ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Master Thesis
Operations Research and Quantitative Logistics

Linear model for estimating the number of boxes in a container: GRASP-based heuristic for synthetic data generation

Name student:
Olivier Knappert
Student ID number:
449890

Supervisor:
dr. T.A.B. Dollevoet
Second assessor:
Y.N. Hoogendoorn

Abstract

To compute the expected number of filled wheeled containers at the end of a production day, bol.com needs a model which accurately predicts the average number of packages that are loaded in a container. In this paper we first present a reactive GRASP for solving a container loading problem. Based on this algorithm we derive a greedy packing heuristic, which mimics the loading process at a single loading dock in a bol.com fulfillment center. This heuristic is used to construct synthetic data for training and testing a linear regression model. Input of the linear model consists of parameter settings of influence to the loading process and the distribution of different package sizes. Based on results, we conclude that 95% of the predictions deviate less than 4.72 boxes. We demonstrate the model to be very accurate and hence useful in practice.

June 27, 2022

Contents

1 Introduction

Bol.com is the biggest e-commerce company in the Netherlands and has several fulfillment centers (FC) spread out over the Netherlands, each storing specific products based on product sizes. The *Central Bookhouse* is the FC where bol.com keeps all its literature stock. At the *Bol Fulfillment Center XL* all products such as tv's and washing machines are stored. Then, there are three other FCs, namely *Bol Fulfillment Center 1* and *2* and *Veerweg* where all products ranging from a pen to the Philips Airfryer are stored.

On most of these products, bol.com offers a next-day delivery service. In addition, bol.com offers a sameday, evening and sunday delivery. Hence, one can order a product at any time of the day and have it delivered as preferred.

The packages are collected by several carriers on a daily basis. Each carrier has its own post sorting centers where the packages are sorted, after which they are delivered to the customer. We can distinguish multiple delivery flows, characterised by the type of delivery service in combination with the post sorting center where the packages are to be sorted. As an example, Figure 1.1 graphically depicts two delivery flows.

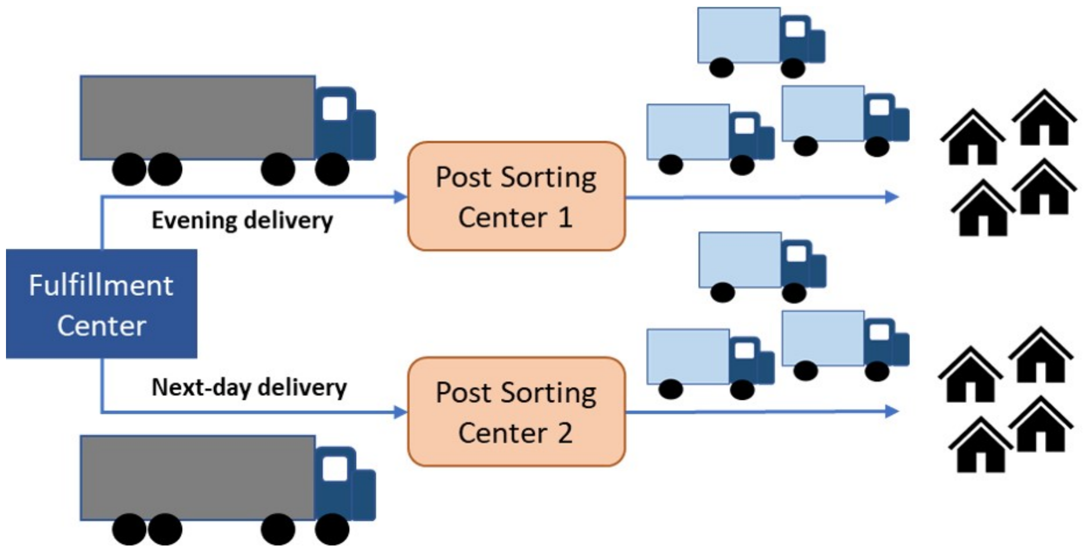


Figure 1.1: Example of delivery flows.

