7.8
14: until optimal solution found or #partial solutions = 0 or time > timeLimit
```

The accelerated tree search can be seen in Algorithm 7.7. The only difference between this and Algorithm 7.6, is the addition of line 13. In this line the partial solutions are extended to contain a part of the greedy solutions, that were found during the evaluation of the partial solutions in line 7. Pseudo code explaining this extension, is shown in Algorithm 7.8. GS is the greedy solution found based on the partial solution PS.

A greedy solution is analysed by looking at the empty spaces left in the container,

Algorithm 7.8: Extension of partial solution

1: $PS :=$ partial solution
2: $GS :=$ ancillary greedy solution
3: $s_{old} :=$ space found in last iteration
4: **for all** empty spaces in GS **do**
5: locate empty space furthest back, s_{back} later then s_{old}
6: **end for**
7: **for all** boxes in $GS \setminus PS$ **do**
8: locate first box i_{first} placed after s_{back}
9: **end for**
10: In $GS \setminus PS$ remove i_{first} and all boxes placed after this and call the resulting set EPS (Extension of Partial Solution)
11: $PS := PS \cup EPS$

when the greedy algorithm has terminated. The empty spaces can be seen as an indication of where the boxes could have been packed better. The idea is to locate the empty space furthest back in the container, and remove all boxes placed after this space. To be able to compare the placements of the spaces, we must define when a point is further back in the container than another.

Definition 7.1 (Point placed further back) A point $p = (x_p, y_p, z_p)$ is placed *further back* than a point $q = (x_q, y_q, z_q)$ according to the following criteria:

Main criterion: $x_p < x_q$

Tie breaker 1: $y_p < y_q$

Tie breaker 2: $z_p < z_q$

Now, spaces and boxes can be compared to see which is further back. This is done by comparing their minimum based on Definition 7.1. If minimum for two spaces are the same, they are considered equally far back in the container and one is arbitrarily chosen.

By removing the first box placed after the space chosen, and all boxes placed later than this by the greedy algorithm, we assure that no boxes are placed after the empty space selected, thereby giving the tree search a possibility to improve the packing in that area. Boxes placed later than the first box chosen, could very well be placed further back than the space this box was compared with. An idea could be to leave these boxes in the container, this would however break the loading sequence, thus removing the guarantee of a feasible load.

The reason to exclude boxes already included in the partial solution when finding the first box to remove (line 7 in Algorithm 7.8), is to assure the convergence of the solution. At the same time it is necessary to keep track of earlier chosen spaces in line 8 to assure always to select a space further out in the container, otherwise if we cannot fill out a space, we would repeatedly find it and then only pack one box at a time, missing the point of the accelerated tree search.

The effect of the accelerated tree search, is that the choice of branching in the algorithm has been changed. Instead of branching on every box and rotation, the branching is reduced to be on the first box placed after the next empty space. An effect of the strategy is that there is no guarantee that the nodes chosen in an iteration will have the same number of boxes loaded in the search tree. But because of the way the solution evaluation criterion is constructed, they can still be compared.

All in all the algorithm will reach the leaf nodes fast when the greedy solutions are tightly packed, and more slowly when there are many empty spaces. Worst performance for the accelerated tree search is the case where branching is done on all boxes and rotations. Then the number of branches equals the number of branches made by the basic tree search.

7.2.3 The dynamic breadth

As mentioned, when the static breadth was introduced, the choice of λ is crucial for the performance. A reasonable choice depends on both the problem at hand and the time available to solve it. Partly because of this and partly because it could be a good idea to change the breadth throughout the search, we introduce a dynamic breadth.

The dynamic breadth is calculated on the fly, based on the remaining time, Δt, and the estimated time necessary to get to the bottom of the search tree through one branch t_{branch}. This leads to the following value of the breadth in iteration p:

$$\lambda_p = \frac{\Delta t}{t_{branch,p}} \cdot \mathcal{W} \tag{7.1}$$

where $\mathcal{W}$ is a weight which will be explained further in Section 7.2.3.1. The time used in one branch of the tree is not trivially estimated, as it depends on many factors varying independently. The most important of these are the number of different boxes possible to place, the possible rotations of these, the greedy method used, the number of spaces in the container and the depth of the branch. All these factors cannot be taken into account and instead a simplified estimate is used.

The time t_{PS}, it takes to expand one partial solution PS in the tree is estimated to be linear dependent on the number of not yet loaded boxes $|B_{nPlaced,PS}|$, and given by:

$$t_{PS} = K \cdot |B_{nPlaced,PS}| \tag{7.2}$$

Here, both t_{PS}, as well as $|B_{nPlaced,PS}|$, are known. To expand a partial solution all feasible combinations of box types and rotations (if the space choice is greedy) are evaluated by the greedy method. t_{PS} and $|B_{nPlaced,PS}|$ are estimated based on the average time used to develop all the possible solutions and the average number of boxes not loaded, in each node. In the not accelerated algorithm $|B_{nPlaced,PS}|$ is the same for all partial solutions and equals the total number of boxes minus the depth of the tree. If the accelerated algorithm is used, a different number of not loaded boxes exists in every partial solution. The expansion time varies between the different partial solutions, mainly because different boxes are available. Based on t_{PS} and $|B_{nPlaced,PS}|$, K can be found. To make a full solution, we need to place many boxes. The best guess on how many boxes should be placed, is the number of boxes placed in the best found solution so far. The number of not placed boxes in the best know solution is denoted $|B^*_{nPlaced}|$. The total time $t_{branch,PS}$ needed to get from a partial solution PS to the leafs in the tree, can be found as the integral:

$$t_{branch,PS} = \int_{|B^*_{nPlaced}|}^{|B_{nPlaced,PS}|} K \cdot x \, dx \tag{7.3}$$

Thereby the breadth λ_p in each iteration is estimated as:

$$\lambda_p = \left\lfloor \frac{2\Delta t \cdot |B_{nPlaced,PS}|}{t_{PS}(|B_{nPlaced,PS}|^2 - |B^*_{nPlaced}|^2)} \cdot \mathcal{W} \right\rfloor \tag{7.4}$$

If $|B_{nPlaced,PS}| < |B^*_{nPlaced}|$ then the denominator in Equation (7.4) is set to t_{PS}.

The new breadth λ_p is sensitive to changes in time measure t_{PS} and fluctuations in this time measure could change the breadth unwanted. Hence, it is chosen that λ_p can change with maximum 20% compared to λ_{p-1}. If the breadth is too small to change at all (for instance $\lambda_{p-1} = 2$) then it is allowed to change the breadth with ± 1.

7.2.3.1 Changing the weight used in the dynamic breadth

When building the search tree, it is not necessarily best to have the same breadth in the start and the end. When only a few boxes are in the container, it is harder to estimate which choice of placement for the next box is best. Therefore a tree that narrows in along the way could lead to better solutions than an equally weighted tree. To test this, a weight $\mathcal{W}$ is introduced in Equation 7.1. It is found as:

$$\mathcal{W} = \left(2 - \frac{|B^{PS}_{placed}|}{|B^{*}_{placed}|}\right)^{\psi} \tag{7.5}$$

where $\psi \in \mathbb{R}$. If $\psi = 1$ the weight $\mathcal{W}$ will in the beginning, be close to 2 and the breadth is thereby twice as wide as the estimated breadth. In the end the weight is close to 1, and not influencing the estimated breadth. By changing ψ the impact of the weight can be adjusted.

7.3 Summary

It has been a goal to develop a general method to solve the CLP with multi-drop constraints. This is reflected in the choice of solution methods. A tree search framework is proposed where every box placement is chosen carefully. This framework provides a general scheme, which can be used to search the solution space. Greedy methods, which favours different box arrangements, are also proposed to evaluate the different choices taken in the tree search.

A number of tweaks are suggested to the tree search. Accelerating the search could be useful when a very short time limit is given. By proposing a dynamic breadth the method uses the available time efficiently independent of the problem at hand. Whether the spaces should be chosen greedily or not, if compact placed block should be left in the partial solutions and if a dynamic chosen breadth will improve the search, should all be investigated further.

Chapter 8

Implementation

The algorithms were implemented in C++ with use of the STL-library. The algorithms were developed in Visual C++ and analysed with the performance profiler Visual Quantify to assure a high-speed implementation. All tests, however, were carried out on the Linux grid, available on IMM, DTU.

A number of different classes are introduced to model the problem. Figure 8.1 shows the conceptual data model for the implemented program.

For each class only the most fundamental attributes and member functions are shown. The basic class is the `Dimension` class. This is used to handle three dimensional values. This could either be coordinates or sizes of boxes or spaces. The member functions of this class can handle relations and basic computations in three dimensions. Both the `Box`, `Space` and `Container` class depend on this to model the placement and size of the different objects.

The `Box` class contains all attributes defining a box, besides the size of the box this is, for example, the sequence and the feasible rotations. When a box is placed in the container, the assigned placement and rotation is also kept as attributes of the box. To manage and manipulate the boxes, a `Batch` class is created. The batch holds a vector of boxes, and the member functions evaluates the boxes in the batch, either all boxes such as the `boxSort()`, that sorts the boxes by a chosen criterion, or a single box such as `placeBox`, that sets

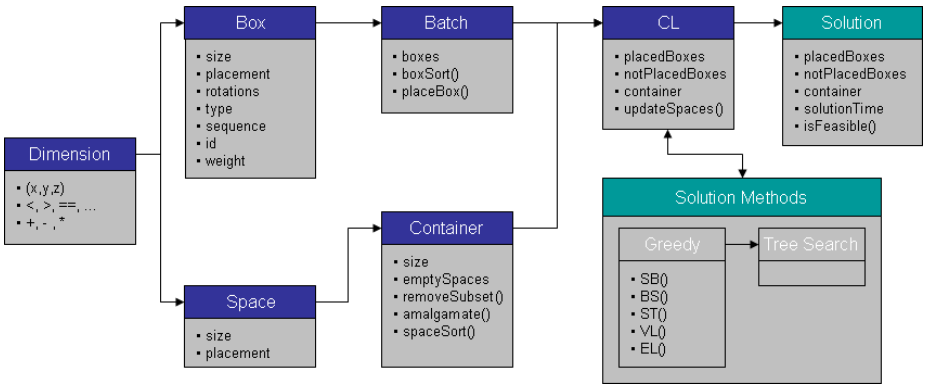


Figure 8.1: Conceptual Data Model for the program. Only the most important attributes and methods are included.

placement and rotation of the box.

The `Space` class is a small class in the program, capturing the attributes of a space, i.e. placement and size. The `Container` class contains all information regarding the container, most importantly its size and the list of empty spaces defining where boxes can be placed in the container. The methods that evaluates the vector of empty spaces, such as `removeSubset()` and `amalgamate()` are also found in the `Container` class.

All information regarding the packing problem at hand, is gathered in the `CL` (Container Loading) class. It contains a batch with the boxes placed in the container, and a batch with the boxes not placed. Furthermore, it contains the container. All methods in the class depend on information regarding both boxes and spaces. When for example a box is placed, the spaces in the container must be updated accordingly. This is done by the method `updateEmptySpaces()`. The `CL` class works in close connection with the `Solution Methods` that contains all the methods used to solve the container loading problem. This is the different greedy methods and the tree search. The greedy methods can either be called directly from `CL` or through the tree search method.

The `Solution` class keeps track of the current best solutions and it contains methods that can check if a solution is feasible. This was very useful when debugging.

Chapter 9

Data

Data from different sources has been used to test the algorithms. We have used data from the OR-Library to make quantitative tests, as many problems are available from this source. This also gives the opportunity to benchmark against other researchers. However, these problems have some limitations, and do not always illustrate the problems faced when considering industrial applications. Therefore, we also include data supplied by the two companies we have worked with, namely Aarstiderne and Johannes Fog.

In the following sections we will go through the characteristics of the data supplied by the different sources.

9.1 OR-Library

From the OR-Library a number of different datasets are available. Seven different files which supply data for the single container loading problem (BR1-BR7) are considered. We will call each file a *data group*. The seven data groups, were first used by Bischoff and Ratcliff in 1995 [3]. In each of these, there are 100 problems. In the different data groups, problems with different number of box types appear. The number of box types in the different data groups are 3(BR1), 5(BR2), 8(BR3), 10(BR4), 12(BR5), 15(BR6) and 20(BR7).

Each problem is generated using a random box generator. First the different box types are generated. All dimensions of the box types are within a defined minimum and maximum and only rotations considered stable, based on the ratio between the dimension, are allowed. When the box types have been generated, one box is generated at a time, chosen randomly from the different box types, until the total volume of the boxes exceed the volume of the container, the last generated box is not part of the problem. Therefore, the total volume of the boxes in a problem is slightly smaller than the volume of the container. In practice, however, this does not mean that all boxes can be loaded. For further details on how the test problems have been generated, we refer to Bischoff and Ratcliff [3].

For the problems, the dimensions as well as rotation restrictions on the boxes are given. However no multi-drop consideration have been made when the problems were generated. To be able to perform a meaningful test of the proposed algorithms, which are designed to solve problems with multi-drop constraints, we need to introduce sequence numbers on boxes. The number of different customers in a problem is denoted nC. To introduce customers to the problems we take a random integer between 1 and nC and leave that number as the sequence number. This is done for each box in the problem.[1]

When solving problems from the OR-Library we always demand full support ($\gamma = 0$) and whenever more customers are present ($nC > 1$), the boxes can only be unloaded in the X^+ direction. Load bearing strength is not considered.

9.2 Aarstiderne

The data from Aarstiderne has been supplied by Transvision. Data from routes already driven, is stored by Transvision, and this is the data we have had access to. This means that we beforehand know that the boxes assigned to a truck can be feasibly loaded. Otherwise the route could not have been driven. It is, however, still interesting to use our method to solve the packing problems, as it will show whether the algorithm can handle the challenges of a real life problem. So far Transvision has used a simple one dimensional measure to assure that the capacity has not been exceeded. One could expect that this is done very conservatively, as the measure poorly reflect the actual capacity.

In the problems considered, Aarstiderne operates with about 10 different box types with standard dimensions. All the boxes which contain vegetables and fruit have the same base dimension. They are $28cm \times 38cm$ and come in two

[1] The random number generator is seeded by the seed given in the problems in BR1-BR7.

different heights. The boxes with fresh meat, fish and groceries are packed in different boxes. These boxes have not got the same base dimension among each other or compared to the vegetable and fruit boxes. This means that they can not be stacked in the same stacks as the vegetable and fruit boxes.

Aarstiderne use many different types of trucks to deliver the goods. It is, in our model, possible to distinguish the size of the containers in which the goods are loaded, and thereby the truck type. However, in the data delivered by Transvision no differentiation of the size is made. Therefore, we have chosen a standard delivery truck for all problems. This is a truck with a $10.3m^3$ container with dimensions $(L, W, H) = (300, 180, 190)$. The boxes are only allowed to be unloaded in the X^+ direction. Of course, all constraints described in Section 6.1 must be respected.

9.3 Johannes Fog

As earlier mentioned Johannes Fog distributes construction products from a depot to different construction sites. When an order is given, it is not known whether or how the products are bundled together. If the order is 126 pieces of $3400mm \times 150mm \times 10mm$ wooden planks, there are many different ways to pack these. The general rule used in the business, is that the packages should be between 1 and 1,2 metres wide. This is done because it makes it easier to load different boxes on top of each other. The fact that Johannes Fog has no exact measure of how large their packages are, when they are loaded onto the vehicles, makes it very difficult to get the necessary information about the consignments.

To be able to work with loads from Johannes Fog, we went to their depot in Lyngby and measured the dimensions of all the boxes in a number of consignments. Along with our own notes we also got the order sheets for each routes. Besides information about each product, the order sheets contain information about the destination of the orders and the accumulated weight. An example of an order sheet is found in Appendix B. In the planning department of Johannes Fog, only the information on the order sheets are used to make both routes and consignments. This is done by hand.

Figure 9.1: Cargo consignments from Johannes Fog

In Figure 9.1 two different consignments are shown. As it can be seen, the sizes of the boxes loaded are very different. Both very large pallets of rockwool (the yellow plastic bag on the right picture in Figure 9.1) and small cardboard boxes with tools are distributed.

Apart from the dimensions of the boxes, it is necessary to get information about what can be put on top of each other. These rules were made in co-operation with Johannes Fog. First of all, only rockwool can be put on top of rockwool. Also, nothing can be put on top of the very odd-sized objects, such as the cylinders placed vertically on the pallet in the middle of the right picture. This also counts for gravel and similar products, which are distributed in sacks. On the other hand, both wooden plank and gypsum boards can be put on top of each other. However, the very heavy gravel sacks and concrete are allowed on top of neither gypsum nor wood. The exact information on the different consignments can be found in Appendix C. This information is reflected in the calculated load bearing strength values.

In feasible packing at Johannes Fog boxes are allowed to overhang each other. To reflect the properties of the problems we choose $\gamma = 0, 2$. Moreover unloading can be done in the directions X^+, Y^- and Y^+.

Chapter 10

Tuning the algorithms

In Chapter 7, an algorithmic framework to solve the CLP with multi-drop constraints, was proposed. A tuning of the algorithms should now be done. First, we will find the greedy methods which are best suited to solve the problem. Afterwards, the best greedy algorithms are used as a part of the tree search framework. Both considerations on the solution quality and time usage are included in the evaluation of the methods.

All the tests are performed on data from the OR-Library. This is done, because it is the only data source - available to us - where enough data exists. In the test of the greedy methods all 700 problems are used, whereas only the first 10 from each data group are included when tuning the tree search. The algorithms are tuned to problems, that can be solved by the realistic model. To reflect this, OR-Library problems with customers added are used.

All the tests are performed on the grid available at IMM, DTU. The grid presently consist of 30 linux machines with 2,7 GHz 64 bit AMD dual core processors and 2 GB RAM, and four high memory Linux machines with eight 2,7 GHz 64 bit AMD processors and 32 GB RAM. To make the quantitative analysis SAS 9.1 was used.

10.1 Best settings for the greedy algorithm

The greedy algorithms are characterised by making fast solutions to the problem, and distinguish themselves by different strategies on how to make these. In this section, the best settings for the greedy algorithms, will be found.

The greedy methods were introduced in the following sections 7.1.1.1 (SB), 7.1.1.2 (BS), 7.1.2.1 (ST), 7.1.2.2 (VL) and 7.1.2.3 (EL).

Some of the methods, namely SB, BS and EL depend on the sequence in which the boxes are tried inserted in the container. This is because they, when a box placement is evaluated, have a *first fit* principle. When the first valid placement in the container has been found, the box is inserted. The other methods (ST and VL) try all different box types with all possible rotations in all possible spaces before a combination is chosen. These methods will also be denoted *best fit* methods. When a first fit method is used, the sequence of how the boxes are tried will have an effect on the final solution quality.

Before we complicate the problems by adding customers, it is decided to find the best choices for both rotation and sorting of the boxes in the first fit methods SB, BS and EL. We need to test, in which sequence the different rotations, for a given box, should be tried and how the boxes are sorted. The different sort strategies are: 1) largest dimension, 2) largest surface or 3) largest volume. The box rotation strategies are XYZ, XZY, YXZ, YZX, ZXY and ZYX, as explained in Section 7.1.1, page 71.

10.1.1 Rotation of boxes

As mentioned in Section 5.2 we work with six different ways to rotate a box. When we use the methods BS, SB and EL, the choice of the sequence in which the different possible rotations are tried, will affect the solution quality. The six different rotation sequence strategies, presented in Section 7.1.1, are tested. The results are shown in Table 10.1. The volume utilisations reported are calculated based on tests over 700 problems, with the 3 different sort strategies possible for the boxes, which means that each number is an average over 2100 different solutions. From the table it is clear, that on average the best rotation sequence is YXZ. This is true for all three greedy methods. It is also the case that not much can be gained when comparing YXZ with the other box rotation sequences.

The YXZ strategy favours box placements where the largest dimension is along

Method	Volume utilisation					
	XYZ	XZY	YXZ	YZX	ZXY	ZYX
SB	81.41	80.88	82.10	81.37	80.82	80.74
BS	81.65	81.48	82.16	81.77	81.28	81.03
EL	79.74	79.72	80.22	80.21	79.59	79.80
Overall	80.93	80.69	81.49	81.12	80.57	80.52

Table 10.1: The influence of different box rotation sequences for the different first fit methods.

the back of the container, thereby protruding least possible into the remaining container floor. Moreover, having the smallest box dimension in the z-dimension, generally allows for a larger area to support other boxes.

A more detailed look at the different box rotation sequences compared with both batch sort strategy and greedy method does not deviate from what is shown in Table 10.1. This is shown in Figure D.1 in Appendix D.

In all succeeding tests, the rotation strategy YXZ is used. This strategy favours placing the biggest dimension along the y-dimension, the middle dimension along the x-dimension and the smallest dimension along the z-dimension.