Each FC consists of a vast network of production lines and conveyor belts. In a FC packages are produced, i.e. products are packed into cardboard boxes on a production line. Then, the packages are transported over the conveyor belts towards predefined loading docks in the FC. Packages accumulate at a loading dock, after which the packages are manually taken off the belt and loaded into a wheeled container (see Figure 1.2). At each loading dock packages arrive belonging to the same delivery flow. In the following, we will refer to boxes instead of packages.

To ensure that all boxes are collected and transported at the end of a production day, it is important that enough trucks are scheduled for each delivery flow. Trucks are scheduled based on the expected number of filled wheeled containers. The number of containers that fit in a truck is given. The average number of boxes that fit in a wheeled container may vary. Also, at the moment of scheduling trucks, the sales are not yet realised. Hence, the number of boxes that are going to be produced is stochastic as well.

A *distribution forecast* serves as input for the scheduling of trucks. This forecast includes an expectation on the total number of boxes T that need to be transported on each delivery flow. If it is assumed that a container transports Y boxes on average, then the expected number of filled containers equals T/Y. In the current process, an intelligent method to predict the value for Y is not available, other than setting Y equal to a moving average of past production days. On a single day several hundred containers are loaded per delivery flow.

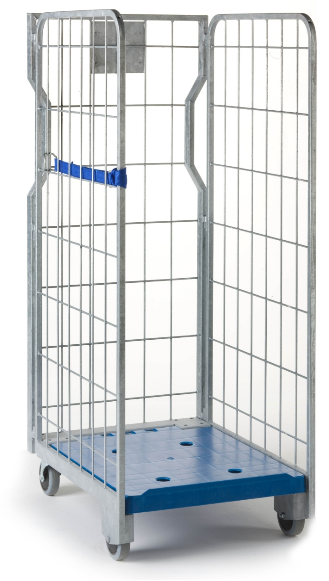


Figure 1.2: Wheeled container, 180 x 75 x 75 (cm).

The problem of loading boxes in a container, such that the used volume is maximized, is also referred to as a container loading problem (CLP). In most loading problems the input is given beforehand, i.e. the boxes that need to be loaded. At the bol.com fulfillment centers, we distinguish a different kind of loading problem. In this variant of the CLP, the input is not completely known at the start of packing the container. Boxes arrive at a loading dock according to a stochastic process and are packed in loading iterations. In each iteration, at most several boxes accumulate before they are being loaded into a container. Hence, we consider an online variant of the CLP.

The goal of this research is to develop a linear regression model, which accurately predicts for a given day and for a single delivery flow the value for Y, i.e. the average number of boxes that are loaded in a wheeled container. The model takes an input group X consisting of (a) the number of boxes that accumulate in each loading iteration at the loading dock before the workman starts packing, (b) the number of containers that are loaded in parallel and (c) the distribution of different box sizes. Each FC has a certain number of different box sizes which can be used to pack products. The distribution consists of the proportion of each box size.

In order to train such a linear model, we construct synthetic data. We develop an online greedy packing heuristic, which mimics the loading process at one loading dock. The greedy packing heuristic receives as input a predefined order in which boxes arrive at the loading dock. The hyper parameter settings of the heuristic consist of (a) the number of boxes that accumulate in each loading iteration and (b) the number of containers that are loaded in parallel.

We are going to compute an observation Y_i for an input group X_i. Based on the distribution of box sizes of X_i, we can generate many different box orders, which are all separately processed by the packing heuristic. This way we obtain many loaded containers, based on which we can compute one observation Y_i, i.e. the average number of boxes packed per container corresponding to the input group X_i. This process is repeated for many different input groups, in order to construct an extensive synthetic data set.

The packing heuristic is derived from a maximal-space reactive GRASP proposed by Parreño, Alvarez-Valdés, Tamarit, and Oliveira (2008). We extend this algorithm as it obtains very good results in short run times. We are going to adjust the algorithm such that it can solve an online loading problem. The results are therefore going to be of less quality than those of the reactive GRASP. Hence, it is preferred to start with an algorithm with a performance as good as possible. Also, due to the greedy element of the reactive GRASP, this algorithm is fit for mimicking the workman loading the container.