10.1.2 Sorting the batch

For the first fit methods, it is also important how the batch is sorted. Different sort strategies are described in section 7.1.1.

In Figure 10.1 the results of the test are shown. On the horizontal axis the different data groups from the OR-Library are shown, and the volume utilisation is on the vertical axis. The different graphs in the figure correspond to the different greedy algorithms SB, BS and EL. On each graph the three different sort strategies are shown. For all three methods there is no big difference between sorting by the surface or the volume. On the other hand if one sorts by the biggest dimension, a markedly worse solution is obtained for the problems in data groups 1-3, whereas the average solution values seem to be about the same for the problems in data groups 4-7. For methods SB and BS there even seems to be a tendency that sorting by the biggest dimension gives better results for the more heterogeneous data groups.

When considering the sorting strategies, we thought it most likely, that the volume was the best measure to describe when a box is hard to place. The results

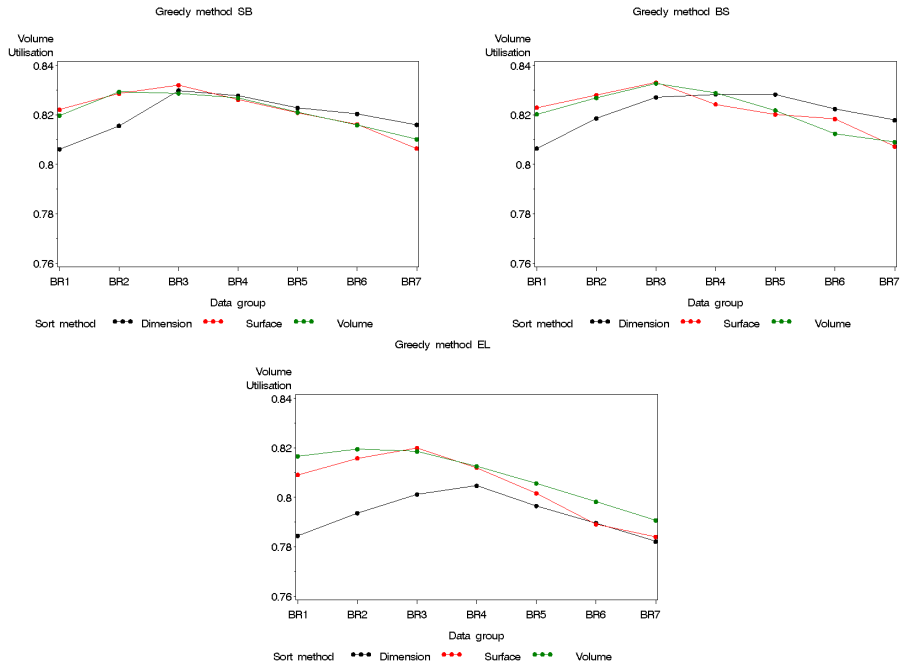


Figure 10.1: The influence of different batch sort strategies for the different first fit methods.

of the test is somewhat surprising, as it shows that sorting by biggest dimension, should be preferred when more heterogeneous data sets are considered. However, the sort strategy that yields the overall best results is sorting by the largest volume. The precise results are shown in Table 10.2. Because a generally good solution method is preferred we choose to sort the batches by the largest volume.

Method	**Volume utilisation**		
	Dimension	Surface	Volume
SB	81.98	82.17	82.16
BS	82.12	82.20	82.17
EL	79.32	80.45	80.88
Overall	81.14	81.61	81.74

Table 10.2: Aggregated results when different batch sort strategies are used.

10.1.3 Several customers in the problems

Now we introduce several customers in the problems. We still use the problems from OR-Library and choose four categories with "several customers", namely $nC \in \{2, 5, 10, 50\}$.

In all the greedy procedures, it is a possibility to load the not already placed boxes in subsets, only considering one subset at a time. This is controlled by the parameter κ. We have tested with $\kappa \in \{1, 5, 10, 25, 50, 100, \infty\}$.

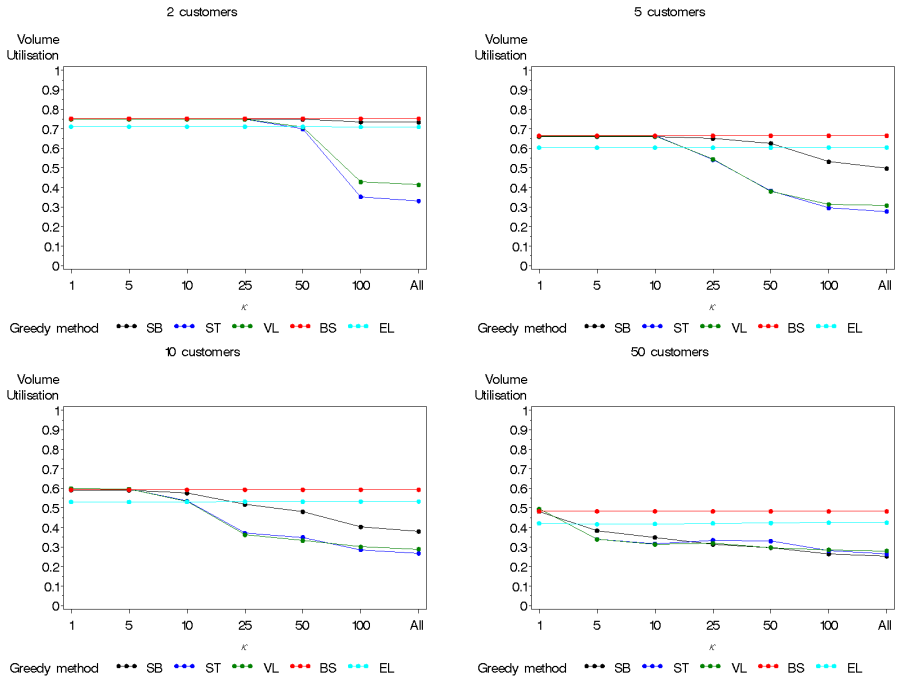


Figure 10.2: The effect of adjusting κ when having more customers.

In Figure 10.2 the result of the test is shown. Four graphs are visible in the figure. Each graph is created based on problems with a different number of customers. On the horizontal axis the minimum number of boxes that should be present in each iteration, κ, and on the vertical axis the average volume utilisation, is shown. The different lines on the graphs indicate the different methods used. There is a clear tendency showing that, when κ increases the solution value decreases. This is true whatever the number of customers is. However, it is also visible that methods BS and EL do not depend on κ. The

reason for this is, that these methods take one box at a time, and tries to place it. Since the sequence in which the boxes are tried does not change, the results are still the same. For SB, also a first fit algorithm, the sequence in which the boxes are loaded does depend on κ. The method tries to find a box to place in a given space, and when more boxes are present at the same time, some boxes with lower sequence number may be loaded in the container early on, leaving less space where boxes can be placed because of the multi-drop constraints.

The graphs in the figure clearly shows, that the methods perform best when κ is small. There may, however, be more book-keeping when a smaller amount of boxes is tried at a time, hence we must also investigate how the solution times depend on κ. This is done in the next section.

By comparing the four graph in Figure 10.2 it is seen, that the overall solution quality decreases with the number of customers in the problem. For $nC = 2$ the solutions are in the interval from 0,7 to 0,75, whereas when $nC = 50$ the solutions are in the interval from 0,45 to 0,5.

10.1.4 Time usage by the greedy methods

It is important to know how much time is used by the proposed algorithms. Some of the algorithms evaluate more carefully each time a box is inserted than others. Hence, the time usages should show this.

In Figure 10.3, it is shown how the average time usage depends on the number of boxes loaded in the problem. In the graph to the left, it is seen that method

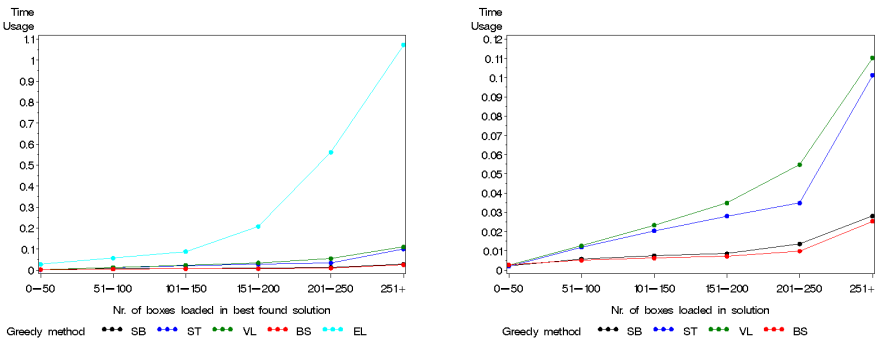


Figure 10.3: Time used depending on how many boxes are loaded. The right graph is without method EL.

EL generally uses much more time than the rest of the greedy algorithms. It becomes especially bad for problems with many boxes, indicating that method EL is less scalable than the other methods. On the graph to the right the same times are shown, but without method EL, as the presence of this method makes it hard to conclude anything about the other methods. On the right graph it can be seen that there is a clear connection between the number of boxes found in a solution and the solution time. The time seems to increase exponentially with the number of boxes. It is also seen that the methods group in two. One group consisting of the methods ST and VL and one group consisting of SB and BS. The methods in the first group are the best fit methods, which try all different box types with all possible rotations in all empty spaces, and based on some criteria, choose a placement for a box. The second group consist of the first fit methods, which picks the first feasible place for an insertion. Therefore, the division of the methods in these two groups does not come as a surprise. That method EL is significantly more time consuming than the other methods is to some extent surprising. The reason for the bad performance, is that whenever a box insertion is tried, the resulting empty spaces need to be found, to be able to evaluate that particular insertion. This is done for every feasible box rotation in every feasible empty space. Because method EL is much more time consuming than any of the other methods, and it never on average performs best, as depicted on Figure 10.2, it is excluded from further tests.

10.1.4.1 The time usage's dependency on κ

In Figure 10.4, the connection between the time usage and κ is shown. It can

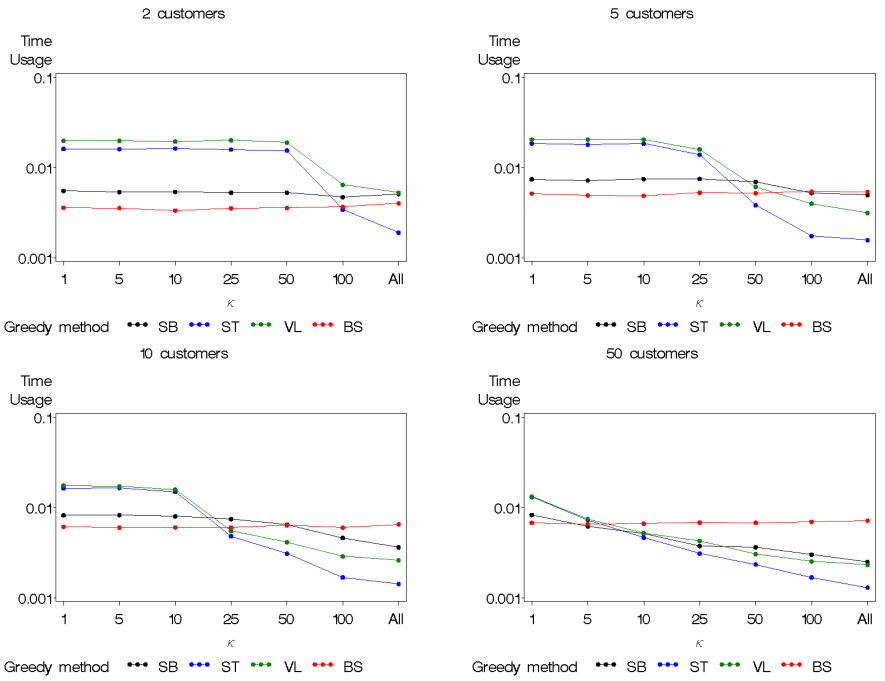


Figure 10.4: The time dependency of having more customers and adjusting the minimum number of boxes available to the algorithms.

be seen, that the time usage of methods ST and VL decrease when κ increases. This should be seen in relation with the results from Figure 10.2. The time decrease is partly because the solution quality also decreases and therefore less boxes are packed. And as we saw on Figure 10.3 there is a clear connection between how many boxes are loaded in a problem and the time used.

In Section 10.1.3, it was concluded that $\kappa = 1$ gives the best results, and should be chosen, if only solution quality was considered. Figure 10.4 shows, that solution time is longer when $\kappa = 1$, but it is still preferred on basis of the much better results.

10.1.5 The best settings for the greedy algorithms

To sum up the findings for the best settings of the greedy algorithms, the following has been found:

Box rotation strategy: YXZ

Batch sort strategy: Largest volume

Minimum number of boxes when loading (κ): 1

Furthermore, it is decided that method EL should not be tested any further, as there are always better and faster alternatives. For the remaining greedy methods (SB, ST, VL and BS), Table 10.3 shows the average results over the problems from the OR-Library with 1, 2, 5, 10 and 50 customers. From the table

Method	Min.	Avg.	Max.	SD.
SB	27,17	66,11	91,46	13,87
ST	26,98	66,82	92,42	13,60
VL	26,98	66,73	93,05	13,53
BS	29,55	66,32	91,53	13,69

Table 10.3: Results obtained with the greedy methods.

it is seen that there is a little difference between the results obtained by the first fit and the best fit algorithms. The best fit algorithms perform slightly better than the first fit. This should be seen in relation to the time usage, as shown in Figure 10.3, where the first fit methods are faster. Both the time and the solution quality of the greedy methods will have influence on the performance of an improvement method. Therefore, we choose to keep the remaining four greedy methods and test them in combination with the suggested improvement framework.

10.2 Tuning tree search

Based on the settings found for the greedy algorithm, we will now tune the tree search algorithm. The improvement algorithm is to be used as an integrated part of a heuristic to the vehicle routing problem and it is therefore important to find solutions fast. We choose two different time perspectives. A short one on 10 seconds and a long one on 60 seconds.

There are a number of different parameters to tune in the tree search framework introduced in Section 7.2. These are listed below:

1. The effect of accelerating the search should be tested. Most likely, it is better to choose the accelerated version when less time is available.
2. The value of evaluating all empty spaces when inserting a box should be compared to making this choice greedily. Again, this choice could very well depend on the time available. If the choice is made greedily, the same greedy method is used to both find the empty spaces and generate the rest of the solution.
3. The choice of breadth should be evaluated:
   (a) First static breadth is considered. The breadth is constant throughout the search.
   (b) Secondly dynamic breadth is considered. The algorithm decides the breadth, based on the amount of time left.

When tuning tree search, we have chosen to use the first ten problems from each of the seven BR data groups. To test the algorithms ability to solve problems with more customers, we use the possibility to incorporate more customers in the problems. We have used $nC \in \{1, 2, 5, 10, 50\}$. First, we will examine the results that can be obtained, when using a static breadth. In the subsequent section the dynamic breadth strategy is tested.

10.2.1 Static tree breadth

In the first test we use a static breadth. This is done to evaluate the influence of the breadth, both on the solution quality, but also on the time used and how it depends on the reached depth in the tree. The different tested breadths are $\lambda \in \{1, 2, 5, 10, 100\}$. Along with the breadth, we have tried settings to test both accelerated search and not, and if the space choice in the tree search should be made greedy or not. Finally there are 4 different greedy methods, which can be applied in the tree search. This gives a total of 28.000 test runs.

10.2.1.1 Overall performance

In Figure 10.5 the overall results, when a time limit of 10 seconds is applied, are shown. The results cover a very wide aspect of different problem types -

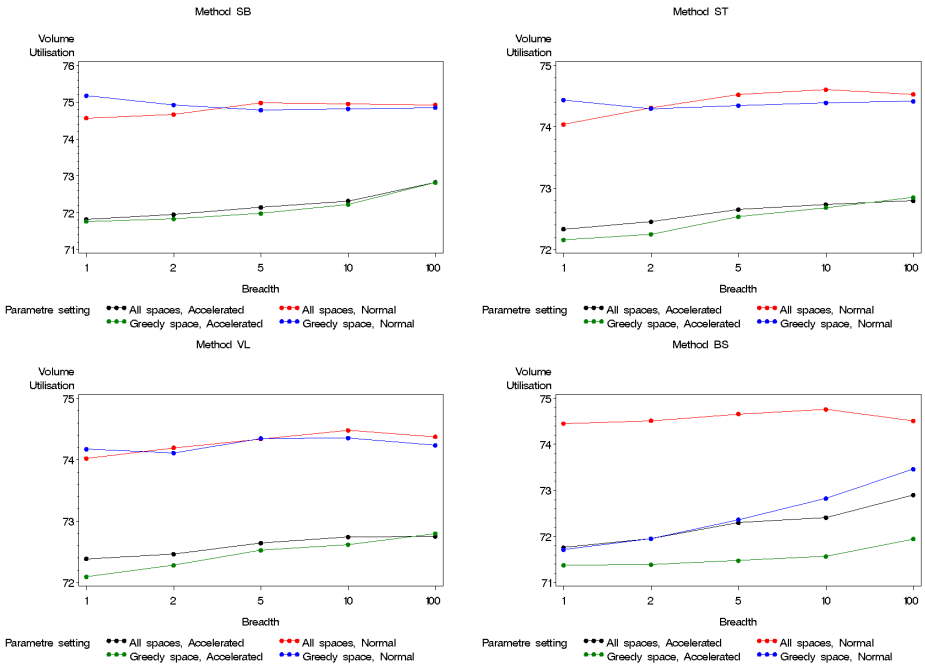


Figure 10.5: Aggregated results for the different methods with different breadths. Time limit: 10 seconds.

problems with between 1 and 50 different customers and 3 and 20 different box types.

The results clearly show that the effect of accelerating the search is negative. This is sound for all the different greedy methods and applies both when all spaces are tried and when this choice is made greedy. Hence, at least when there is no prior knowledge to the problem, the search should not be accelerated.

Based on the graphs in the figure, it is not possible to say anything about the best setting for the breadth. It does, though, seem as if, a smaller breadth is better when using the greedy space choice whereas a larger breadth is better when trying insertion in all spaces, but the difference is minimal. It is noteworthy, that if the search is accelerated, it is always better to keep the tree wide. The generally bad results found with the accelerated search can be explained by the fact that we look in the container for empty spaces. When there are several customers in the problem, focusing only on empty spaces is not enough to guarantee better loads. The big issue when there are several customers is,

whether the placement of a new box makes other boxes infeasible. Hence, the search for spaces alone is not enough to guarantee that the boxes we leave in the solutions are loaded in a good way.

It can also be seen, that whenever method BS is used, it should be done in combination with a tree search setup where the spaces are chosen by the search, and thereby not done greedily by method BS.

However, as seen in Figure 10.6, looking only on the data where one customer is present, the effect of accelerating the search cannot be disregarded. Here it can be seen, that for method ST the accelerated search outperforms the not accelerated version and for method VL the accelerated search is competitive with the not accelerated alternatives. But for both SB and BS, there is no advantage in accelerating the search. The same results are found when 60 seconds are

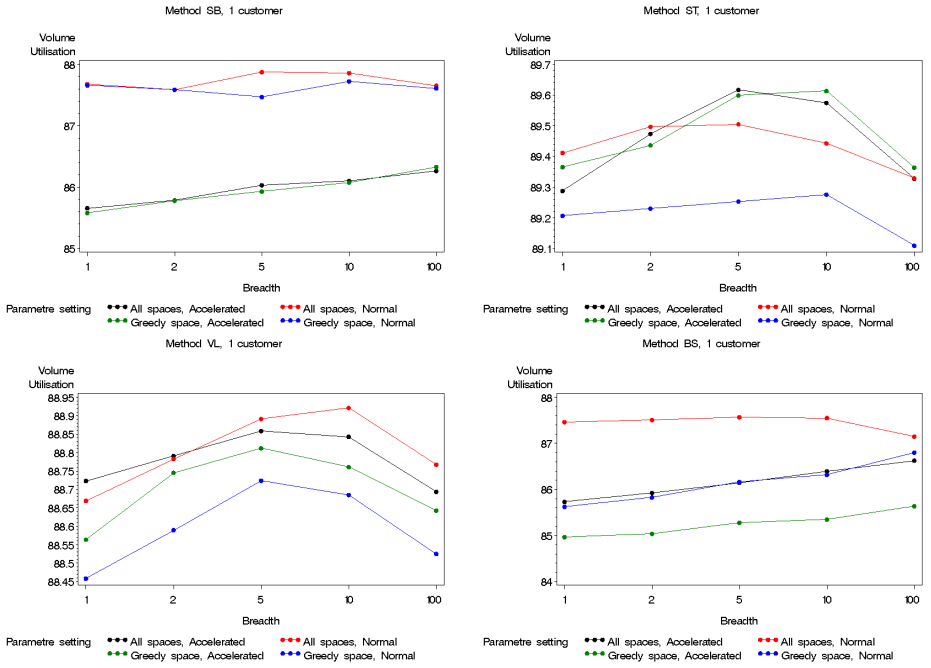


Figure 10.6: Performance when one customers exists in the problems. Time limit: 10 seconds.

given as time limit and when the breadth is chosen dynamically. This is shown on Figures D.2 and D.3 in Appendix D.

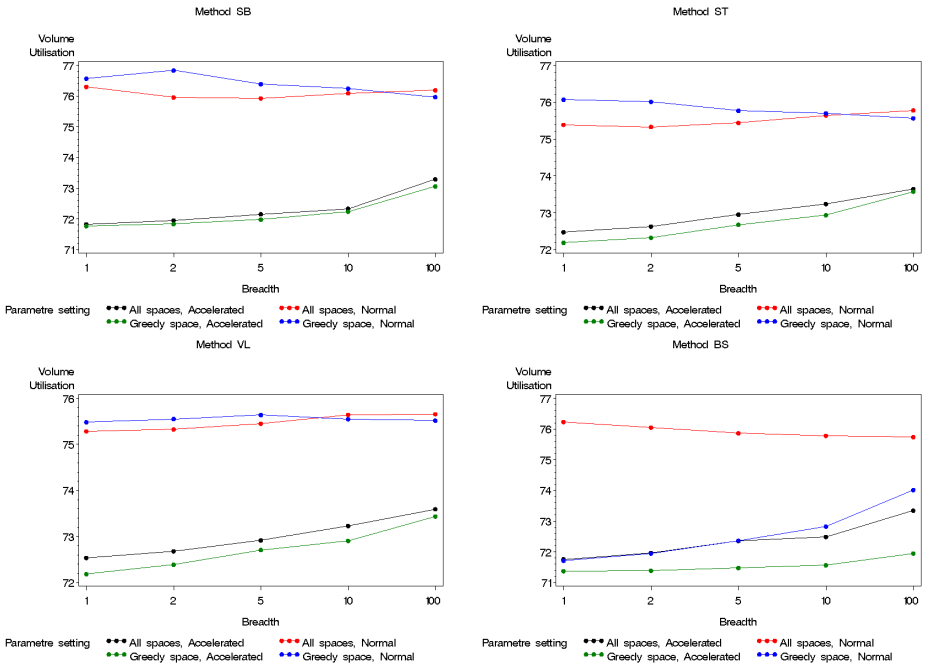


Figure 10.7: Aggregated results for the different methods with different breadths. Time limit: 60 seconds.

In Figure 10.7 the time limit is changed to 60 seconds, but otherwise the same test setup is used as in Figure 10.5. Overall the average volume utilisation have increased. This increase is more profound when the tree search is not accelerated.

As for the 10 second runs, it is seen that when using a greedy insertion the breadth should be smaller than when trying insertion in all spaces.

Based on the 10 and 60 seconds run (Figure 10.5 and 10.7) it can be seen that method SB without acceleration overall outperforms all other greedy methods. This is the case for all breadths tested. Because of this, only method SB without acceleration, is considered in further tests.

10.2.1.2 Time usage and obtained depths

When the breadth is static, there is no guarantee that all time available will be used trying to find a good solution. Moreover, it is not known how deep in the tree the search will get. If the breadth is chosen too small, the search will terminate early of its time limit. On the other hand if the breadth is chosen too large the search will terminate still being in the top of the search tree. The effect of this will be revealed in this section.

In the following we will use the measure *relative depth*, to express the relation between the depth reached when the search terminates and the number of boxes loaded in the best found solution.

In Figure 10.8 the results of the test are shown. On graphs to the left, the part of the used time is shown and on graphs to the right, the relative depth is shown. On the graphs it is seen, that whenever the breadth is small, not all

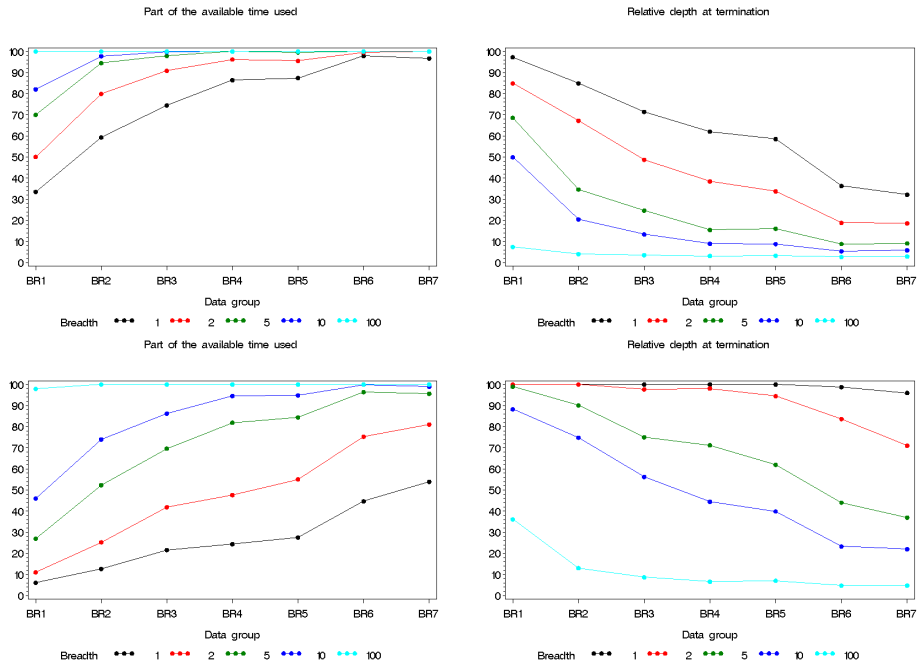


Figure 10.8: The part of the available time used (left) and the depth obtained (right) with different static breadths. The top graphs show results with a time limit of 10 seconds, and the bottom graphs with a time limit of 60 seconds.

the time available is used. However, it is also seen that only when the breadth is small, there is enough time to search to the bottom of the search tree. It is so both when the time limit is 10 seconds and when it is 60 seconds. It is clear that when 60 seconds are available, the tree search reaches further down in the tree. This is easily seen on the graphs showing the relative depth (right), but also on the left graphs, where a much higher percentage of the runs terminate prematurely. From the tests with a time limit of 10 seconds, it is interesting to follow the results obtained with the breadth equal 1. When we look at the left graphs we see that many solutions terminate early of the time limit. On the right graphs, however, it is seen, that even with the smallest possible breadth, we are sometimes unable to reach the bottom of the tree. This is the case, if the number of box types are greater than 3 (BR2-BR7). This indicates, that 10 seconds is very little time, compared to the complexity of the problems.

Furthermore, there is a clear connection between the different data groups and the results. This tells us that, not surprisingly, more time is used, in each node, when more different box types exist in the problems. The same correlation is found with the number of boxes present in the problems.

The overall conclusion of this section is that the breadth should be chosen based on the type of problem that should be solved when a fixed time limit must be kept. This is exactly the purpose of the dynamic breadth.

Even though not all the time is used when the breadth is small, the solutions obtained are competitive with the solutions found by the wider breadths. This was shown in Figure 10.5 and 10.7. This means that good solutions can be obtained when the search gets deeper in the search tree, as well as when the tree is kept wide.

10.2.2 Dynamic tree breadth

In the previous sections, it was shown how the overall performance of the tree search is, when a static breadth is used. The major drawback of the static breadth is, that the same breadth is not necessarily the best choice when different problems are solved. The most obvious problem characteristics such as the number of box types and the number of boxes present in the problem are closely related to the solution time and thereby also, as it was shown in Section 10.2.1.2, to the preferred breadth and the obtained depth. In this section we will explore properties of the dynamic breadth, introduced in Section 7.2.3.

If the proposed strategy works, the breadth should be scaled, such that the time is better utilised. Thereby also better results should be obtained. The time usage and the relative depth, as well as the breadth actually chosen by the dynamic breadth, should be investigated.

10.2.2.1 Overall performance

The first test we make with the dynamic breadth is comparable to the test made to the tree search with static breadth. However, the breadth is no longer a parameter. Instead the breadth weighting factor ψ, described in Section 7.2.3.1, page 84, becomes of importance. This parameter is tested with $\psi \in \{-1, 0, 1, 2\}$.

The results are shown in Figure 10.9. Here, respectively, 10 and 60 seconds are used as time limits.

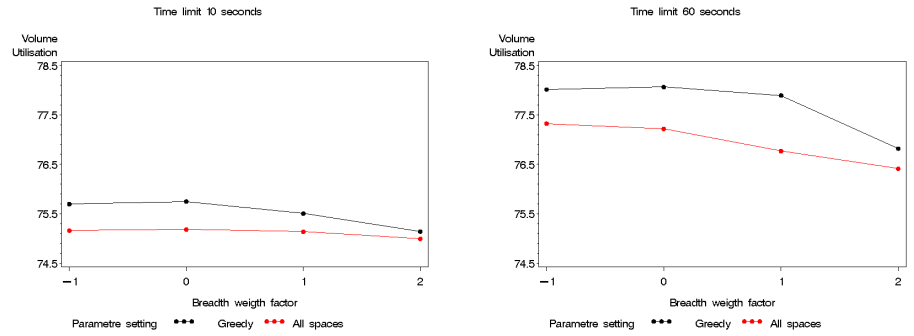


Figure 10.9: Aggregated results for method SB with dynamic breadth. On the left graph the time limit is 10 seconds, on the right it is 60 seconds.

The figure shows that no matter what breadth weight factor is used, the volume

utilisation is above 75% for the short runs and 76,5% for the long runs. When this is compared to the test using static breadth, see Figure 10.5 and 10.7, it can be seen that the dynamic breadth performs better then the static, no matter which static breadth is used. Figure 10.9 shows, that the best setting for the breadth weight factor ψ is either -1 or 0. This means that the breadth of the tree in the top should be kept relatively narrow. Moreover, it is seen that a greedy space choice is better than trying insertion in all spaces. This indicate that the faster greedy space choice in quality is close enough to trying all spaces. In Figure 10.10 results are shown when the data groups are varied.

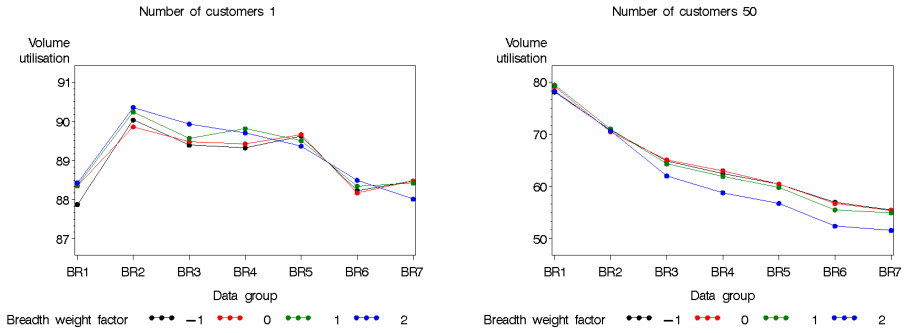


Figure 10.10: Performance dependencies on data groups, the number of customers and the breadth weight factor ψ. Time limit: 60 seconds.

The left graph shows the average achieved volume utilisation, when one customer exists in the problems. The right graph shows the average volume utilisation, when 50 customers are present in the problems. Both graphs are based on results obtained with a time limit on 60 seconds. Graphs showing results for the remaining customers and with a 10 seconds time limit, can be seen in Appendix D, Figure D.5 and D.6.

On the graphs it is seen, that when the number of customers is low (here 1) there is no big difference between the different choices of ψ. However, when many customers exist, ψ should not be chosen to large. This will be analysed further in the next section.

10.2.2.2 Time usage and obtained depths

Even though the dynamic breadth seeks to use all the time given, it is not always possible. This is because the calculated breadth is only based on an estimate of the actual time it would take to complete the search with the given breath. In some cases, this will lead to too few solutions and the algorithm will terminate prematurely. Because of this, it is interesting to find out how large a part of the available time is used and how deep the search gets into the tree, when the dynamic breadth is used. The results are found in Figure 10.11. From the

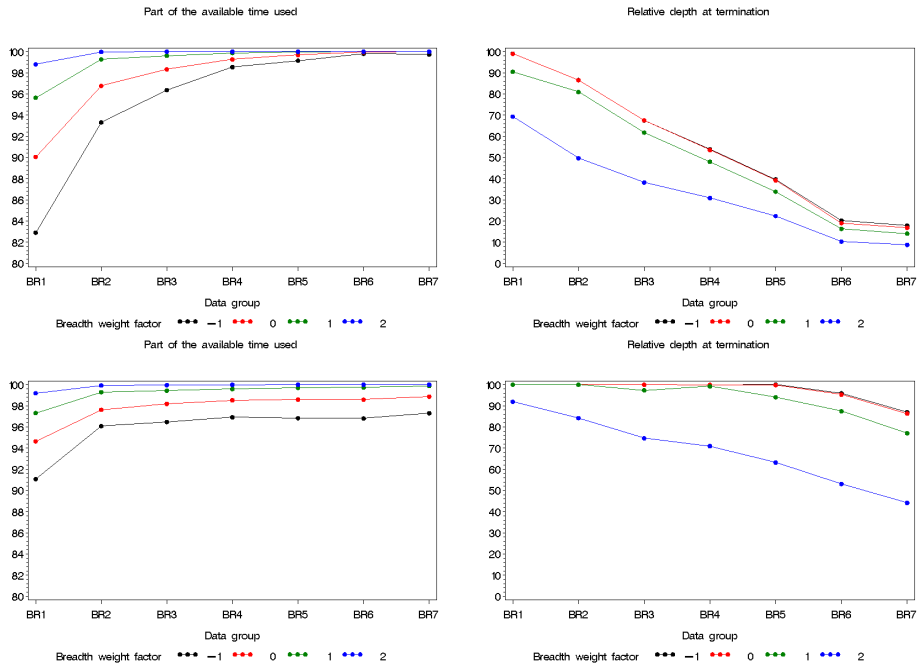


Figure 10.11: The part of the available time used (left) and the depth obtained (right) with dynamic breadth and different breadth weight factors. The top graphs are based on 10 seconds runs and the bottom ones are based on 60 seconds runs.

graphs on the left in the figure, it is seen that almost all the available time is used. Only for the fast runs with low breadth weight factor and few different box types, this is not the case.

Just as for the static breadths, it is seen on the top right graph that when the time limit is 10 seconds, we are generally not able to get to the bottom of the search tree. Here the results for $\psi = -1$ and $\psi = 0$ looks very much alike. This

is because the only breadths found is either 1 or 2, as these are the smallest valid choices of λ. In this case the obtained depths should be close to the results found for the static breadth with $\lambda = 1$ and $\lambda = 2$, which is also the case. By choosing $\psi = 2$ it is clear that we do not get very deep in the search tree, both when the time limit is 10 seconds and when it is 60 seconds.

When 60 seconds is used and many different box types exist (BR6 and BR7), we are not able to get to the bottom of the search tree. This, again, is because the breadth cannot be chosen small enough. The overall conclusions on this test is, that 10 seconds is simply not enough time to search to the bottom of the search tree, when more than 3 box types exist in the problem. The dynamic breadth, however, is able to choose breadths which utilise all the time available, and whenever possible also gets to the bottom of the search tree. The later is seen on the bottom graphs. This, of course, depends on the choice of ψ.

From Figure 10.9 it was found, that the best choice for ψ is either -1 or 0. In combination with Figure 10.11 this shows, that overall it is better to get rather deep in the search tree, than seek thoroughly in the top nodes of the tree. The results for $\psi = -1$ and $\psi = 0$ from Figure 10.9 does not deviate much. Marginally better results are, however, obtained with $\psi = 0$, which is therefore chosen. This means that the breadth should be kept relatively unchanged throughout the search.

The test shows that whether the time is spent in top of the tree or further down, becomes increasingly important when considering more customers. When there are just one or two customers, greedy solutions found in the top are typically pretty good, but when considering many customer this is not the case. Instead time should be spent going faster down the tree, so that boxes placed later also will be explicitly placed by the tree.

10.2.2.3 Chosen breadths

When the dynamic breadth is used, it is interesting what breadths are actually chosen. Since the breadth can change in every iteration, no single measure can be given describing the size of the breadth. We have chosen to report the average-, start- and end breadths as well as the overall average breadth used throughout a run. This should give a good indication of whether the breadth is adjusted reasonably compared to the time limits.

In Figure 10.12 the results are shown. The different data groups are shown on the horizontal axis and the average start and end breadths and the overall average breadths are shown on the vertical axis. The top graphs show the average start breadth, the middle graphs show the overall average breadth and the bottom graphs show the average end breadths.

Graphs to the left indicate runs with 10 seconds used and graphs to the right indicate that 60 seconds have been used.

On the top two graphs in Figure 10.12 the start breadth are shown. These relate to each other exactly as expected, meaning that a higher choice of ψ yields larger start breadths. This is independent of the data groups. It is also seen that different start breadths are chosen depending on the different data groups and on the available time. On the middle graphs the overall average breadth is shown. Here, the exact opposite relation between the breadth weight factors are present. The result should be seen in combination with the bottom graphs showing that the end breadth becomes much larger, when the start breadth is small. This is a natural development as less time is used in the beginning when the start breadth is small, which means that more time is available later in the search tree and a wider search can be performed. The figures show that the adjustments of the breadth made with the dynamic breadth strategy, depends of the problem size and the number of box types. It is also seen that ψ is effective in controlling whether time should be used in the top or bottom of the tree.

From all six graphs in Figure 10.12 it is clear, that many different breadth have been chosen. This choice is seen to depend on both the data group, the time available and the choice of ψ. A single value for λ - as chosen by the static breadth - will never be able to describe this diversity.

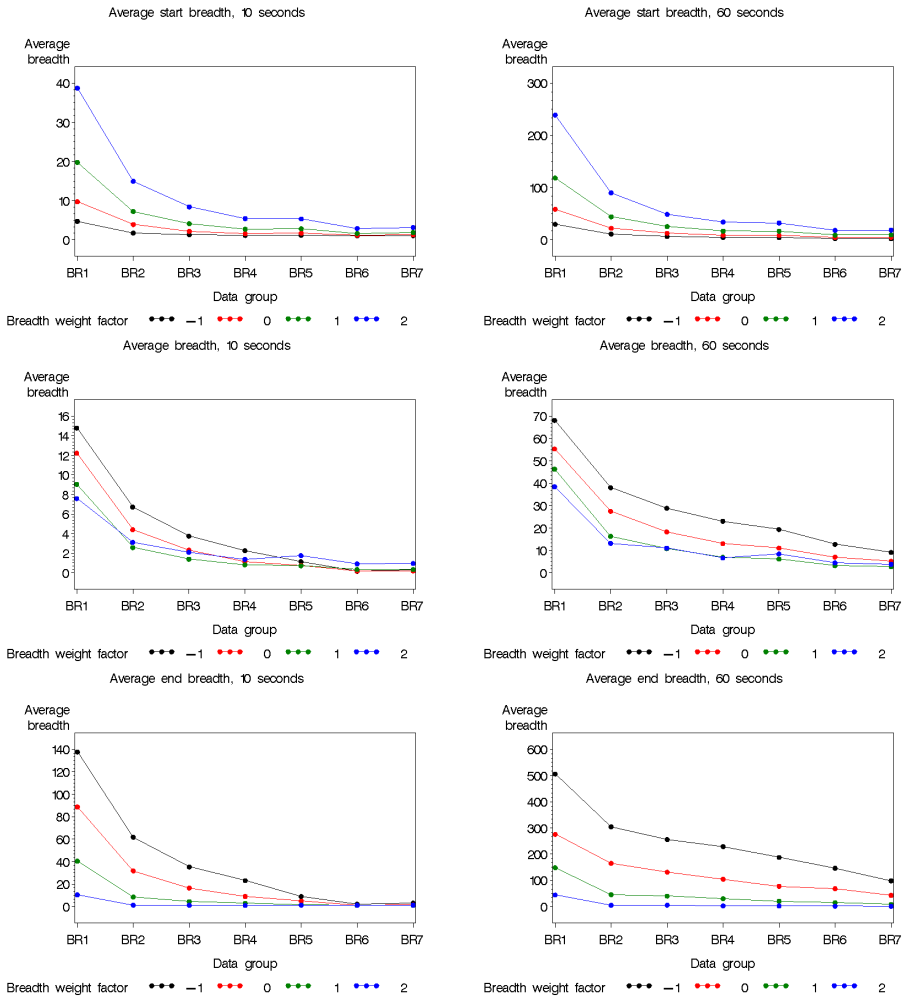


Figure 10.12: Start, average and end breadth when 10 or 60 seconds are used. Method SB, not accelerated and greedy space choice.

10.3 Summary

In this chapter we have found the best settings for our algorithm. In the tuning of the greedy methods it was found that the best box rotation sequence is given by the rule YXZ. Sorting the batch after the volume was the better choice. Moreover, when customers exist, the minimum number of boxes to load at a time κ should be one, meaning that one customer is loaded at a time. Based on large time usage and the poor results, the greedy method EL was discarded, already before it was tried in combination with the tree search framework.