The structure of this paper is as follows. In Section 2 we give a clear description of the CLP at hand, the loading process at a loading dock in a FC and the parameters that are of influence to the loading process. We introduce a well-established benchmark set for container loading problems and we introduce the bol.com problem class. Then, in Section 3 we review related literature and state the most important take-aways and performance results of every paper. In Section 4 we first present the maximal-space reactive GRASP of Parreño et al. (2008). We will do computational experiments on the reactive GRASP to demonstrate the loss of quality by solving an online instead of an offline loading problem. Then, we derive the online greedy packing heuristic followed by the synthetic data generation process. In Section 5 we present extensive computational experiments on the reactive GRASP, on the greedy packing heuristic and finally on the performance of the linear regression model. We end with concluding remarks and suggestions for future research in Section 6.

2 Problem Description

As described in Section 1, we develop a heuristic which solves a container loading problem, in order to construct synthetic data for the linear regression model. We first present the official problem formulation of the CLP which is considered in this research in Section 2.1. In Section 2.2 we describe the online variant of the loading problem considered at the bol.com fulfillment centers. Then, we further elaborate on the box sizes and container dimensions at the fulfillment centers. We compare these with a well-established benchmark in Section 2.3.

2.1 The three dimensional container loading problem (3D-CLP)

The problem of orthogonally loading a set of rectangular-shaped three-dimensional small items (boxes), into a single rectangular-shaped large object (container), such that all boxes lie within the container, no boxes overlap and the used volume is maximized, is also referred to as the *Three Dimensional Single Container Loading Problem* (3D-CLP). A solution to the 3D-CLP is called a *loading pattern*. Bortfeldt and Wäscher (2012) decompose this problem into two problems, based on the set of boxes to be loaded. In case this set consists of weakly heterogeneous boxes, they refer to this problem as a *Single Large Object Placement Problem* (SLOPP). In case of a strongly heterogeneous set of boxes, the problem is referred to as a *Single Knapsack Problem* (SKP). The 3D-CLP is $\mathcal{NP}$-hard in the strong sense, hence it is very difficult to solve this problem to optimality for large size instances (Bortfeldt & Wäscher, 2012).

In this paper we also consider the *Three Dimensional Multi Container Loading Problem*, where multiple identical containers are loaded with boxes. This problem is also referred to as *Multiple Identical Large Object Placement Problem* (MILOPP) or *Multiple Identical Knapsack Problem* (MIKP). In the following we will address both the single and multi container problems with 3D-CLP.

There are many constraints which can be considered in a loading problem. A common constraint is an orientation constraint which may prohibit up to five sides of a box to be faced upwards. An example of such a box is a moving box, which is supposed to be placed with the opening facing upwards. Another frequently used constraint is one that ensures stability of a box. This means that the i-th dimension of a box can only be vertically placed, if the size of the i-th dimension d_i is smaller than L times the size of the smallest dimension. In other words, it must hold that $\frac{d_i}{\min_{i=1,2,3} d_i} < L$.

2.2 Online variant of the 3D-CLP at a bol.com FC

As briefly mentioned in Section 1, we want to construct an algorithm which mimics the loading process at a single loading dock in a bol.com FC. At such a loading dock many containers are packed with boxes during one production day. Boxes arrive at a loading dock, where a workman is waiting to manually load the boxes in a container upon arrival. In other words, the input for each container — the boxes to be packed — is not known in advance. Input is iteratively released after which it is processed. The workman strives to maximize the volume use of a container. Hence, we consider an online variant of the 3D-CLP.

Considering the loading process at a loading dock, we can distinguish three factors that influence the value for Y, the average number of boxes that are loaded in a container.

(a) ***#boxes.*** Firstly, the number of boxes that accumulate before the boxes are loaded in a container. In general, one workman is responsible for the loading at multiple loading docks. Hence, not every box is instantly loaded into a container upon arrival at the loading dock. The quality of a loading pattern improves when the size of the input in each iteration increases. This follows from the fact that results of an offline optimization problem are at least as good as those of an online optimization problem.

(b) ***#containers.*** Another factor might be the number of containers that are loaded in parallel at a single loading dock. If in each loading iteration, one can optimize packing boxes over multiple containers, we might significantly improve the loading patterns.