When the greedy methods were combined with the tree search framework, many new parameters were tested. Accelerating the search, generally worsens the solution quality. However, if all boxes on the load was assigned to the same customer accelerating the search, in combination with greedy method ST, did indeed perform better than any other method.

It was furthermore found that with methods SB, ST and VL the space choice should be made by the greedy procedure, whereas in method BS, insertion in all different spaces should be tried. Considering the results of the static breadth test, SB outperformed the other methods. Therefore it was chosen. Because of the limited time, the speed of SB proved essential. The proposed dynamic breadth is an improvement to the method compared with the static breadth. Mainly because this strategy gives the opportunity to utilise all the time available and ψ gives an effective way to control whether more time should be spend in the top or bottom of the tree. When the dynamic breadth is used the breadth weight factor ψ should be chosen low. This indicates that it is important to spent time placing each box explicitly in the tree search. The overall best breadth weight factor setting proofed to be $\psi = 0$.

Chapter 11

Test

Now the time has come to test our final algorithm. This is done on all the problems described in Chapter 9. The settings for the algorithm that proved best with regard to our type of problems and time horizons were: The tree search in combination with greedy method SB, a greedy space choice, the dynamic breadth with $\psi = 0$ and no acceleration. This is described in Chapter 10. Therefore, these settings of the algorithm has been used through all the following tests. An exception is made in Section 11.1.2. Furthermore, all tests are made with a time limit on 10 or 60 seconds, the choice of time limits should reflect the purpose of using the method in a 3L-CVRP framework.

In Section 11.1, we will test the proposed algorithm on the data available from the OR-Library using the realistic model from Chapter 5. Both problems where customers are assigned (as described in Section 9.1) and the original problems without customers, are solved.

In Section 11.2, we will test how the proposed algorithm performs on the problems faced by Aarstiderne, using the specialised model described in Section 6.1, page 51. It quickly turns out, that very large parts of the container room is unoccupied, which means that the packing problems are easily solved. Therefore, we change make the container room smaller to further challenge the algorithm.

The subsequent section (Section 11.3) deals with the problems from Johannes

Fog. Here few problems exist. The algorithm is tested with the specialised model, proposed in Section 6.2, page 55. Again the original problems are always solved to optimality, even though this is not trivial in all cases. Because of this, extra problems are generated, based on the original test data. This gives the opportunity to test the algorithms further.

Finally, we will look at how the time usage is distributed among the different operations performed, when the algorithms runs. This is done in Section 11.4.

11.1 OR-Library problems

In the first section customers are assigned to the boxes as described in Section 9.1, page 87. This is done to test the performance of the algorithm and the effect of the multi-drop constraint. The next section focuses specifically on the standard OR-Library problems without customers, to allow a comparison with other authors.

11.1.1 Problems with multi-drop

Table 11.1 shows the results obtained when solving the OR-Library problems. All results are obtained with a time limit of either 10 or 60 seconds. The problems where $nC = 1$ are the original OR-Library problems, while the other results are with more customers randomly assigned to the problems.

	10 sec.		**60 sec.**	
nC	**Avg.**	**SD.**	**Avg.**	**SD.**
1	87,96	2,31	89,07	2,10
2	84,31	3,07	85,96	2,71
5	76,10	6,37	78,52	5,45
10	69,64	8,74	72,64	7,54
50	61,13	10,08	64,45	9,37
All	75,83	11,89	78,13	10,81

Table 11.1: Test of OR-Library problems with customers.

The overall average volume utilisation for the 10 seconds run is 75,8%, which can be improved to 78,1% when using 60 seconds, a gain of 2,3%. It is seen that the results are highly dependent on the number of customers. In the 10 seconds run, when there is just one customer, 88,0% of the container is filled. This drops to just 61,1% when 50 customers exist. The same effect is seen for the 60 seconds

run. This clearly shows, that adding customers is a serious constraint to the solution space, which makes it hard to obtain a high volume utilisation. When looking at the standard deviation, it can be seen, that it is increasing with the number of customers. When just one customer is present, the results will only vary a few percent but adding customers leads to much greater variation in the results.

Figure 11.1 shows a more detailed picture of the solution quality obtained when solving the problems. On the horizontal axis the number of customers in the problems solved, are shown. Each line represent results from distinct data groups.

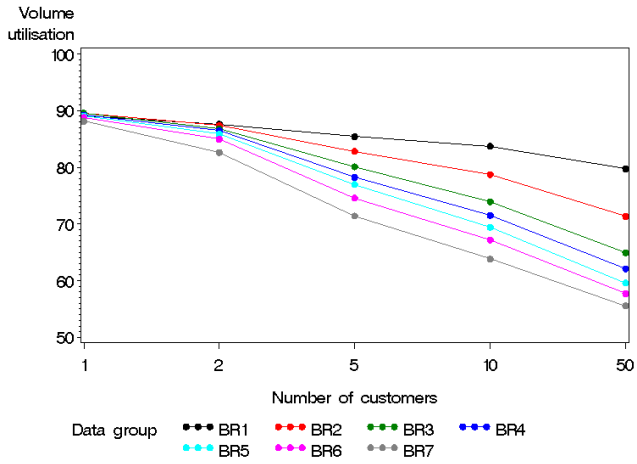


Figure 11.1: The results dependencies on the number of customers and the number of box types. Time limit: 60 seconds.

As already shown in Table 11.1, there is a clear connection between the number of customers and the solution quality. The figure, however, reveals that the solution quality depends not only on this, but also on the combination of the number of box types and the number of customers in the problem. When only 3 box types exist (BR1) the solution quality only drops 5% going from 1 to 50 customers, when 20 box types exist (BR7) the difference is around 30%. This relation between box types and solution quality also explains the large standard deviations seen in Table 11.1, when many customers are present.

It is not surprising, that the difficult problems both have many customers and box types. When the load is only weakly heterogeneous the multi-drop constraint will have little effect, as several boxes of each box type are available to all customers, leaving much freedom when choosing each box.

The general impression gathered when solving the problems is that adding the multi-drop considerations, makes the characteristics of the problems very different from the basic OR-Library problems. The problems becomes more restricted and the solution quality must be expected to decrease.

11.1.2 The original problems from the OR-Library

Many solution procedures to solve the CLP have preceded this work. To be able to compare our method with work done previously, we will in this section focus on solving the original problems from the OR-Library.

We have not optimised our algorithm specifically for this purpose, but it was seen in Section 10.2.1, that when only one customer exists in the problems a different setup of the algorithm may be more suitable. It was found that better performance on problems with only one customer, was obtained by using an accelerated tree search in combination with greedy method ST. Spaces should still be chosen greedily and the breadth should be chosen dynamically with $\psi = 2$. This can be seen in Figure D.4. $\psi = 3, 4$ have also been tested, but with worse results.

The results obtained with our algorithm are reported in Table 11.2. A time limit of 60 seconds have been used, and no customers are added to the problems ($nC = 1$).

Data group	Min.	Avg.	Max.	SD.
BR1	83,38	90,48	96,81	2,70
BR2	85,31	91,16	94,93	1,89
BR3	87,77	91,33	94,97	1,53
BR4	87,91	90,98	94,09	1,34
BR5	86,41	90,83	93,25	1,32
BR6	87,03	90,19	93,12	1,26
BR7	84,81	88,36	91,48	1,27
All	83,38	90,48	96,81	1,93

Table 11.2: Test of OR-Library problems with one customer. Time limit: 60 seconds.

The test shows an average volume utilisation of 90,5%, looking across all data groups. The best results obtained are for BR3, with 8 different box types, where the average is 91,3%. The worst results are obtained for BR7 (20 box types) with 88,4%. The data group that vary most in solution quality is BR1, with the highest standard deviation and also containing both the overall best and worst result.

The overall average of 90,5% seems a very good result compared to our general method where we obtained 89,1% with the same limit of 60 seconds (results from Table 11.1). As seen in Table 11.3, this however, is only in the cases with one customer. The table shows the results obtained when the method is used on problems with more customers. In the last column, the difference between results obtained by our general method and the method tuned for one customer problems, is shown. Even though the gain for the one customer problems is notable, it is also obvious that the method behaves badly, when applied to problems with more customers. This reflects the results from the tuning (Section 10.2.1), where it was shown that accelerating the search leads to bad results, when more customers are considered.

nC	**Avg.**	**SD.**	**Diff.**
1	90,48	1,93	1,41
2	83,33	3,84	-2,63
5	72,10	7,42	-6,42
10	65,25	8,12	-7,39
50	56,16	7,93	-8,29
All	73,46	13,84	-4,67

Table 11.3: Settings good for one customer compared to the standard method (Table 11.1) based on OR-Library problems with more customers. Time limit: 60 seconds.

The results for one customer can be compared to work done by other authors. A selection of results obtained by others, can be seen on Table 11.4 together with our results. In the table, B/R is the results obtained by a construction algorithm proposed by Bischoff and Ratcliff [3], B/G is a genetic algorithm proposed by Bortfeldt and Gehring [4], E is the tree search approach by Eley [10], B/G/M is a parallel tabu search proposed by Bortfeldt et al. [5], M/O is a GRASP proposed by Moura and Oliveira [24] and L/Z is the algorithm proposed by Lim and Zhang [20]. Common for all these algorithms is that full support of the boxes are required.

Authors	**Avg.**
B/R (1995)	84,94
B/G (2001)	90,06
E (2002)	88,75
B/G/M (2002)	92,20
M/O (2005)	89,73
L/Z (2005)	91,81
Our method	90,48

Table 11.4: Solution quality compared to other authors.

When comparing our algorithm with others, it is important to remember that our method is not particularly created to solve problems from the OR-Library.

It should be able to handle them, as well as more complex problems. The time limit used is also a consideration. As said we have set the time limit to 60 seconds, while this vary much between other authors. Bischoff and Ratcliff [3] use a single construction method, while the results by Bortfeld et al. [5] and Eley [10] are obtained, using a time limit of 10 minutes. In Bortfeldt and Gehrings genetic algorithm [4] a time limit of 500 seconds is used. Moura and Oliveira reports that average solutions times are under 64 seconds, when they perform 200 iterations of their GRASP approach [24]. The solution times reported by Lim and Zhang [20], spans between 1 and 4600 seconds.

Our results are not fully competitive with the the newest methods developed to solve these academic problem instances. However, within a reasonable time limit we still obtain an average volume utilisation above 90%.

Many authors have also reported results for either a relaxed stability measure or no assurance of stability at all. This generally allows for larger volume utilisation. A hybrid parallel method developed by Mack et al. [21] reports average volume utilisations around 93,8% obtained within 10 minutes. A GRASP approach by Parreño et al. [26] also gives volume utilisation around 94%. We have chosen not to focus on tests within this area, even though reducing the demand for support for the boxes is possible, in our implementation.

11.2 Aarstiderne

11.2.1 The original problems

The problems from Aarstiderne were tested with the tree search using greedy method SB with a time limit of 10 seconds. The result of the test is shown in Table 11.5.

	Min.	**Avg.**	**Max.**	**SD.**
Volume utilisation	4,75	26,34	50,18	10,66
Time	0,00	0,00	0,01	0,00

Table 11.5: Test result for the 84 Aarstiderne problems

The results are very convincing. All 84 problems are solved to optimality and all, are solved in less than 0,01 seconds. All solutions are found in the first node of the tree. The table shows however, that the volume utilisation is very low.

On average only 26% of the container is filled, and the problem taking up most space fills half of the container. An example of a packing is seen in Figure 11.2. This problem consists of 152 boxes going to 108 customers. The same packing is shown on the two figures, but from different angles. The colours on the left figure indicate the sequence on the boxes, blue colours indicate low sequence and red colours high. The colours on the right figure indicate the different box types, the blue boxes are standard boxes with fruit and vegetables, the red boxes are specialities like meat, fish or beer.

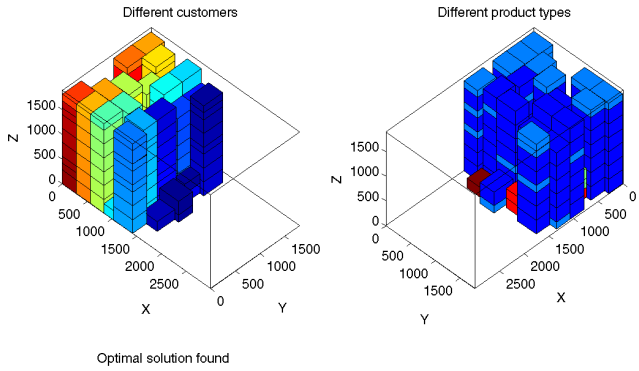


Figure 11.2: An example of an Aarstiderne packing (optimal solution).

The picture of the packing shows a half filled container, the volume utilisation is only 37%. The boxes are stacked according to the specified rules for Aarstiderne, therefore it is seen that only boxes with the same base area are stacked. The effect of the sequence constraint, is easily seen by the bands of changing colours on the left figure. The packing has many stacks that do not expand to the top of the container. It is easy to see, that the boxes with specialities are hard to place. There are only few of them that are alike, which in combination with the sequence constraint, means that they cannot be stacked.

Figure 11.3: On the left, the route corresponding to the Aarstiderne packing shown on Figure 11.2. On the right a truck packed with groceries, ready to leave the depot.

On Figure 11.3 to the left, the route corresponding to the packing seen on Figure 11.2 is shown. The route is seen to be in the area around Horsens.

As all the data used corresponds to routes actually driven, the test shows that Aarstiderne has a very low volume utilisation on their loads. The reason for this could be, that other constraints than the capacity are binding, when looking on the combined problem of vehicle routing and container loading. This could for instance be the maximal allowed driving time for the driver.

On Figure 11.3 to the right, a loaded Aarstiderne truck, ready to deliver groceries, can be seen. It is seen on the picture that the limits concerning stacking is reasonable, it is also seen that the truck is not heavily loaded - which the test has shown, is the general case.

The results clearly show, that the trucks are overdimensioned, compared to the loads driven and a suggestion could be, to use smaller trucks to deliver the goods.

11.2.2 Generated problems - smaller container

The test on the original data from Aarstiderne did not really give an opportunity to show the performance of our algorithm. The problems were to easy. Because of this, we have chosen to generate some new problems based on the original Aarstiderne data. To make the problems harder the container is reduced in size. Three new settings have been tested, where the size of the container is reduced between 35% and 55%. Figure 11.4 shows the results, with a time limit of 10 seconds. The horizontal axis shows the ratio between the volume of the

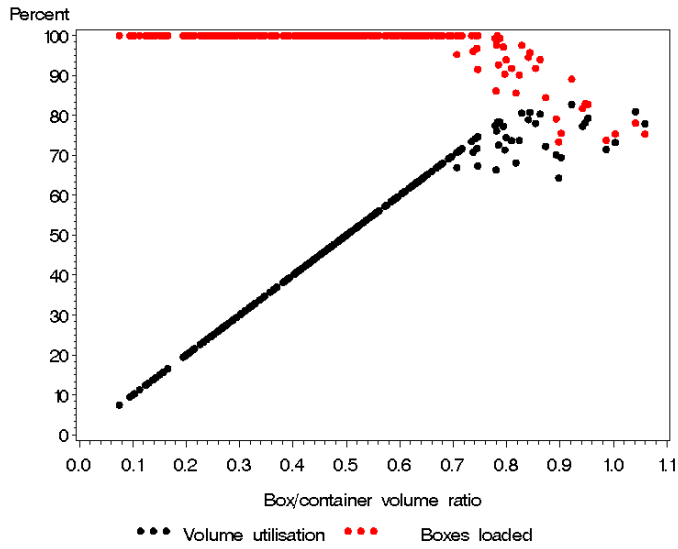


Figure 11.4: The connection between volume utilisation and ratio of boxes packed. Aarstiderne data. Time limit: 10 seconds.

boxes and the container in a problem. On the plot, the dots are connected two and two, so that for a chosen box/container volume ratio, a pair of black and red dots corresponds to the solution to one problem. The black dot shows the volume utilisation of the solution, the red shows the percentage of the volume of the boxes packed. This means that if the red dot is at 100%, the solution is optimal. The graph shows that as long as the boxes fill less than 70% of the container, an optimal solution is always found. For a ratio above 70% the volume utilisation flattens out, and the highest volume utilisation is around 80%. At the same time the percentage of packed boxes falls. Optimal solutions are not found when the ratio is above 80%, and cannot be guaranteed when the ratio is between 70% and 80%. When the total volume of the boxes is the

same as the container volume (the ratio is 1) only about 70% of the boxes are contained in the solution.

Figure 11.4 shows, that for the Aarstiderne data, the box/container ratio should be above 70% for the problems not to be trivial. When the ratio is higher we cannot guarantee optimality in all cases, but some problems are solvable. A volume utilisation above 80% is, however, rarely seen.

That problems with a box/container ratio up to 70% and many customers are solved to optimality, are explained by the OR-Library test, Figure 11.1. As the box types of Aarstiderne are not that diverse, the multi-drop constraint is not limiting the solution space much. This allows high volume utilisations even for problems with many customers.

On Figure 11.5 the same consignment as on Figure 11.2 can be seen, but the container has been reduced in size. This is done by lowering the container roof from 190cm to 90cm, a reduction of 53%. The solution shown is optimal and has a volume utilisation of 78%. It was found within 10 seconds. Compared to Figure 11.2, the boxes are placed with greater care. More stacks are filled to the roof. As the problem no longer is easy the algorithm is forced to make more clever choices.

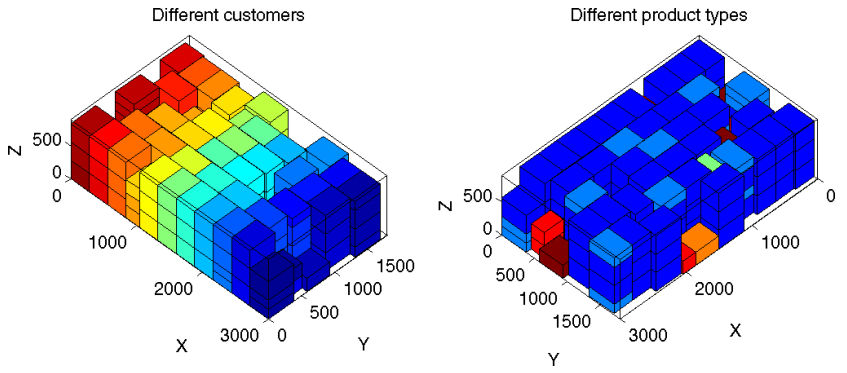


Figure 11.5: An example of an Aarstiderne packing, where the container room has been reduced with 53% (optimal solution).

11.3 Johannes Fog

11.3.1 The original problems

In Table 11.6 it is seen, that all the problems were solved in less than 0,001 seconds. The mean volume utilisation is 26% but loads with box/container ratios up to 71% are packed. All the problems were solved to optimality. This shows that the algorithm is fully capable of loading consignments of the type driven today by Johannes Fog.

	Min.	**Avg.**	**Max.**	**SD.**
Volume utilisation	5,47	25,85	71,21	23,13
Time	0,00	0,00	0,00	0,00

Table 11.6: Test results for the 8 Johannes Fog problems. Time limit: 10 seconds.

Most of the loads were found in the first node of the tree, but for two problems, the solutions were found in a depth of 2 and 4, respectively. Figure 11.6 shows one of the solutions.

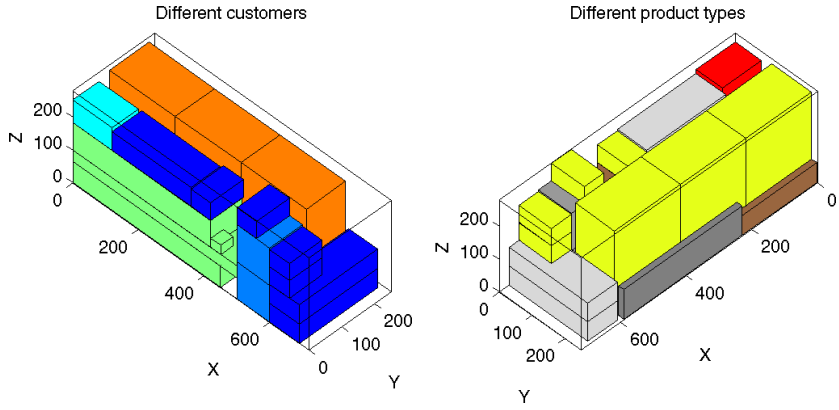


Figure 11.6: An example of an Johannes Fog packing (optimal solution).

There are two sub figures, the left shows the sequence of the boxes, the right shows the product types. The solution shown is optimal and has a volume utilisation of 71%. There are 6 customers and 19 boxes. Information about which colour corresponds to which customer and all the products and the truck, can be seen on Table 11.7. The list shows that very different products are placed on the same load, both big pallets of rockwool, stacks of gypsum boards and

long tree poles. Figure 11.7 (right) shows a selection of these products. The pictures show that it is a key factor to be able to measure the load bearing strength (LBS) of the products. A ton of stone cannot be placed on top of some tiles for the bathroom.