(c) ***Distribution of different box sizes.*** The last factor is the distribution of different box sizes. In a FC it is not centrally managed which packages are produced at what time. Hence, there is no insight in the order in which boxes will arrive at the loading dock. We could however randomly generate the order in which boxes arrive, based on the distribution of different box sizes.

The distribution of box sizes remains undetermined. It would require extensive data analysis to draw up these distributions. We do have other sources of information. Firstly, *line production forecasts* give estimates of the number of packages that are going to be produced per production line per hour. There are different lines and on each line, packages of certain sizes are produced. Another source of information could be the historic proportion of each box size in the total number of packages produced on a day. To compute these proportions, we could analyse data regarding the products that were packed each day. It is known which and how many of each product were packed. For each product a packing advice informs us in which box size the item should have been packed. For this research, we will assume that the distribution of different box sizes is given.

2.3 Bol.com problem class compared with an established benchmark set

Bol.com operates several fulfillment centers, each with its own characteristics and specific product size groups. For this research, we focus on the *Veerweg*. In this FC all products ranging from a pen to a Philips Airfryer are kept in stock. In order to ship all these products, there is a total of 14 different box sizes which can be used to pack a product. To get a better understanding of the box sizes, Table 2.1 can be consulted.

Table 2.1: Box sizes and corresponding dimensions of bol.com problem class.

	Dimensions (cm)			
Box size	Height	Width	Depth	Volume (cm^3)
Soap box	20	14	6	1680
Match box	20	12	8	1920
Parfumebox	24	16	8	3072
Sound box	28	20	8	4480
Lunch box	24	20	12	5760
Sandbox	31	23	14	9982
Outbox	28	24	20	13440
Colorbox	40	32	12	15360
PA-8 Juke box	36	28	22	22176
Pillowbox	54	40	12	25920
Shoebox	48	38	20	36480
Pandora's box	45	35	30	47250
Toolbox	64	48	22	67584
PA-14 Sky Box	70	50	35	122500

These 14 different boxes are packed in a wheeled container with dimensions equal to 180 cm high, 75 cm wide and 75 cm deep, earlier depicted in Figure 1.2. In most literature on the 3D-CLP, the researchers consider twenty-foot equivalent unit (TEU) containers generally used in sea freight, where, in the literature, the inner dimensions of such a container are assumed to be 220 cm high, 233 cm wide and 587 cm deep.

One evident difference between the containers is the capacity, where the TEU container has a capacity more than 29 times the volume of a wheeled container. Another important contrast is the relative size of the dimensions. In case of the wheeled container the largest dimension is the height, whereas in case of the TEU container the largest dimension is the depth. For both containers the remaining two dimensions are (almost) the same size. One could consider a wheeled container as TEU container tilted upwards, where the size of each dimension of the TEU container is approximately 3 times the size of respective dimensions of the wheeled container.

In many papers a benchmark set is used for comparison of performance, consisting of 1500 problem instances from Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) (BRD problem classes). This benchmark set is well known and commonly used in the literature. The complete set ranges from weakly to strongly heterogeneous problem classes, further referred to as BRD_{01} to BRD_{15}, where each class consists of 100 instances. The number of different box sizes in each problem class is 3, 5, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, where the number of boxes per instance can range from circa 100 to 150. For these instances a TEU container is used to pack the boxes. In each problem instance it is also stated for every box size and its dimensions whether a vertical placement is allowed, based on the stability threshold explained before in Section 2.1. We will disregard this stability constraint, unless mentioned otherwise.

We have already described the (dis)similarities between a wheeled and a TEU container. It is also interesting to investigate whether the dimensions of the boxes are of similar proportions. To put the boxes of the bol.com problem class into perspective with those of the BRD problem classes, we are interested in the relative dimensions of the boxes with respect to the dimensions of the container they are being loaded into. Table 2.2 presents several measures regarding the dimensions of boxes of both the bol.com and the BRD_{06} problem class, proportionate to the dimensions of the container. We focus on BRD_{06}, since this class consists of 15 different box types, where the bol.com class has 14 different box types.

Table 2.2: Proportion of each dimension relative to the container for the bol.com class and the average of the BRD_{06} instances.

Problem class	bol.com			BRD_{06}		
Container	Wheeled container			Tilted TEU container		
Measure	Height	Width	Depth	Height	Width	Depth
MIN	0.11	0.16	0.08	0.09	0.15	0.11
MAX	0.39	0.67	0.47	0.20	0.38	0.30
AVG	0.21	0.38	0.22	0.15	0.26	0.19

We will give two clarifying examples to aide with interpreting Table 2.2. We can conclude that the heights of the boxes with the smallest and highest height, are respectively 11% and 39% of the wheeled container's height. The BRD_{06} problem class has 100 instances. For each instance there is a minimum and a maximum proportion with respect to the dimension size. From the table, we can deduce that on average over all 100 instances, the height of the box with the smallest (largest) height, is 9% (20%) of the height of a tilted TEU container. We see that the minimum sizes of all three dimensions are quite similar for both containers. However, the maximum sizes of all three dimensions are significantly smaller for the tilted TEU container.

Observing the average proportions of the dimensions with respect to the container's dimensions, we note that on average the boxes are relatively larger for the bol.com problem class. We expect that an algorithm constructs loading patterns with a higher volume use for BRD_{06} than for the bol.com problem class. This follows from the reasoning that it is easier to pack 100 small boxes instead of 10 bigger boxes with the same total volume.

3 Literature Review

As described in Section 2 this research is concerned with the 3D-CLP. Bortfeldt and Wäscher (2012) concluded that the 3D-CLP is strongly $\mathcal{NP}$-hard. Only few papers present an exact approach for solving the problem at hand, whereas most papers cover heuristic algorithms. To get a good understanding of the different types of container loading problems and their exact and heuristic algorithms, Bortfeldt and Wäscher (2012) can be consulted. The authors give an extensive review of the state-of-the-art in the field of container loading problems and state all terminology, relevant factors and constraints to be considered when dealing with loading problems. All research published until 2012 is summarized according to the type of loading problem covered and the constraints considered. They globally review exact and heuristic algorithms and propose future research opportunities. In the following Sections 3.1 and 3.2, we focus on relevant literature on exact and heuristic algorithms respectively.

3.1 Exact algorithms

For a more recent and highly extensive overview on most relevant exact algorithms for 3D loading problems, Silva, Toffolo, and Wauters (2019) should be consulted. The authors present a comparative study based on over 15000 CPU hours of computations using classic benchmark data sets and newly-generated instances. Main take-aways of the paper are that the 3D-CLP remains very difficult to solve to optimality, even for medium-sized instances. Run times increase when dimensions of a container are smaller and increasing the number of box sizes generally results in a higher run time. Linear relaxations are not effective. In order to improve the field of exact solution methods, research should focus on the search for effective combinations of relative positions of boxes. This could potentially speed up the algorithms.

Two examples of papers that studied an exact solution method are Junqueira, Morabito, and Yamashita (2012) and Martello, Pisinger, and Vigo (2000). Junqueira et al. (2012) present a mathematical formulation for solving the 3D-CLP with stability and load bearing constraints. The paper showed good performance for moderate problem instances of up to 5 different box sizes. Martello et al. (2000) developed an exact branch-and-bound algorithm for the 3D-CLP, which was able to solve instances of up to 90 boxes to optimality within reasonable time.

3.2 Heuristic algorithms

In the following, we will cover relevant literature with respect to heuristic algorithms for solving the 3D-CLP, structured in three parts. We consider Genetic and Variable Neighborhood Search (VNS) algorithms, three GRASP algorithms and a tree search algorithm.

3.2.1 Genetic and Variable Neighborhood Search algorithms

Gonçalves and Resende (2012) present a multi-population biased random-key genetic algorithm (BRKGA) for the 3D-CLP. The BRKGA considers two constraints, namely an orientation and support constraint. The support constraint states that all bottom sides of a box not placed on the floor, need to be fully supported by the top sides of one or more boxes beneath it. In this paper an extensive comparison is given with respect to the performance of 13 approaches — according to Gonçalves and Resende (2012) the most effective to date at the time of writing — for solving the 3D-CLP where a stability constraint is and is not enforced. For the comparison the BRD instances, which were introduced in Section 2.3, are used. The algorithm proposed by Gonçalves and Resende (2012) finds the best solutions for each instance. The average volume use for the BRD instances is 94.54% and the average run time is 147 seconds when the stability constraint is not enforced. The comparison with respect to the run time is left out, as algorithms were implemented and tested on computers with different processing power.