Customer	#	Dimensions $L \times W \times H(cm)$	Rotations	Weight kg	LBS kg/cm^2	Description
-	-	$720 \times 250 \times 280$	-	9500*	-	The truck
1 (Dark blue)	3	$240 \times 90 \times 58$	$0 \times 0 \times 1$	777	0.1	Gypsum boards
1	1	$360 \times 90 \times 16$	$0 \times 1 \times 1$	193	0.02	Gypsum bars
1	4	$100 \times 60 \times 53$	$1 \times 1 \times 1$	3	0.0002	Rockwool
2 (Light blue)	2	$100 \times 100 \times 113$	$0 \times 0 \times 1$	900	0.1	Founda. blocks
3 (Turquoise)	1	$120 \times 80 \times 70$	$0 \times 0 \times 1$	250	-1	Tiles
4 (Green)	1	$420 \times 95 \times 83$	$0 \times 0 \times 1$	1500	0.1	Tree poles
4	1	$450 \times 112 \times 65$	$0 \times 0 \times 1$	1500	0.1	Tree poles
4	1	$500 \times 85 \times 35$	$0 \times 1 \times 1$	100	0.1	Wood boards
4	1	$38 \times 31 \times 29$	$1 \times 1 \times 1$	5	-1	Cardboard box
5 (Yellow)	1	$228 \times 60 \times 16$	$0 \times 1 \times 1$	40	0.03	Flooring
6 (Orange)	3	$200 \times 120 \times 265$	$0 \times 0 \times 1$	75	-1	Rockwool

* *The capacity of the truck*

Table 11.7: List of products corresponding to the packing shown on Figure 11.6.

Comparing the solution of Figure 11.6 with the packing list, it can be seen that tiles and some gypsum boards are placed on top of the wood, stone is placed on stone, and rockwool is placed on top of both wood, gypsum and stone. These would all be realistic choices.

The route corresponding to this load can be seen on Figure 11.7 to the left. The route is seen to be run all the way across Northern Sealand. Two customers are

Figure 11.7: To the left, the route corresponding to the Johannes Fog packing shown on Figure 11.6. To the right, some of the products included in the packing.

placed by the green mark (Sjællands Odde) on the map. Because of the great distance travelled with this load, the value of packing all the boxes is high.

The test shows, that we are able to solve the problems corresponding to the loads actually driven by Johannes Fog, and within a very short time. This shows that the model is flexible enough to allow realistic loads to be packed. A concern could be that the model is not strict enough, and it is possible to pack loads that would not be feasible in a real world scenario. To test this, new random problems are generated.

11.3.2 Generated problems - new random consignments

The problems from Johannes Fog were all solved to optimality, without testing the algorithm to a great extend. To put the algorithm to a harder test, new problems have been generated. The new problems have been generated by gathering all the products collected at Johannes Fog, keeping the information of which customer each product belongs to. From this list of about 80 products, random customers have, in groups, been assigned to loads. In this way 9 new problems have been generated, each containing between 4 and 9 customers and the total volume of the boxes is between 28% and 86% of the container volume. The problems are listed in Appendix C. The sequence of the customers is chosen randomly without considering the actual route. In Table 11.8 the results of a test where the tree search has run for 10 seconds is shown.

	Min.	**Avg.**	**Max.**	**SD.**
Volume utilisation	24,16	44,54	61,97	14,47
Time	1,26	6,30	9,57	2,47

Table 11.8: Test results for the nine random consignments. Time limit: 10 seconds.

It is seen that the average volume utilisation is 44,5%, nearly twice as high as for the original problems. However, in none of these problems all boxes are placed. On average, only 63% of the time has been used, indicating that for a very short run, with relatively few boxes, the dynamic breadth does not have enough opportunity to adjust itself.

The test shows, that even for the problems with only around 30% box/container volume ratio, all boxes cannot be placed. This shows that for problems of this type, with many highly odd sized boxes, the volume ratio alone is not a very good indication of how hard a problem is to solve. The characteristics of each box makes a difference. This is because of the very different types of boxes and their ability to carry other boxes.

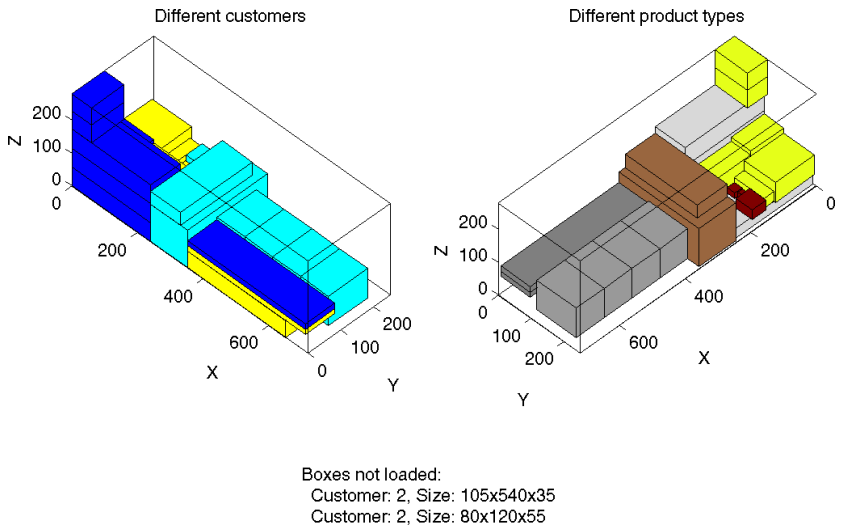


Figure 11.8: An example one of the generated problems using Johannes Fog products. Two boxes have not been packed.

In Figure 11.8 an example of a solution to one of the generated problems can be seen. The problem has 4 customers and a total of 25 boxes. The volume utilisation is 34%. In the solution, two boxes have not been packed. In Table C.9 in Appendix C the packing list corresponding to the problem can be seen. The colours blue, turquoise, yellow and red (hidden) on the left figure correspond to customers 1, 2, 3 and 4. The boxes not packed are some 5,4 metres long wooden planks and a heavy sack of concrete. By only looking at Figure 11.8, it could seem, that there is a large area unused on top of the four equally large turquoise boxes, where the concrete could easily be placed. It should be remembered, that those boxes actually represent four sacks of cement mix, and that it is impossible to stack on top of the uneven surface of such sacks. It is easy to see that the long wooden planks, going from one end of the truck to the other, are hard to place without neglecting the sequence constraint.

Figure 11.8 clearly shows, that if the solution should be used for more than a feasibility check, a secondary objective is needed. The rockwool placed last (blue boxes), is just piled up in one end to the full height of the container. To make the load more stable they could just as well have been placed next to each other. The reason why the algorithm stops, is that it does not matter where the boxes are placed, as long as they are inside the container.

11.4 Time usage

We have now tested the algorithm on a number of different problems and seen the overall performance of the algorithm. It is, however, also important to analyse what procedures in the algorithm that are time consuming compared to others. A detailed analysis has been performed in Rational Quantifier to uncover this information.

Figure 11.9 shows the time used in the different parts of the algorithm. The times are collected from a test run on a problem from the OR-Library, with the realistic model and five customers assigned. The time used by each method is shown in percent of the total time, used by the tree search.

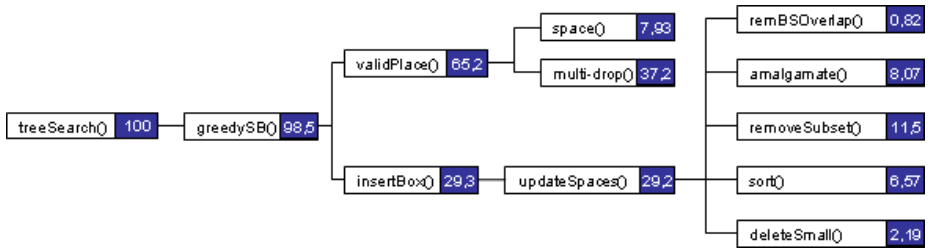


Figure 11.9: The distribution of the time used on key methods of the algorithm, in percent. OR-Library problem BR4 no. 56 with five customers and 233 boxes. Time limit: 60 seconds.

It can be seen, that almost all the time is used by the greedy method `greedySB()`, leaving only 1,5% of the time for administration of the tree framework. The greedy method SB uses about 2/3 of the time on checking whether a chosen box/rotation/space combination is valid. In this case almost 40% of the total computation time is spend on checking the multi-drop constraint, `multi-drop()`, and 8% is spend checking if the space is big enough to hold the box, `space()`.

The remaining time of `greedySB()` is used on the procedure to insert a box in the container. This procedure consists of two things, to actually place a box and to update the empty spaces. Basically all time is spend on updating the empty spaces, after the box is placed. It can be seen, that only little time is used on the actual creation of the new spaces (`remBSoverlap()`), instead time is used on amalgamation of the spaces and on removing subsets. The greedy algorithm demands, that the spaces are sorted at all times, and it can be seen that 6,6% of the time is spend doing this.

It is important to notice that a major amount of time is spend checking at each box insertion, that the load is multi-drop feasible. This can be explained by the fact, that the check is made regarding all other boxes in the container. As this is a problem containing more than 200 boxes, it is a demanding check. The insert procedure is not as time consuming as the valid place check. This is connected to the fact, that many valid place checks are performed before every single box is inserted. It should also be noticed, that nearly no time is spend in `greedySB()` besides the two methods mentioned. This is because of the simple nature of the method where no evaluation of box/rotation/space combinations, is done. The first match is simply chosen.

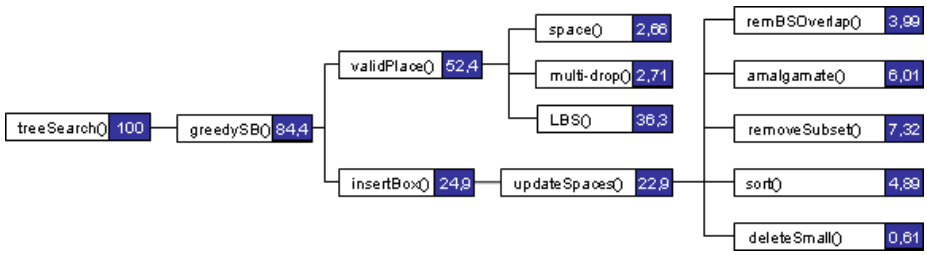


Figure 11.10: The distribution in percent of the time used on key methods of the algorithm. Johannes Fog problem Generated 4 with five customers and 22 boxes. Time limit: 60 seconds.

Figure 11.10 shows a similar test on one of the generated problems from Johannes Fog, using the specialised model from Section 6.2. The main difference is that on these problems, load bearing strength is also a concern. In the problem, 22 boxes should be distributed to five different customers. Compared to Figure 11.9 an extra method is included, it is the load bearing strength check, `LBS()`, which takes a substantial amount of the total time. It is also seen, that the time is used in `remBSOverlap()`. This can be explained by the fact, that overhanging spaces are allowed in this model, which results in more box/space overlaps.

Because of the few boxes present in the problem, the check on the size of the space and the sequence check, is not that time consuming. Therefore, about the same fraction of the time used by `greedySB()`, is spend checking for valid places in both problems. Also, notice that less of the total time is spend in `greedySB()`, only 84,5%. This is because greedy solutions are produced faster, as less boxes exists, leaving relatively more time for bookkeeping in the tree search.

11.5 Summary

We have now tested the algorithm on a large and very diverse set of problem instances. Both real world problems and academic problems have been considered. The test clearly shows, that solution quality depends strongly on what type of problem is solved.

The OR-Library problems were solved with customers assigned to the boxes. This gave a clear indication of the extra complexity introduced by adding customers to the problem, especially combined with a high number of box types. The test concentrating on the original problems with just one customer showed, that our algorithm produces results comparable to other authors, although our focus has been to solve problems with more constraints.

The original problems from Aarstiderne were all easily solved, because of the large container compared to the total volume of the boxes. The problems with reduced container size showed, that a volume utilisations between 70% and 80% can be achieved. Considering the very large number of customers present in each load this achievement is good. The results are also comparable with the ones found for the problems from the OR-Library, where problems with many customers and 5 to 10 different box types, yield the same results.

The original problems from Johannes Fog were all solved very convincingly. To all the problems optimal solutions were found within fractions of a second. Generating new loads showed, that the possible volume utilisation depends heavily on the mix of products and also the number of customers. Even when the box/container volume ratio is 25%, packings containing all boxes cannot be guaranteed. On average a volume utilisation of 44,5% was found.

The analysis of the time usage of the algorithm revealed several issues. The primary thing to note, is that both the multi-drop constraint and the LBS constraint are very time consuming to check, making the valid place method much more difficult, than if only the basic box/rotation/space comparison was made. About 20-30% of the time is spend maintaining the spaces.

Both the results based on Johannes Fog and Aarstiderne data, clearly shows that the proposed algorithm is fully capable of solving problems from the industry. Moreover also the time limit wherein solutions are obtained, makes it possible to incorporate the proposed algorithm in a VRP application.

Chapter 12

Discussion

Throughout this work, our focus has been to develop an algorithm, that can solve a variety of packing problems actually met by companies today. The proposed algorithm should furthermore be able to work in connection with a vehicle routing application. This is achieved with a flexible model and an efficient algorithm.

In this chapter the different results obtained with our algorithm will be discussed. We will especially focus on how the algorithm can be used by Transvision and the advantages it will offer the two companies, Aarstiderne and Johannes Fog. Finally we will discuss in which areas future work could be focussed.

An effective algorithm depends on the model it is based upon. In our case it was not trivial to model the interaction between the different entities and their properties. Therefore, much effort has been put into the model, compared to the algorithmic part, as reflected in the focus of this work. Looking at the specific industry problems, the need for emphasis on the model was clearly seen. Otherwise, no solutions, which are feasible in a real world scenario, could be found.

12.1 Industrial application

12.1.1 Transvision

The requirements to the algorithm were given by Transvision, so that it would be relevant to use in a setup together with the Fleet Planner (their vehicle routing application). These were, to make a general CLP algorithm, which caters for multi-drop constraints and is able to find solutions fast. These requirements are met by the proposed algorithm. Smaller changes can, however, be expected when applying the method to specific companies. This could either be changes to the model or the algorithm. An important feature of the suggested framework, is that new greedy methods, taking advantage of problem specific characteristics, can easily be incorporated in the existing tree search. Of course the possibility to fine-tune the already existing methods to a specific case, is also present.

As our studies of the Johannes Fog and Aarstiderne problems show, the algorithm can, within fractions of a second, find feasible solutions to real life packing problems. These results are very promising, and show that there is a potential in using the algorithm in combination with the Fleet Planner. This would give Transvision an improved product and open up the marked of customers, where packing and routing must be combined before route planning will have any real value. This is certainly the case for builders merchants like Johannes Fog, but it is also relevant in other fields.

A necessary step, however, is to test how the algorithm will perform in combination with the Fleet Planner. More studies are needed to describe this interaction, so that questions, like how many times the CLP must be solved in the combined optimisation process, can be answered. The actual setup must of course be considered very carefully. To save time, the CLP should only be solved when there is need for it. In the case of Aarstiderne, it was found, that whenever the box/-container ratio was less than 70%, solutions were found trivially. Moreover, when the ratio was above 80%, solutions containing all boxes were rarely found. Therefore it should be considered whether routes with a ratio over 80%, should simply be discarded or checked, in the hope of finding a feasible solution, where all boxes are loaded. This indicate that the use of problem specific knowledge in the combined optimisation process, can be useful, when deciding how the interaction between the VRP and the CLP is constructed. As the CLP is not a simple check it should only be done in the most necessary cases. This would, for instance, be, when a new best, but not necessarily loading feasible, solution is found, in the VRP.

When a mathematical model is used as decision support in a company, high

standards of data quality is required. The company must by able to supply all necessary input to the model. In our model, knowledge about the different attributes on the boxes and container should be available. For some companies this is not a trivial, as discussed in the case of Johannes Fog. This should be considered when choosing a target audience for the application.

12.1.2 Aarstiderne

Today the vehicles handled by Aarstiderne are maximally half full when delivering their goods. This happens when around 180 boxes are distributed on the same route. When 180 boxes are present in one consignment, it is not possible to add more customers to the route without violating other constraints in the vehicle routing problem. This is constraints such as the drivers working hours and mileage. When the boxes are very alike in size, the multi-drop constraint is not very binding. Thus, given the vehicles used today, the loading constraint is not binding for Aarstiderne. To lower the cost of the distribution, smaller vehicles could be used. We found, that whenever the box/container ratio is below 70%, feasible loads could easily be found. When this measure get over 80%, the problems get harder to solve and one can generally not count on finding feasible loads. Investing in delivery trucks with a 30% smaller container, would still allow for a trouble free loading process. If smaller vehicles are used, savings would be available concerning both petrol usage and possibly a reduction in the vehicle license taxes.

A concern about the solutions found, is that the same product type can be loaded in all corners of the container. This greatly increase the time used to load the vehicles compared to how it is handled today, where each stack most often contain the same type of boxes. Before the found solutions can be used without revision from the drivers, this issue should be addressed. This could be done by including a secondary objective. Another obstacle is the fact, that many of the vehicles from Aarstiderne have doors on the sides, which gives some extra possibilities when unloading the vehicles. These issues are further considered in Section 12.3.

12.1.3 Johannes Fog

In the planning of routes for Johannes Fog, the packing constraint is often binding. This means that the routes cannot be made, without at the same time having an idea of, whether or not the corresponding loads can be constructed. Today this is done manually. We have shown that it is possible to use our

algorithm to solve the complicated packing problem. Furthermore it has been shown, that the problem complexity depends both on the combination of products packed and the total volume of the products. What makes the combination of products important, is that they have very different characteristics. Therefore, when the boxes are stacked, both the load bearing strength of the products and the multi-drop constraint, must be considered.

To test the algorithms ability to find realistic solutions, new consignments were constructed. All the solutions found to these problems are also valid in real life. This shows, that the model is constrained enough to handled real life situations, and does not produce solutions which are not actually possible. In general, the model is to strict compared to the real world scenario. Looking at the loads as they are driven by Johannes Fog today, boxes are often allowed to tilt a bit and if necessary overhang the back and the sides of the truck. These situations are totally prohibited in our solutions. Also in general, there is no need for the multi-drop constraint to be equally strict on all boxes. Small bags of rockwool for instance, could easily be rearranged when a customer is visited and other boxes are dropped of. This introduces an idea of allowing to rearrange the boxes, as discussed in Section 12.3.

Our test shows, that the algorithm is able to solve the type of problems faced by Johannes Fog. However, before a routing/packing tool can be considered it is also necessary that Johannes Fog is ready to use such an application. As mentioned in Section 9.3, the only information available to the planners at Johannes Fog, besides what they know by experience, is what is shown on the order sheets (Appendix B). Here, often no information about the specific dimensions of the products are available. In praxis orders are often bundled together before the trucks are loaded. This means that a standard should be introduced regarding which entities can be bundled together, and how, which should be available electronically. If, for instance, a customer request 50 gypsum boards it becomes important if they are all bundled together or if they are split in smaller packages.

12.2 Academic results

The tests made on the academic problems from the OR-Library had several purposes. Compared to the industrial data, these data sets are very large, and none of the problems have been proven to be solved to optimality.

The large amount of data gave us an opportunity to test the effect of the multi-drop constraint. A varying number of customers were assigned to the problems,

and tests clearly showed that as more customers are added, the found volume utilisation drops. The trend is only weak when few box types are present, but very strong as more box types are mixed.

Secondly, a test was made to the problems where only one customer was considered. This gave us some results that could be compared to work done be other researchers. The algorithm did not outperform the best algorithms available today, but it did provide comparable results, even using less time. The results show, that even though we have focused on making a framework that can handle a wide variety of problems, it can still compete with more specialised methods, considering academic problems.

12.3 Future work

As we have worked on this thesis, it has been an important part of the process to choose the focus of our work. Given only a limited amount of time, not all possible solution procedures can be tried and not all ideas can be carried out. In this section we will give a short overview of, in which direction future work, with basis in the subject studied in this thesis, could take.

The most obvious and possible also the most exciting extension to our algorithm, would be to incorporate it with a vehicle routing application. Only by doing this, one would be able to truly measure the proposed algorithms capability to work in the desired context. The most obvious way of doing this would be to incorporate it with the Fleet Planner, as proposed in Chapter 1. An interesting idea would be to allow rehandles when reaching the customer. This could very well be less time consuming from the drivers point of view than driving farther or splitting the load. A way to handle, this could be to relax the multi-drop constraints and punish rehandles at customers in the objective function.

12.3.1 Improvements of the model

A concern about the solutions found by the algorithm, is that no guarantee to the weight distribution can be given. When solving problems from Aarstiderne, this is no problem, because all boxes have just about the same weight. It is a more serious matter, when problems from Johannes Fog are solved. Here the products have very different weight characteristics. When this matter was discussed with the drivers at Johannes Fog, we learned that the weight distribution along the length of the container is most important. A given amount of the overall weight

should be put on the steering axis, which is the part of the container closest to the driver's cabin. If rockwool is loaded on the steering axis and gypsum boards in the back, the truck will have very bad handling properties. To solve the weight distribution issue, one should have in mind, that the load must be valid on the entire route, meaning that when a customer have been visited and products dropped off, the weight distribution of the remaining products should still be valid. Instead of incorporating the weight distribution in the model, another possibility is to add a secondary objective, favouring the desired weight distribution.

Concerning the multi-drop constraints, trucks often have doors on the sides of the container, that are too small to be modelled as whole surfaces. Such doors have not been considered in this work. In neither a standard container nor the trucks of Johannes Fog, this is needed. Here, rather whole surfaces can be used when unloading, making our model appropriate. However, if smaller doors are introduced as valid unloading places, the concept of a passage to a box should probably be changed. In general, there are many issues concerning this concept. When packing for Aarstiderne, feasible loads with 2 metres long very narrow passages could be build, where the next box to be unloaded is the box at the end of the passage. This box is, however, not practically unloadable. Moreover, when small boxes are loaded, like in the case from Aarstiderne, it is in praxis possible to walk around other boxes to find the desired box inside the container. This as well calls for a new way to handle the multi-drop constraints, and more precisely the passage concept. One way could be to find a passage to the box through sufficiently large empty spaces.

As well as a refinement of the multi-drop constraints could be in place, also a refinement to the concept of overhanging boxes could be necessary. In our model, all boxes are allowed to overhang with the same amount. In a practical setting this is not always the case. If we consider the consignments from Johannes Fog, then the very tall rockwool boxes probably needs a larger proportion of support from below, than the relative low gypsum boards. This is because the centre of gravity is placed higher for the rockwool than for the gypsum. A possible way to handle, this is to make the empty spaces as large as possible, neglecting the demand for support. Then it should be checked if boxes are properly supported, before they are placed.

The concept of load bearing strength has been incorporated in the model with success. However, it should be considered how the concept can be more integrated in the model. For instance, by deleting spaces on top of boxes, which cannot carry any of the remaining boxes. This would improve the performance of the algorithm. Further studies considering overhang in combination with load bearing strength, should be done to reveal, whether the model describes real life situations close enough.

12.3.2 Improvements of the algorithm

As mentioned before, the purpose of this work has been to develop an algorithm, that performs overall good to problems with different individual characteristics. Better results to the individual problems could obviously be found if many different specialised algorithms are constructed. When solving problems for Aarstiderne, it properly could be utilised that almost all boxes have the same base dimension. It could for instance be an idea to use an approach like the one by Gehring and Bortfeldt [13], where box stacks are created "on the run" leaving only a two dimensional packing problem. This approach could also, to some extent, be used to group the different product in the container.

When an optimal load is found by the proposed method, the procedure stops. This is independent of how the load is arranged. Often, many different loads, which are all optimal, can be found. However, in the eyes of the drivers, two loads are seldom equally good. To compare loads containing the same boxes a secondary objective could be introduced. When an optimal load is found, it could be improved by re-optimising according to another objective. In connection with Aarstiderne it could be to gather products of the same type closer together, which would lower the loading time at the depot. For Johannes Fog, it could be to lower the highest placed product, which would increase the stability of the load. Depending on how binding a weight distribution constraint is imposed on the problem, this could also be handled by a secondary objective, as proposed by Davies and Bischoff [7].

12.3.3 Communicating the solutions

An important issue is how to explain to the personnel, which are to actually load the vehicles, how to pack according to the solutions found. If there is enough spare room in the container, this is often not necessary, as the packing personnel often have a lot of experience. But if each box must be placed exactly as the solution suggests, some guidelines are necessary. It is possible to make lists, showing in which sequence the products should be loaded, but with 200 boxes to load, some coordination between how the boxes can be distinguished and what is written on the packing list, is needed. Alternatively, small animations on how the containers must be loaded could be shown on some hand-held devices, available to the packing personnel.

Chapter 13

Conclusion

In this thesis, we have developed an algorithm to solve the container loading problem (CLP) with multi-drop constraints. The algorithm is capable of solving a wide variety of packing problems met by companies today. It is furthermore possible to incorporate the algorithm in a framework used to solve the three dimensional loading capacitated vehicle routing problem (3L-CVRP). If this is done, the algorithm will perform loading feasibility checks on routes proposed by a vehicle routing problem (VRP) application.

The basis for the developed algorithm, is a highly detailed model, which makes it possible to guarantee that loads found by the algorithm, are realistic in a real world scenario. The model has been introduced in a realistic form, that afterwards was specialised to accommodate more specific needs in relation to the two Danish companies Aarstiderne and Johannes Fog. The realistic model offers the framework to handle three dimensional objects and allows boxes to be manipulated in a container. Besides this, all boxes placed will be multi-drop feasible and fully supported from below. The realistic model furthermore makes sure that fragile boxes are placed safely and the rotation restriction of the boxes are respected.

The model developed to describe problems from Aarstiderne differs only slightly from the realistic model. The boxes used by Aarstiderne can only be stacked if there is support in all four corners. This was ensured by a minor change to the

check performed before a box can be placed.

In the case of Johannes Fog, more severe changes were needed. Many of the assumptions in the realistic model are simply not reasonable, when construction products must be packed. In this specialised model, boxes can be unloaded from three sides, thereby changing the multi-drop considerations. Secondly, the model allows the boxes to overhang other boxes below, still guaranteeing a certain degree of support. Furthermore the model allows for a graduation of the fragility of boxes. This is possible by assigning a load bearing strength to each box. With all these changes incorporated into the model, it is possible to resemble the reality, seen at Johannes Fog, very closely. Even though this model is made as a specialisation of the realistic model, it should rather be seen as a generalisation. All the changes in the model adds further flexibility and a greater sense of reality.

To solve the problems, a tree search heuristic and five greedy methods have been developed. The tree search can be used in combination with any of the greedy methods. It is possible to accelerate the search, going faster down the nodes in the tree, based on an analysis of the greedy solutions. Furthermore a dynamic breadth strategy allows the algorithm to adjust the breadth of the search tree depending on characteristics of the specific problem and the time limit imposed.

The algorithms have been tested on three different dataset. In the early stages of the tuning and testing, theoretical data was used to choose the best choice of greedy method and setting for the tree search. One generally good solution method was found, based on the problems that could be modelled with the realistic model. The final algorithm is based on the tree search with dynamic breadth, and uses a fast first fit greedy method. We found that the suggested dynamic breadth strategy is an improvement to the algorithm. However, accelerating the search is only lucrative, for problems with one customer.

The algorithm and the realistic model was tested on theoretical benchmark problems, to allow a comparison with algorithms developed others. This was, however, only possible on data without multi-drop considerations. The results showed that, although our algorithm is not as good as the best developed for these problems, the performance is still comparable. Tests were also performed where customers were added to the problems. This illustrated the extra complexity added by the multi-drop constraint.

The algorithm and the specialised models, were tested on real life problems, from the two companies Aarstiderne and Johannes Fog. The results, of the test on the data from Aarstiderne, revealed that as the loads are driven today, the loading constraint is never binding, when considering the overall 3L-CVRP. Further experiments were conducted, where the space in the container was reduced.

This showed that the algorithm found optimal solutions to all problems, when the volume utilisation of the container was below 70%. For problems with a box/container ratio of up to 80%, optimal solutions can be found, within a time limit of 10 seconds.

The test of the problems from Johannes Fog showed, that all the original problems could be solved to optimality within 0,01 seconds. Based on the new generated problems it was found, that the box/container ratio is not an appropriate measure for the problem complexity. The combination of the load bearing strength and multi-drop constraints complicates the problems. The overall results, however, are very promising as all the loads found, are also feasible in a real world scenario.

Appendix A

List of Symbols

Symbols used with the boxes

$(\Delta a_i, \Delta b_i, \Delta c_i)$	The extension of box $i \in B_{placed}$ in the x-, y- and Z-direction, respectively
$\mathcal{B}$	A set holding for every box type bt present in a batch the number of boxes with this box type
B	The set of all boxes
$B_{nPlaced}$	The set of *not* placed boxes
B_{placed}	The set of placed boxes
$bs \in BS$	The set of box stacks BS
bt_i	The box type of box i
d_i	The sequence number of box i
f_i	1 if box i is fragile, 0 otherwise
fr	The rotation of a box
FR_i	The set of feasible rotation for box i

i, j, k	Index variables for boxes
k_{bt}	Index in $\mathcal{B}$
LBS_i	The load bearing strength of box i
m_i	The mass of box i
nC	The number of different customers in B
S_i^{area}	The supported area of box $i \in B_{placed}$
sm	The smallest dimension among all boxes in a batch
x^+, y^+, y^-	Boolean variables indicating whether the box can be unloaded from the surfaces $(L,0,0)-(L,W,Z)$, $(0,W,0)-(L,W,Z)$ and $(0,0,0)-(L,0,Z)$, respectively
$(\overline{a}_i, \overline{b}_i, \overline{c}_i)$	The maximum of a box.
$(\underline{a}_i, \underline{b}_i, \underline{c}_i))$	The minimum of a box.
(a_i, b_i, c_i)	Continuous variables indicating the location of box i, along the x-, y- and z- dimension respectively.
l_i, w_i, h_i	Parameters indicating the length, width and height of box i
N	The total number of boxes in the problem
$\ell_i^x, \ell_i^y, \ell_i^z$	Binary variables in the MIP model indicating whether the length of box i is parallel to the x- ($\ell_i^x = 1$), y- ($\ell_i^y = 1$) or z-axis ($\ell_i^z = 1$)
w_i^x, w_i^y, w_i^z	Binary variables in the MIP model indicating whether the width of box i is parallel to the x- ($w_i^x = 1$), y- ($w_i^y = 1$) or z-axis ($w_i^z = 1$)
h_i^x, h_i^y, h_i^z	Binary variables in the MIP model indicating whether the height of box i is parallel to the x- ($h_i^x = 1$), y- ($h_i^y = 1$) or z-axis ($h_i^z = 1$)
le_{ik}	Binary variable in the MIP model indicating if box i is placed on the left side of box k
ri_{ik}	Binary variable in the MIP model in the MIP model indicating if box i is placed on the right side of box k
be_{ik}	Binary variable in the MIP model indicating if box i is placed behind box k

fr_{ik}	Binary variable in the MIP model indicating if box i is placed in front of box k
ab_{ik}	Binary variable in the MIP model indicating if box i is placed above box k
un_{ik}	Binary variable in the MIP model indicating if box i is placed underneath box k
p_i	Binary variable in the MIP model indicating if box i is placed in the container

Symbols used with the container and empty spaces

(L, W, H)	The container dimensions
α	Support from below in a space in the x-dimension
β	Support from below in a space in the y-dimension
γ	The overhang factor when boxes are allowed to overhang one another
$\mathcal{S}^x$	The maximum supported area of a space in the x-dimension
$\mathcal{S}^y$	The maximum supported area of a space in the y-dimension
S	The set of empty spaces
s, t, r	Index variables for empty spaces
X^+	Boolean parameter indicating if the container can be loaded/unloaded from the surface given by $(L, 0, 0) - (L, W, Z)$
Y^+	Boolean parameter indicating if the container can be loaded/unloaded from the surface given by $(0, W, 0) - (L, W, Z)$
Y^-	Boolean parameter indicating if the container can be loaded/unloaded from the surface given by $(0, 0, 0) - (L, 0, Z)$
$(\Delta x_s, \Delta y_s, \Delta z_s)$	The size of an empty space
$(\overline{x}_s, \overline{y}_s, \overline{z}_s)$	The maximum of an empty space
$(\underline{x}_s, \underline{y}_s, \underline{z}_s)$	The minimum of an empty space

Symbols used in the algorithms

ψ	Parameter to change the effect of the weight in the dynamic tree search.
κ	The minimum number of boxes to pack at a time in the greedy algorithms
λ	The breadth of the tree in the tree-search algorithm
$K^{*1}_{bt_i,s}$	The number of boxes $i \in B_{nPlaced}$ with box type bt that can be placed in a stack in empty spaces s
$K^{*2}_{bt_i,s}$	The number of boxes $i \in B_{nPlaced}$ with box type bt that can be placed in the empty spaces s

Other symbols

$\square\!\!\square$	Binary operator denoting that cuboids are overlapping or leading up to each other in all 3 dimensions
$\square\!\!\square_x, \square\!\!\square_y, \square\!\!\square_z$	Binary operator denoting that cuboids are overlapping or leading up to each other in respectively x-, y- or z-dimension
$\boxdot$	Binary operator denoting that cuboids are overlapping in all three dimensions
$\boxdot_x, \boxdot_y, \boxdot_z$	Binary operators denoting that cuboids are overlapping in respectively x-, y- or z-dimension
M	Large number used in big-M constraints in the MIP model

Appendix B

Order sheets from Johannes Fog

This is an example of the order sheets used by Johannes Fog. This is the only information used by the planners today, to make the consignments and routes for all the vehicles handled by Johannes Fog. There is no information about how the different products are bundled, on the order sheets. In the example from this order the following entities are packed together: "TAGLÆGTER" and "FYR SEKSTA TRYKIMP." giving a package with dimensions $540cm \times 100cm \times 45cm$ and "VTA FYR", "REGLER" and "FORSKALLING" giving the dimensions $480cm \times 110cm \times 65cm$. As it is today, the planning personnel have no influence on how the drivers actually pack the different entities together. From page 3 of the order (Figure B) 3 entities with "ISOVER" is requested.

Johannes Fog Trælast

Fog®

[redacted]
2600 GLOSTRUP

Side 1
Fakturadato 19-12-2006
Sagsnummer
Kundenummer [redacted]
Vor reference

FØLGESEDDEL

Leveres til
[redacted]
2820 GENTOFTER TUR 2
[redacted]

Levering 20-12-2006 LEVERET LAGER
Deres ref. [redacted]
Ordre dato 19-12-2006

Varebetegnelse	Leveret	Enh	Priserne er excl. moms	Beløb
3810 3553054 HUNTONIT UNDERTAG TRÆFIBERPL. OVERFLADEBEH. 160X210CMX3MM PK A 75 PL	80 268,8	STK M2	DV37001	
1634 038073 TAGLÆGTER T1 EGAL. 38 X 73 MM 120/5,00	600,000	M		
1086 025050 FYR SEKSTA TRYKIMP. 25X50 MM IMPRÆG. KLASSE AB I.H.T NTR 56/5,40	302,400	M	G03 005	
4610 1029693 ICOPAL BUILD 310 C (THOR 6) 1 X 20 M 45 RL PR PALLE	10	RL	BH09001	
1305 050125 VTA FYR 50 X 125 MM 36/4,20	151,200	M	E38	
1437 047100 REGLER 43 X 95 MM 43/3,60	154,800	M	F36	
1007 019100 FORSKALLING 19 X 100 MM 105/4,80	504,000	M	F7	

Transport 0,00

Form nr. 20

ANCO Herlev, Lyskær 2, 2730 Herlev, Tlf.: 44 94 40 66, Fax: 44 84 22 84
Asminderød Savværk & Trælast, Kongevejen 25, 3480 Fredensborg, Tlf.: 48 48 00 13, Fax: 48 48 25 45
Ejby Trælast, Ejby Industrivej 72, 2600 Glostrup, Tlf.: 43 96 23 22, Fax: 43 43 31 12
Farum Tømmerhandel Johannes Fog Byggecenter, Farum Gydevej 59, 3520 Farum, Tlf.: 44 95 01 44, Fax: 44 95 50 64
Hørsholm Trælasthandel Johannes Fog Byggecenter, Kokkedal Industripark 15, 2970 Hørsholm, Tlf.: 45 86 10 10, Fax: 45 86 63 97
Kvistgård Trælasthandel, Egeskovvej 1, 3490 Kvistgård, Tlf.: 49 12 50 50, Fax: 49 13 80 37
Johannes Fog Trælast, Firskovvej 20, 2800 Lyngby, Tlf.: 45 87 10 01, Fax: 45 93 39 83
Johannes Fog Byggecenter, Nørgårdsvej 3, 2800 Lyngby, Tlf.: 45 87 10 01, Fax: 45 87 50 51

Administration: Johannes Fog A/S - Firskovvej 20 - 2800 Lyngby - Tlf.: 45 87 10 01 - Fax: 45 87 12 83 - CVR-nummer: 16314439 - Bank: Nordea Bank Danmark A/S

www.johannesfog.dk

Figure B.1: Order sheet used by Johannes Fog today - page 1. Problem $Rum3\_1 - 2006 - 12 - 19$.

Johannes Fog Trælast

Fog®

[redacted]
2600 GLOSTRUP

Side 2
Fakturadato 19-12-2006
Sagsnummer
Kundenummer [redacted]
Vor reference

FØLGESEDDEL

Levering
20-12-2006 LEVERET LAGER

Deres ref. [redacted]

Ordre dato
19-12-2006

Varebetegnelse	Leveret	Enh	Priserne er excl. moms	Beløb
			Transport	0,00
3610 5708463			E002018	
DANOGIPS VINDTÆT E3 (RV)	67	STK		
90 X 250 CM X 9 MM				
40 PR BDT	150,75	M2		
3820 9000057			DV18001	
S.Y.P. X-FINER B/C	10	PL		
244 X 122 CM X 18 MM	29,76	M2		

50X75 SEKSTA I RESTORDRE

***

I alt vægt i KG 3470,702 I alt rumfang i M3

De blev betjent af
TØMMERHANDEL FIRSKOVVEJ
NADHEM ZID LOKAL 264

Form nr. 20

AMCO Herlev
Lyskær 2
2730 Herlev
Tlf.: 44 94 40 66
Fax: 44 84 22 84

Asminderød Savværk & Trælast
Kongevejen 25
3480 Fredensborg
Tlf.: 48 48 00 13
Fax: 48 48 25 45

Ejby Trælast
Ejby Industrivej 72
2600 Glostrup
Tlf.: 43 96 23 22
Fax: 43 43 31 12

Farum Tømmerhandel
Johannes Fog Byggecenter
Farum Gydevej 59
3520 Farum
Tlf.: 44 95 01 64
Fax: 44 95 50 64

Hørsholm Trælasthandel
Johannes Fog Byggecenter
Kokkedal Industripark 15
2970 Hørsholm
Tlf.: 45 86 10 10
Fax: 45 86 63 97

Kvistgård Trælasthandel
Egeskovvej 1
3490 Kvistgård
Tlf.: 49 12 50 50
Fax: 49 13 80 37

Johannes Fog Trælast
Firskovvej 20
2800 Lyngby
Tlf.: 45 87 10 01
Fax: 45 93 39 83

Johannes Fog Byggecenter
Nørgårdsvej 3
2800 Lyngby
Tlf.: 45 87 10 01
Fax: 45 87 50 51

Administration: Johannes Fog A/S - Firskovvej 20 - 2800 Lyngby - Tlf.: 45 87 10 01 - Fax: 45 87 12 83 - CVR-nummer: 16314439 - Bank: Nordea Bank Danmark A/S

www.johannesfog.dk

Figure B.2: Order sheet used by Johannes Fog today - page 2. Problem $Rum3\_1 - 2006 - 12 - 19$.

Johannes Fog Trælast

Fog®

[redacted]
2600 GLOSTRUP

Side 1
Fakturadato 19-12-2006
Sagsnummer
Kundenummer [redacted]
Vor reference

KOPI

FØLGESEDDEL

Leveres til [redacted]

Levering
20-12-2006 LEVERET LAGER *

Deres ref. [redacted]

Ordre dato
19-12-2006

Varebetegnelse	Leveret	Enh	Priserne er excl. moms	Beløb
4510 5019693			C007004	
ISOVER FLEX Q-BIG	25	PK		
120X600X920MM				
25 PAKKER PR PALLE	110,5	M2		
4510 5019706			C007002	
ISOVER FORMSTYKKE Q-BIG	35	PK		
70X600X900MM				
25 PK PR PALLE	264,6	M2		
4510 5019688			C007003	
ISOVER FLEX Q-BIG	50	PK		
95X600X920MM				
25 PK PR PALLE	276	M2		

I alt vægt i KG

I alt rumfang i M3

De blev betjent af
TØMMERHANDEL FIRSKOVVEJ
NADHEM ZID LOKAL 264

Form nr. 23

ANCO Herlev
Lyskær 2
2730 Herlev
Tlf.: 44 94 40 66
Fax: 44 84 22 84

Asminderød Savværk & Trælast
Kongevejen 25
3480 Fredensborg
Tlf.: 48 48 00 13
Fax: 48 48 25 45

Ejby Trælast
Ejby Industrivej 72
2600 Glostrup
Tlf.: 43 96 23 22
Fax: 43 43 31 12

Farum Tømmerhandel
Johannes Fog Byggecenter
Farum Gydevej 59
3520 Farum
Tlf.: 44 95 01 64
Fax: 44 95 50 64

Hørsholm Trælasthandel
Johannes Fog Byggecenter
Kokkedal Industripark 15
2970 Hørsholm
Tlf.: 45 86 10 10
Fax: 45 86 63 97

Kvistgård Trælasthandel
Egeskovvej 1
3490 Kvistgård
Tlf.: 49 12 50 50
Fax: 49 13 80 37

Johannes Fog Trælast
Firskovvej 20
2800 Lyngby
Tlf.: 45 87 10 01
Fax: 45 93 39 83

Johannes Fog Byggecenter
Nørgårdsvej 3
2800 Lyngby
Tlf.: 45 87 10 01
Fax: 45 87 50 51

Administration: Johannes Fog A/S - Firskovvej 20 - 2800 Lyngby - Tlf.: 45 87 10 01 - Fax: 45 87 12 83 - CVR-nummer: 16314439 - Bank: Nordea Bank Danmark A/S

www.johannesfog.dk

Figure B.3: Order sheet used by Johannes Fog today - page 3. Problem $Rum3\_1 - 2006 - 12 - 19$.