Gehring and Bortfeldt (2002) present a parallel genetic algorithm for the 3D-CLP. In this paper the goal was to examine whether the solution quality of a basic genetic algorithm — previously published by Gehring and Bortfeldt (1997) — could be improved . The authors conclude that the parallel implementation of the genetic algorithm dominates over a basic genetic algorithm of Gehring and Bortfeldt (1997) and a hybrid genetic algorithm of Bortfeldt and Gehring (2001). This comparison is based on the average volume utilization over BRD instances. It must be noted that Bortfeldt and Gehring (2001) state that the average volume utilization of Gehring and Bortfeldt (1997) and Bortfeldt and Gehring (2001) is 86.4% and 88.6%, respectively. The average run time for the latter is 316 seconds, relatively high, compared to 11.7 seconds for the basic genetic algorithm.

Parreño, Alvarez-Valdés, Oliveira, and Tamarit (2010) propose and compare several VNS algorithms. A proposed VND uses five neighborhoods of which three are simple and fast neighborhoods and two are more complex neighborhoods. The order in which the VND algorithm considers the neighborhoods influences the quality of the solution. It is shown that a mixed order produces much better results than taking them in order of complexity, though the computing times are longer for strongly heterogeneous classes. The average volume utilization over BRD_{01} to BRD_{07} is 94.5% and the average run time is only 28 seconds. The proposed algorithm does not enforce the stability constraint. However, as the volume usage is very high, stability of boxes is not an issue.

3.2.2 GRASP algorithms

Moura and Oliveira (2005) present a greedy randomized adaptive search procedure (GRASP) for solving the 3D-CLP. This approach is based on a modified version (GRMod) of a greedy heuristic from George and Robinson (1980) for solving the 3D strip-packing problem (3D-SPP). The heuristic is modified such that the resulting loading pattern has a length equal to or less than the container's length. The GRMod is focused on a wall-building procedure and builds on the concept of empty spaces. Wall-building is an approach where an arrangement of boxes of the same type is structured in rows and columns. In the container rectangular parallelepiped spaces exist where no boxes are packed, also referred to as empty spaces. If the sizes of the width and height dimensions of such an empty space are equal to the container's dimensions, this space is used to pack a new wall. GRMod ensures that each box is fully supported from underneath. The heuristic could be adjusted such that this constraint is not enforced. In GRMod empty spaces are amalgamated. Therefore the approach rather focuses on box interchanges than on wall interchanges. The proposed heuristic consists of two steps. First, an initial solution is constructed by the GRMod. Then, the solution is improved by means of a local-search algorithm. The average volume use for the BRD instances is 86.75% and the average run time is 68 seconds.

In another paper written by Parreño et al. (2008), the authors propose another GRASP for the 3D-CLP. The initial solution is constructed by a block heuristic. The block heuristic constructs walls of boxes which can be placed horizontally and vertically, therefore it is also called a layer-building heuristic. The proposed block heuristic differs from other similar heuristics as it represents the empty spaces as a set of non-disjoint empty spaces, i.e. the maximal-spaces. This results in more flexibility and higher quality of the solution. The average volume utilization over BRD_{01} to BRD_{07} is 92.9% and the average run time is only 8 seconds.

In a very recent paper by Gajda, Trivella, Mansini, and Pisinger (2022), the authors present a GRASP for a real-life 3D-CLP faced by a logistics company that loads and unloads hundreds of trucks per day. This is the first paper to address all the following constraints jointly, namely constraints to ensure safety and facilitate cargo handling, including customer priorities, load balancing, cargo stability, stacking constraints, positioning constraints, and limiting the number of unnecessary cargo move operations during multi-shipment deliveries. The 3D-CLP is solved based on multiple objectives, as they consider a trade-off between volume utilization and unnecessary move operations. The algorithm obtains very good results in only a few seconds in most cases. With respect to solving the BRD instances, the algorithm does not perform as well as other solution approaches. This can be justified by the fact that for these instances only the stability constraint is enforced. Hence, a simplified formulation is used whereas the proposed approach of Gajda et al. (2022) is developed for more complex problems.