Appendix C

Problems from Johannes Fog

C.1 Original consignments

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	1	$114 \times 112 \times 244$	$1 \times 0 \times 0$	1.652	0.04	Wood
2	4	$90 \times 85 \times 120$	$1 \times 0 \times 0$	501	-1	Concrete/gravel
2	1	$55 \times 80 \times 120$	$1 \times 0 \times 0$	1.000	-1	Concrete/gravel
2	1	$35 \times 105 \times 540$	$1 \times 0 \times 0$	500	0.03	Wood
2	1	$40 \times 92 \times 218$	$1 \times 0 \times 0$	535	0.03	Wood
2	1	$50 \times 30 \times 43$	$1 \times 1 \times 1$	1	-1	Cardboard boxes

** The capacity of the truck*

Table C.1: $OV96581\_1 - 2006 - 12 - 14$

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	6.700 *	-	The truck
1	1	$110 \times 112 \times 520$	$1 \times 0 \times 0$	3.162	0.04	Wood
1	1	$45 \times 60 \times 90$	$1 \times 1 \times 1$	5	-1	Rockwool
2	1	$55 \times 119 \times 252$	$1 \times 0 \times 0$	58	-1	Eternite

** The capacity of the truck*

Table C.2: $PC90153\_1 - 2006 - 12 - 14$

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$605 \times 250 \times 280$	-	6.700 *	-	The truck
1	2	$90 \times 85 \times 120$	$1 \times 0 \times 0$	500	-1	Concrete/gravel
1	1	$50 \times 120 \times 80$	$1 \times 0 \times 0$	1.000	-1	Concrete/gravel
						** The capacity of the truck*

Table C.3: $RV92760\_1 - 2006 - 12 - 14$

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	1	$56 \times 360 \times 90$	$1 \times 0 \times 0$	800	0.03	Gypsum boards
1	1	$14 \times 208 \times 12$	$1 \times 0 \times 0$	25	-1	Wood
1	1	$25 \times 16 \times 21$	$1 \times 1 \times 1$	27	-1	Cardboard boxes
1	3	$45 \times 60 \times 90$	$1 \times 1 \times 1$	0	0	Rockwool
2	1	$31 \times 360 \times 97$	$1 \times 0 \times 0$	500	0.03	Gypsum boards
2	1	$4 \times 204 \times 83$	$1 \times 0 \times 0$	25	0.01	Wood
2	1	$9 \times 361 \times 19$	$1 \times 1 \times 1$	10	0.02	Wood
2	1	$14 \times 208 \times 12$	$1 \times 0 \times 0$	7	-1	Wood
2	1	$25 \times 16 \times 21$	$1 \times 1 \times 1$	0	-1	Cardboard boxes
2	2	$45 \times 60 \times 90$	$1 \times 1 \times 1$	0	0	Rockwool
3	3	$51 \times 240 \times 90$	$1 \times 0 \times 0$	800	0.03	Gypsum boards
3	1	$25 \times 240 \times 90$	$1 \times 0 \times 0$	300	0.03	Gypsum boards
3	1	$8 \times 360 \times 19$	$1 \times 0 \times 1$	36	-1	Gypsum bars
3	1	$8 \times 360 \times 60$	$1 \times 0 \times 0$	91	0.015	Gypsum bars
3	3	$45 \times 60 \times 90$	$1 \times 1 \times 1$	0	0	Rockwool
						** The capacity of the truck*

Table C.4: $Rum2\_1 - 2006 - 12 - 14$

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	2	$95 \times 120 \times 60$	$1 \times 1 \times 1$	5	-1	Rockwool
1	1	$90 \times 30 \times 30$	$1 \times 1 \times 1$	5	-1	Rockwool
1	1	$51 \times 300 \times 90$	$1 \times 0 \times 0$	900	0.04	Gypsum boards
1	1	$15 \times 360 \times 90$	$1 \times 0 \times 0$	300	0.02	Gypsum bars
1	1	$24 \times 16 \times 23$	$1 \times 1 \times 1$	4	-1	Cardboard boxes
1	1	$13 \times 23 \times 13$	$1 \times 1 \times 1$	10	-1	Cardboard boxes
2	1	$31 \times 90 \times 240$	$1 \times 0 \times 0$	486	0.04	Gypsum boards
						** The capacity of the truck*

Table C.5: $OV96581\_1 - 2006 - 12 - 19$

C.2 Generated consignments

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	1	$48 \times 395 \times 105$	$1 \times 0 \times 0$	1.243	0.04	Wood
2	1	$134 \times 100 \times 100$	$1 \times 0 \times 0$	1.118	-1	Concrete/gravel
3	1	$42 \times 445 \times 105$	$1 \times 0 \times 0$	10	0.04	Wood
3	1	$24 \times 22 \times 16$	$1 \times 1 \times 1$	4	-1	Cardboard boxes
3	1	$23 \times 100 \times 45$	$1 \times 1 \times 1$	20	0.04	Cover board
4	1	$21 \times 430 \times 110$	$1 \times 0 \times 0$	330	0.04	Wood
4	1	$105 \times 120 \times 80$	$1 \times 0 \times 0$	246	-1	Concrete/gravel
4	1	$55 \times 120 \times 80$	$1 \times 0 \times 0$	435	-1	Wood
						* *The capacity of the truck*

Table C.6: $PC90355\_1 - 2006 - 12 - 19$

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	4	$256 \times 120 \times 120$	$1 \times 0 \times 0$	10	-1	Rockwool
1	1	$90 \times 120 \times 120$	$1 \times 0 \times 0$	5	-1	Rockwool
1	1	$115 \times 120 \times 80$	$1 \times 0 \times 0$	50	-1	Rockwool
1	1	$26 \times 160 \times 210$	$1 \times 0 \times 0$	500	0.1	Wood
1	1	$19 \times 122 \times 244$	$1 \times 0 \times 0$	500	0.04	Wood
1	1	$72 \times 250 \times 90$	$1 \times 0 \times 0$	800	0.03	Gypsum boards
1	1	$45 \times 100 \times 540$	$1 \times 0 \times 0$	1.000	0.04	Wood
1	1	$65 \times 110 \times 480$	$1 \times 0 \times 0$	1.000	0.04	Wood
						* *The capacity of the truck*

Table C.7: $Rum3\_1 - 2006 - 12 - 19$

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	3	$58 \times 240 \times 90$	$1 \times 0 \times 0$	777	0.1	Gypsum boards
1	1	$16 \times 360 \times 90$	$1 \times 0 \times 1$	193	0.02	Gypsum bars
1	4	$53 \times 60 \times 100$	$1 \times 1 \times 1$	3	0.0002	Rockwool
2	2	$113 \times 100 \times 100$	$1 \times 0 \times 0$	900	0.1	Concrete/gravel
3	1	$70 \times 120 \times 80$	$1 \times 0 \times 0$	250	-1	Tiles
4	1	$29 \times 31 \times 38$	$1 \times 1 \times 1$	5	-1	Cardboard boxes
4	1	$83 \times 420 \times 95$	$1 \times 0 \times 0$	1.500	0.1	Wood
4	1	$65 \times 450 \times 112$	$1 \times 0 \times 0$	1.500	0.1	Wood
4	1	$35 \times 500 \times 85$	$1 \times 0 \times 1$	100	0.1	Wood
5	1	$16 \times 228 \times 60$	$1 \times 0 \times 1$	40	0.03	Wood
6	3	$265 \times 200 \times 120$	$1 \times 0 \times 0$	75	-1	Rockwool
						* *The capacity of the truck*

Table C.8: $Rum3\_1 - 2007 - 02 - 15$

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	3	$58 \times 240 \times 90$	$1 \times 0 \times 0$	777	0.1	Gypsum boards
1	1	$16 \times 360 \times 90$	$1 \times 0 \times 1$	193	0.02	Gypsum bars
1	4	$53 \times 60 \times 100$	$1 \times 1 \times 1$	1	0.0002	Rockwool
2	1	$114 \times 112 \times 244$	$1 \times 0 \times 0$	1.652	0.04	Wood
2	4	$90 \times 85 \times 120$	$1 \times 0 \times 0$	501	-1	Concrete/gravel
2	1	$55 \times 80 \times 120$	$1 \times 0 \times 0$	1.000	-1	Concrete/gravel
2	1	$35 \times 105 \times 540$	$1 \times 0 \times 0$	500	0.03	Wood
2	1	$40 \times 92 \times 218$	$1 \times 0 \times 0$	535	0.03	Wood
2	1	$50 \times 30 \times 43$	$1 \times 1 \times 1$	1	-1	Cardboard boxes
3	2	$95 \times 120 \times 60$	$1 \times 1 \times 1$	5	-1	Rockwool
3	1	$90 \times 30 \times 30$	$1 \times 1 \times 1$	5	-1	Rockwool
3	1	$51 \times 300 \times 90$	$1 \times 0 \times 0$	900	0.04	Gypsum boards
3	1	$16 \times 360 \times 90$	$1 \times 0 \times 0$	300	0.02	Gypsum bars
3	1	$24 \times 16 \times 23$	$1 \times 1 \times 1$	4	-1	Cardboard boxes
3	1	$13 \times 23 \times 13$	$1 \times 1 \times 1$	10	-1	Cardboard boxes
4	1	$31 \times 90 \times 240$	$1 \times 0 \times 0$	486	0.04	Gypsum boards
						** The capacity of the truck*

Table C.9: Generated 1

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	1	$16 \times 228 \times 60$	$1 \times 0 \times 1$	40	0.03	Wood
1	3	$265 \times 200 \times 120$	$1 \times 0 \times 0$	80	-1	Rockwool
1	1	$110 \times 112 \times 520$	$1 \times 0 \times 0$	3.162	0.04	Wood
1	1	$45 \times 60 \times 90$	$1 \times 1 \times 1$	5	-1	Rockwool
1	1	$55 \times 119 \times 252$	$1 \times 0 \times 0$	58	-1	Eternite
2	1	$48 \times 395 \times 105$	$1 \times 0 \times 0$	1.243	0.04	Wood
2	1	$134 \times 100 \times 100$	$1 \times 0 \times 0$	1.118	-1	Concrete/gravel
2	1	$42 \times 445 \times 105$	$1 \times 0 \times 0$	10	0.04	Wood
2	1	$24 \times 22 \times 16$	$1 \times 1 \times 1$	4	-1	Cardboard boxes
2	1	$23 \times 100 \times 45$	$1 \times 1 \times 1$	20	0.04	Cover board
2	1	$21 \times 430 \times 110$	$1 \times 0 \times 0$	330	0.04	Wood
2	1	$105 \times 120 \times 80$	$1 \times 0 \times 0$	246	-1	Concrete/gravel
2	1	$55 \times 120 \times 80$	$1 \times 0 \times 0$	435	-1	Wood
3	2	$90 \times 85 \times 120$	$1 \times 0 \times 0$	500	-1	Concrete/gravel
3	1	$50 \times 120 \times 80$	$1 \times 0 \times 0$	1.000	-1	Concrete/gravel
4	1	$56 \times 360 \times 90$	$1 \times 0 \times 0$	800	0.03	Gypsum boards
4	1	$14 \times 208 \times 12$	$1 \times 0 \times 0$	7	-1	Wood
4	1	$25 \times 16 \times 21$	$1 \times 1 \times 1$	27	-1	Cardboard boxes
4	3	$45 \times 60 \times 90$	$1 \times 1 \times 1$	0	0	Rockwool
4	1	$31 \times 360 \times 97$	$1 \times 0 \times 0$	500	0.03	Gypsum boards
4	1	$4 \times 204 \times 83$	$1 \times 0 \times 0$	25	0.01	Wood
4	1	$9 \times 361 \times 19$	$1 \times 1 \times 1$	10	0.02	Wood
4	1	$14 \times 208 \times 12$	$1 \times 0 \times 0$	7	-1	Wood
4	1	$25 \times 16 \times 21$	$1 \times 1 \times 1$	27	-1	Cardboard boxes
4	2	$45 \times 60 \times 90$	$1 \times 1 \times 1$	0	0	Rockwool
						** The capacity of the truck*

Table C.10: Generated 2

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	3	$58 \times 240 \times 90$	$1 \times 0 \times 0$	777	0.1	Gypsum boards
1	1	$16 \times 360 \times 90$	$1 \times 0 \times 1$	193	0.02	Gypsum bars
2	2	$113 \times 100 \times 100$	$1 \times 0 \times 0$	900	0.1	Concrete/gravel
3	1	$70 \times 120 \times 80$	$1 \times 0 \times 0$	250	-1	Tiles
4	1	$29 \times 31 \times 38$	$1 \times 1 \times 1$	80	-1	Cardboard boxes
4	1	$83 \times 420 \times 95$	$1 \times 0 \times 0$	1.500	0.1	Wood
4	1	$65 \times 450 \times 112$	$1 \times 0 \times 0$	1.500	0.1	Wood
5	1	$114 \times 112 \times 244$	$1 \times 0 \times 0$	1.652	0.04	Wood
6	1	$35 \times 105 \times 540$	$1 \times 0 \times 0$	500	0.03	Wood
6	1	$40 \times 92 \times 218$	$1 \times 0 \times 0$	535	0.03	Wood
6	1	$50 \times 30 \times 43$	$1 \times 1 \times 1$	1	-1	Cardboard boxes
7	1	$42 \times 445 \times 105$	$1 \times 0 \times 0$	10	0.04	Wood
7	1	$24 \times 22 \times 16$	$1 \times 1 \times 1$	4	-1	Cardboard boxes
7	1	$23 \times 100 \times 45$	$1 \times 1 \times 1$	20	0.04	Cover board
8	1	$21 \times 430 \times 110$	$1 \times 0 \times 0$	330	0.04	Wood
8	1	$105 \times 120 \times 80$	$1 \times 0 \times 0$	246	-1	Concrete/gravel
8	1	$55 \times 120 \times 80$	$1 \times 0 \times 0$	435	-1	Wood
9	2	$90 \times 85 \times 120$	$1 \times 0 \times 0$	500	-1	Concrete/gravel
						** The capacity of the truck*

Table C.11: Generated 3

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	2	$95 \times 120 \times 60$	$1 \times 1 \times 1$	5	-1	Rockwool
1	1	$90 \times 30 \times 30$	$1 \times 1 \times 1$	5	-1	Rockwool
1	1	$51 \times 300 \times 90$	$1 \times 0 \times 0$	900	0.04	Gypsum boards
1	1	$15 \times 360 \times 90$	$1 \times 0 \times 0$	300	0.02	Gypsum bars
1	1	$24 \times 16 \times 23$	$1 \times 1 \times 1$	4	-1	Cardboard boxes
1	1	$13 \times 23 \times 13$	$1 \times 1 \times 1$	10	-1	Cardboard boxes
2	1	$31 \times 90 \times 240$	$1 \times 0 \times 0$	486	0.04	Gypsum boards
3	1	$110 \times 112 \times 520$	$1 \times 0 \times 0$	3.162	0.04	Wood
3	1	$45 \times 60 \times 90$	$1 \times 1 \times 1$	5	-1	Rockwool
4	1	$55 \times 119 \times 252$	$1 \times 0 \times 0$	58	-1	Eternite
5	4	$256 \times 120 \times 120$	$1 \times 0 \times 0$	10	-1	Rockwool
5	1	$90 \times 120 \times 120$	$1 \times 0 \times 0$	5	-1	Rockwool
5	1	$115 \times 120 \times 80$	$1 \times 0 \times 0$	50	-1	Rockwool
5	1	$26 \times 160 \times 210$	$1 \times 0 \times 0$	500	0.1	Wood
5	1	$19 \times 122 \times 244$	$1 \times 0 \times 0$	500	0.04	Wood
5	1	$72 \times 250 \times 90$	$1 \times 0 \times 0$	800	0.03	Gypsum boards
5	1	$45 \times 100 \times 540$	$1 \times 0 \times 0$	1.000	0.04	Wood
5	1	$65 \times 110 \times 480$	$1 \times 0 \times 0$	1.000	0.04	Wood
						** The capacity of the truck*

Table C.12: Generated 4

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	720 × 250 × 280	-	9.500 *	-	The truck
1	1	114 × 112 × 244	1 × 0 × 0	1.652	0.04	Wood
2	1	55 × 80 × 120	1 × 0 × 0	1.000	-1	Concrete/gravel
2	1	35 × 105 × 540	1 × 0 × 0	500	0.03	Wood
2	1	40 × 92 × 218	1 × 0 × 0	535	0.03	Wood
2	1	50 × 30 × 43	1 × 1 × 1	1	-1	Cardboard boxes
3	2	95 × 120 × 60	1 × 1 × 1	5	-1	Rockwool
3	1	90 × 30 × 30	1 × 1 × 1	5	-1	Rockwool
3	1	51 × 300 × 90	1 × 0 × 0	900	0.04	Gypsum boards
3	1	15 × 360 × 90	1 × 0 × 0	300	0.02	Gypsum bars
3	1	24 × 16 × 23	1 × 1 × 1	4	-1	Cardboard boxes
3	1	13 × 23 × 13	1 × 1 × 1	10	-1	Cardboard boxes
4	1	31 × 90 × 240	1 × 0 × 0	486	0.04	Gypsum boards
5	1	55 × 119 × 252	1 × 0 × 0	58	-1	Eternite
6	4	256 × 120 × 120	1 × 0 × 0	10	-1	Rockwool
6	1	90 × 120 × 120	1 × 0 × 0	5	-1	Rockwool
6	1	115 × 120 × 80	1 × 0 × 0	50	-1	Rockwool
6	1	26 × 160 × 210	1 × 0 × 0	500	0.1	Wood
6	1	19 × 122 × 244	1 × 0 × 0	500	0.04	Wood
6	1	72 × 250 × 90	1 × 0 × 0	800	0.03	Gypsum boards
6	1	45 × 100 × 540	1 × 0 × 0	1.000	0.04	Wood
6	1	65 × 110 × 480	1 × 0 × 0	1.000	0.04	Wood
					** The capacity of the truck*	

Table C.13: Generated 5

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	720 × 250 × 280	-	9.500 *	-	The truck
1	1	114 × 112 × 244	1 × 0 × 0	1.652	0.04	Wood
2	1	55 × 80 × 120	1 × 0 × 0	1.000	-1	Concrete/gravel
2	1	35 × 105 × 540	1 × 0 × 0	500	0.03	Wood
2	1	40 × 92 × 218	1 × 0 × 0	535	0.03	Wood
2	1	50 × 30 × 43	1 × 1 × 1	1	-1	Cardboard boxes
3	2	95 × 120 × 60	1 × 1 × 1	5	-1	Rockwool
3	1	90 × 30 × 30	1 × 1 × 1	5	-1	Rockwool
3	1	51 × 300 × 90	1 × 0 × 0	900	0.04	Gypsum boards
3	1	15 × 360 × 90	1 × 0 × 0	300	0.02	Gypsum bars
3	1	24 × 16 × 23	1 × 1 × 1	4	-1	Cardboard boxes
3	1	13 × 23 × 13	1 × 1 × 1	10	-1	Cardboard boxes
4	1	31 × 90 × 240	1 × 0 × 0	486	0.04	Gypsum boards
5	1	55 × 119 × 252	1 × 0 × 0	58	-1	Eternite
6	4	256 × 120 × 120	1 × 0 × 0	10	-1	Rockwool
6	1	90 × 120 × 120	1 × 0 × 0	5	-1	Rockwool
6	1	115 × 120 × 80	1 × 0 × 0	50	-1	Rockwool
6	1	26 × 160 × 210	1 × 0 × 0	500	0.1	Wood
6	1	19 × 122 × 244	1 × 0 × 0	500	0.04	Wood
6	1	72 × 250 × 90	1 × 0 × 0	800	0.03	Gypsum boards
6	1	45 × 100 × 540	1 × 0 × 0	1.000	0.04	Wood
6	1	65 × 110 × 480	1 × 0 × 0	1.000	0.04	Wood
					** The capacity of the truck*	

Table C.14: Generated 6

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	3	$265 \times 200 \times 120$	$1 \times 0 \times 0$	80	-1	Rockwool
1	1	$35 \times 500 \times 85$	$1 \times 0 \times 1$	100	0.1	Wood
2	1	$48 \times 395 \times 105$	$1 \times 0 \times 0$	1.243	0.04	Wood
2	1	$134 \times 100 \times 100$	$1 \times 0 \times 0$	1.118	-1	Concrete/gravel
3	1	$42 \times 445 \times 105$	$1 \times 0 \times 0$	10	0.04	Wood
3	1	$24 \times 22 \times 16$	$1 \times 1 \times 1$	4	-1	Cardboard boxes
3	1	$23 \times 100 \times 45$	$1 \times 1 \times 1$	20	0.04	Cover board
4	1	$21 \times 430 \times 110$	$1 \times 0 \times 0$	330	0.04	Wood
4	1	$105 \times 120 \times 80$	$1 \times 0 \times 0$	246	-1	Concrete/gravel
4	1	$55 \times 120 \times 80$	$1 \times 0 \times 0$	435	-1	Wood
5	2	$90 \times 85 \times 120$	$1 \times 0 \times 0$	500	-1	Concrete/gravel
5	1	$50 \times 120 \times 80$	$1 \times 0 \times 0$	1.000	-1	Concrete/gravel
6	1	$56 \times 360 \times 90$	$1 \times 0 \times 0$	800	0.03	Gypsum boards
6	1	$14 \times 208 \times 12$	$1 \times 0 \times 0$	25	-1	Wood
6	1	$25 \times 16 \times 21$	$1 \times 1 \times 1$	27	-1	Cardboard boxes
6	3	$45 \times 60 \times 90$	$1 \times 1 \times 1$	0	0	Rockwool
7	1	$31 \times 360 \times 97$	$1 \times 0 \times 0$	500	0.03	Gypsum boards
7	1	$4 \times 204 \times 83$	$1 \times 0 \times 0$	25	0.01	Wood
7	1	$9 \times 361 \times 19$	$1 \times 1 \times 1$	10	0.02	Wood
7	1	$14 \times 208 \times 12$	$1 \times 0 \times 0$	25	-1	Wood
7	1	$25 \times 16 \times 21$	$1 \times 1 \times 1$	27	-1	Cardboard boxes
7	2	$45 \times 60 \times 90$	$1 \times 1 \times 1$	0	0	Rockwool
						** The capacity of the truck*

Table C.15: Generated 7

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	1	$114 \times 112 \times 244$	$1 \times 0 \times 0$	1.652	0.04	Wood
2	4	$90 \times 85 \times 120$	$1 \times 0 \times 0$	501	-1	Concrete/gravel
2	1	$55 \times 80 \times 120$	$1 \times 0 \times 0$	1.000	-1	Concrete/gravel
2	1	$35 \times 105 \times 540$	$1 \times 0 \times 0$	500	0.03	Wood
2	1	$40 \times 92 \times 218$	$1 \times 0 \times 0$	535	0.03	Wood
2	1	$50 \times 30 \times 43$	$1 \times 1 \times 1$	1	-1	Cardboard boxes
3	1	$48 \times 395 \times 105$	$1 \times 0 \times 0$	1.243	0.04	Wood
4	1	$134 \times 100 \times 100$	$1 \times 0 \times 0$	1.118	-1	Concrete/gravel
5	1	$42 \times 445 \times 105$	$1 \times 0 \times 0$	10	0.04	Wood
5	1	$24 \times 22 \times 16$	$1 \times 1 \times 1$	4	-1	Cardboard boxes
5	1	$23 \times 100 \times 45$	$1 \times 1 \times 1$	20	0.04	Cover board
6	1	$21 \times 430 \times 110$	$1 \times 0 \times 0$	330	0.04	Wood
6	1	$105 \times 120 \times 80$	$1 \times 0 \times 0$	246	-1	Concrete/gravel
6	1	$55 \times 120 \times 80$	$1 \times 0 \times 0$	435	-1	Wood
7	2	$90 \times 85 \times 120$	$1 \times 0 \times 0$	500	-1	Concrete/gravel
7	1	$50 \times 120 \times 80$	$1 \times 0 \times 0$	1.000	-1	Concrete/gravel
						** The capacity of the truck*

Table C.16: Generated 8

Customer	#	Dimensions cm	rotations	Weight kg	LBS kg/cm^2	Describtion
-	-	$720 \times 250 \times 280$	-	9.500 *	-	The truck
1	1	$114 \times 112 \times 244$	$1 \times 0 \times 0$	1.652	0.04	Wood
2	4	$90 \times 85 \times 120$	$1 \times 0 \times 0$	501	-1	Concrete/gravel
2	1	$55 \times 80 \times 120$	$1 \times 0 \times 0$	1.000	-1	Concrete/gravel
2	1	$35 \times 105 \times 540$	$1 \times 0 \times 0$	500	0.03	Wood
2	1	$40 \times 92 \times 218$	$1 \times 0 \times 0$	535	0.03	Wood
2	1	$50 \times 30 \times 43$	$1 \times 1 \times 1$	1	-1	Cardboard boxes
3	2	$95 \times 120 \times 60$	$1 \times 1 \times 1$	5	-1	Rockwool
3	1	$90 \times 30 \times 30$	$1 \times 1 \times 1$	5	-1	Rockwool
3	1	$51 \times 300 \times 90$	$1 \times 0 \times 0$	900	0.04	Gypsum boards
3	1	$15 \times 360 \times 90$	$1 \times 0 \times 0$	300	0.02	Gypsum bars
3	1	$24 \times 16 \times 23$	$1 \times 1 \times 1$	4	-1	Cardboard boxes
3	1	$13 \times 23 \times 13$	$1 \times 1 \times 1$	10	-1	Cardboard boxes
4	1	$31 \times 90 \times 240$	$1 \times 0 \times 0$	486	0.04	Gypsum boards

** The capacity of the truck*

Table C.17: Generated 9

Appendix D

Additional graphs and tables

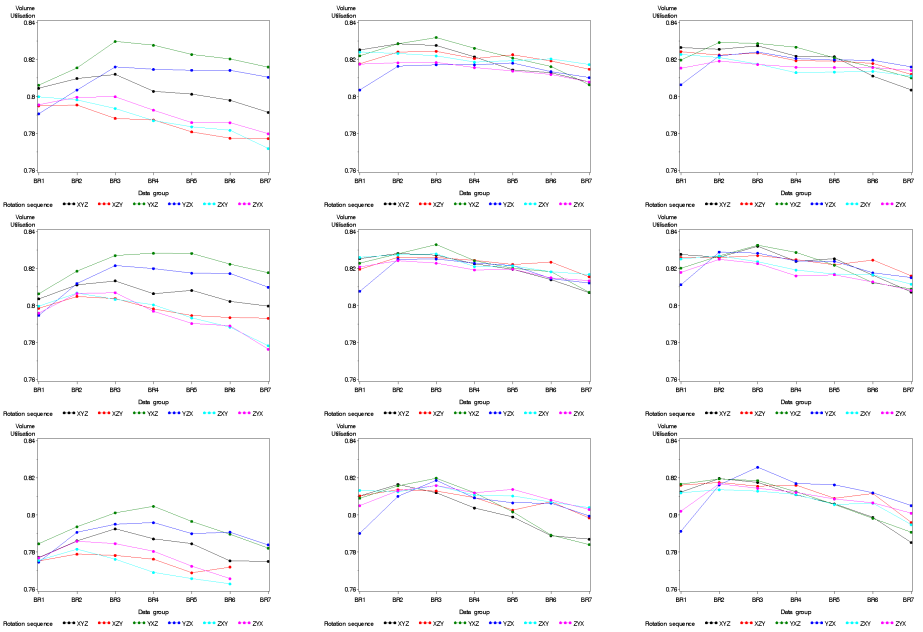


Figure D.1: Nine graphs showing results for the different sort methods in "column". The first column is sorting after biggest dimension, the middle column is after the biggest surface and the third column is after the biggest volume. The different greedy methods are shown on "rows". The greedy methods are; SB in the first row, BS in the second row and EL in the third row. On each graph the different data groups are shown on the horizontal axis. The different lines corresponds to the distinct box rotation strategies.

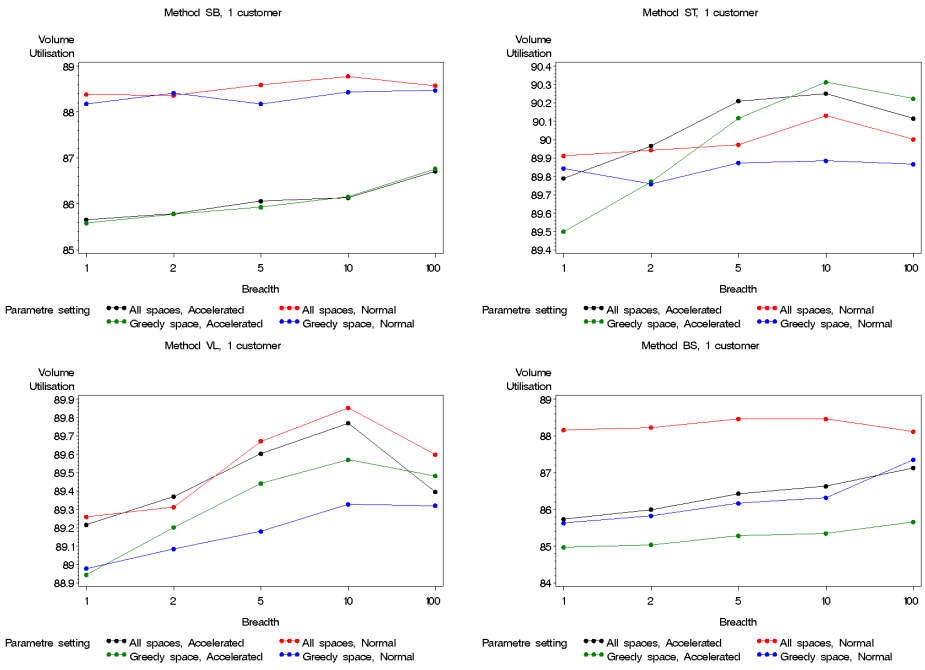


Figure D.2: Performance when one customers is present in the problems. Time limit: 60 seconds.

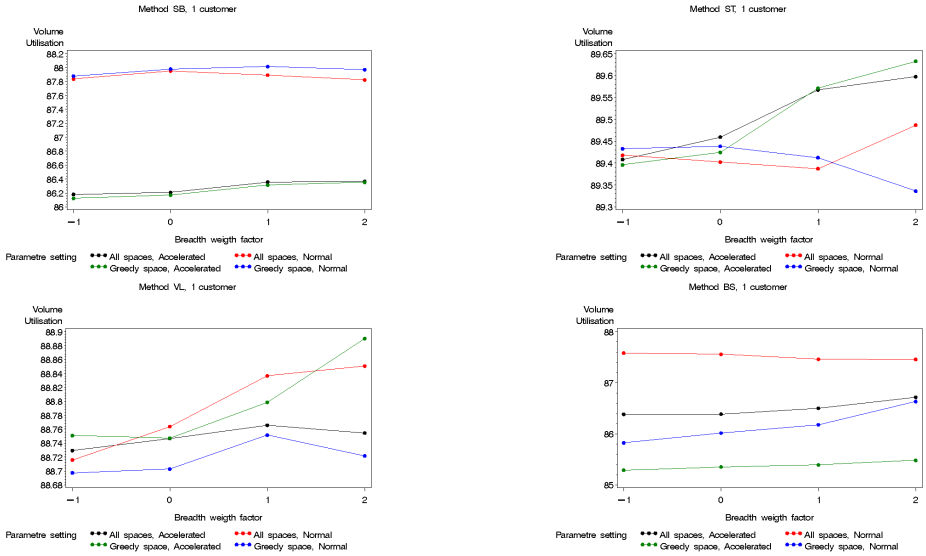


Figure D.3: Performance when one customer exists in the problems. Dynamic breadth. Time limit: 10 seconds.

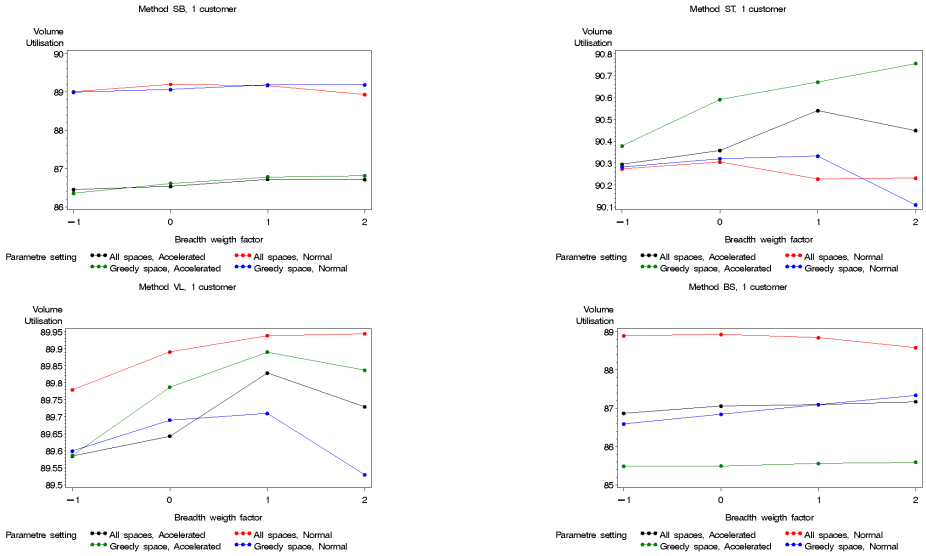


Figure D.4: Performance when one customer exists in the problems. Dynamic breadth. Time limit: 60 seconds.

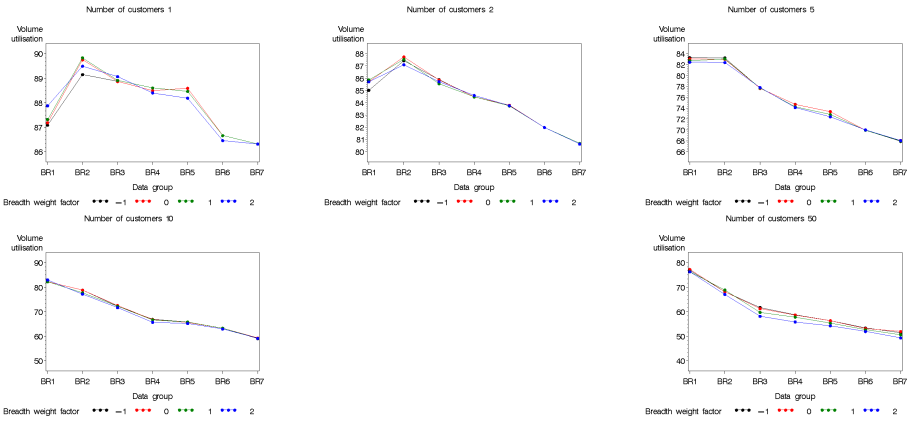


Figure D.5: Method SB. 1-50 customers. 1-50 customers. Time limit: 10 seconds.

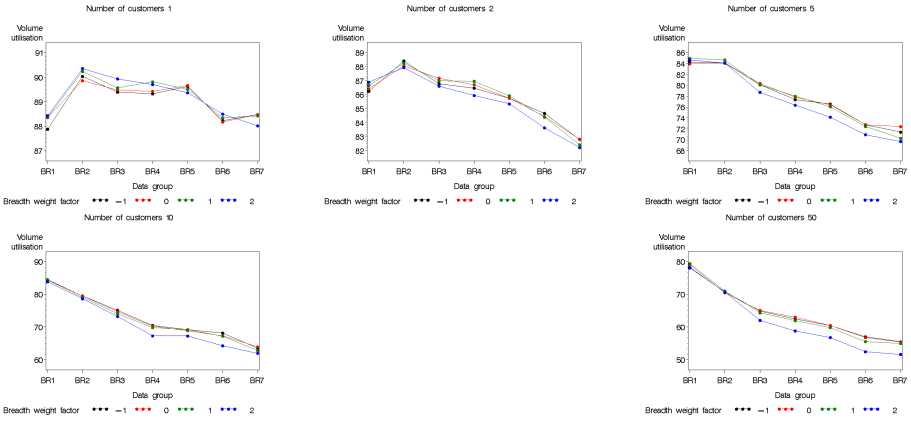


Figure D.6: Method SB. Time limit: 60 seconds.

Bibliography

[1] J.E. Beasley. OR-Library. *http://people.brunel.ac.uk/∼mastjjb/jeb/info.html*, October 2006.

[2] E.E. Bischoff. Three-dimensional packing of items with limited load bearing strength. *European Journal of Operations Research*, 168:952–966, 2004.

[3] E.E. Bischoff and M.S.W. Ratcliff. Issues in the delevopment of container loading problem. *Omega*, 23:277–390, 1995.

[4] A. Bortfeldt and H. Gehring. A hybrid genetic algorithm for the container loading problem. *European Journal of Operations Research*, 131:143–161, 2001.

[5] A. Bortfeldt, H. Gehring, and D. Mack. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29:641–662, 2003.

[6] C.S. Chen, S.M. Lee, and Q.S. Shen. An analytical model for the container loading problem. *European Journal of Operations Research*, 80:68–76, 1993.

[7] A.P. Davies and E.E. Bischoff. Weight distribution considerations in container loading. *European Journal of Operations Research*, 114:509–527, 1999.

[8] K. Doerner, G. Fuellerer, M. Gronalt, R. Hartl, and M. Iori. Metaheuristics for the vehicle routing problem with loading constraints. *Networks*, to appear, 2006.

[9] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operations Research*, 44:145–159, 1990.

[10] M. Eley. Solving container loading problems by block arrangement. *European Journal of Operations Research*, 141:393–409, 2002.

[11] ESICUP. EURO special interest group on cutting and packing. *http://paginas.fe.up.pt/∼esicup/*, February 2007.

[12] T. Feo and M.G.C. Resende. Greedy randomized adaptive search procedure. *Journal of Global Optimization*, 6:109–133, 1995.

[13] H. Gehring and A. Bortfeldt. A genetic algorithm for solving the container loading problem. *International Transactions in Operations Research*, 4:401–418, 1997.

[14] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40:342–350, 2005.

[15] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, to appear, 2006.

[16] J.A. George and D.F. Robinson. A heuristic for packing boxes into a container. *Computers and Operations Research*, 7:147–156, 1980.

[17] P.C. Gilmore and R.K. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–121, 1965.

[18] J.C. Herz. Recursive computational procedure for twodimensional stock cutting. *IBM Journal of Research and Development*, 16:462–469, 1972.

[19] M. Iori, J.J. Salazar Gonzalez, and D. Vigo. An exact approach for the vehicle routing problem with two dimensional loading constraints. *Transportation Science*, to appear, 2006.

[20] A. Lim and X. Zhang. The container loading problem. *2005 ACM Symposium on Applied Computing*.

[21] D. Mack, A. Bortfeldt, and H. Gehring. A parallel hybrid local search algorihtm for the container loading problem. *International Transactions in Operations Research*, 11:511–533, 2004.

[22] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48:256–267, 2000.

[23] R. Morabito and M. Arenales. An AND/OR-graph approach to the container loading problem. *International Transactions in Operations Research*, 1:59–73, 1994.

[24] A. Moura and J.O. Oliviera. A GRASP approach to the container-loading problem. *Transportation and Logistics*, pages 50–57, 2005.

[25] B.K.A. Ngoi, M.L. Tay, and E.S. Chua. Applying spatial representation techniques to the container packing problem. *International Journal of Production Research*, 32:111–123, 1994.

[26] F. Parreño, R. Alvarez-Valdes, J.F. Oliveira, and J.M Tamarit. A maximal-space GRASP algorithm for the container loading problem. *Proc. 4th ES-ICUP Meeting*.

[27] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.

[28] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operations Research*, 141:382–392, 2002.

[29] D. Pisinger and M.M. Sigurd. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2:154–167, 2005.

[30] G. Scheithauer. Algorithms for the container loading problem. *Operations Research Proceedings*, pages 445–452, 1992.

[31] G. Scheithauer. LP-based bounds for the container and multi-container loading problem. *International Transactions in Operations Research*, 6:199–213, 1998.

[32] G. Scheithauer and J. Terno. The G4-heuristic for the pallet loading problem. *Journal of the Operational Research Society*, 47:511–522, 1996.

[33] D. Vigo. VRPLIB: A vehicle routing problem library. *http://www.or.deis.unibo.it/research.html*, February 2007.

[34] L.A. Wolsey. *Integer Programming*. Wiley-InterScience publication, 1998.

[35] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operations Research*, to appear, 2006.