3.2.3 Tree search algorithm

Fanslau and Bortfeldt (2010) present a tree search algorithm for solving the 3D-CLP. The algorithm employs a traditional block building approach (small gaps are allowed), after which a tree search is used to pack the blocks. This algorithm respects in addition to a support and rotation constraint, a guillotine cutting constraint. This suggests that a loading pattern can be reproduced by a series of guillotine cuts. Interesting to note is that the paper presents results for two different parameter settings. The differences between the sets are the imposed time limits. The set with higher time limits (set A) is solved with a 'fast' computer and the set with lower time limits (set B) is solved with a 'slow' computer. Set A serves for finding better solutions, while set B serves for comparison with older methods. The average volume use using set A for the BRD instances is 91.9% and the average run time is 320 seconds. For set B these values are 91.2% and 54 seconds respectively.

4 Methodology

In this section we first present a reactive GRASP of Parreño et al. (2008) for solving the 3D-CLP in Section 4.1. This algorithm forms the basis for the greedy packing heuristic for solving an online variant of the 3D-CLP, proposed in Section 4.2. Then, in Section 4.3 we explain the process for constructing the BRD benchmark set. Finally, in Section 4.4 we elaborate on the process for generating the synthetic data and on the linear regression model.

4.1 A reactive GRASP for solving the 3D-CLP

For this research, we need to develop an algorithm which can solve an online variant of the 3D-CLP and it needs to mimic the loading process at a loading dock in a bol.com FC. We decided that the reactive GRASP proposed by Parreño et al. (2008) would form a good basis for the algorithm.

The greedy constructive heuristic and the improvement phase of the reactive GRASP can easily be adjusted, such that they resemble the loading logic of a workman. Also, we can modify the reactive GRASP, such that there are multiple iterations of a constructive step followed by an improvement step, in order to resemble the loading iterations for one container at a loading dock. We want to obtain containers with a high volume use. Hence, we want to start off with an algorithm that obtains very good results, since the results of the online variant are always going to be of less quality.

In this section we extensively describe the reactive GRASP, in order to derive the greedy packing heuristic later. A GRASP consists of a constructive phase which builds an initial solution, followed by an improvement phase. In order to obtain different starting solutions, a stochastic component is incorporated in the constructive phase. We first present the constructive heuristic used in the constructive phase in Section 4.1.1. Then, the constructive and improvement phase of the GRASP are more thoroughly explained in Section 4.1.2.

We are going to do computational experiments to examine whether we can adjust some parameter settings in order to improve the performance of the reactive GRASP. Also, we will use the reactive GRASP to demonstrate the difference in solution quality, as a result of solving an online variant of the CLP instead of an offline CLP.

4.1.1 A constructive heuristic

The constructive heuristic at hand is a block heuristic. A block heuristic assembles boxes of the same type in a column or a layer and places these in an *empty maximal space*. Note that we use the term column, even when a column is put horizontally. A layer is composed of multiple columns. This heuristic differs from other similar block heuristics, in the sense that the empty spaces are non-disjoint. This suggests that some empty spaces might overlap. The spaces are called maximal, as they are the largest empty rectangular parallellepiped that can be used to pack blocks of boxes. The heuristic iteratively selects the next empty space which is to be filled with a block. Then, a block is assembled and loaded into the empty space. Finally, the set of empty maximal spaces is updated. This process continues until no boxes remain to be packed, or there are no empty maximal spaces left, which can fit at least one box.

Several variables for lists and sets need introducing. First, let $\mathcal{B}$ be the set of box sizes i that still need to be packed, which is initially the complete set of box sizes $i = 1, ..., m$. Then, we consider the set $\mathcal{S}$, which consists of all the empty maximal spaces, which is initially only the container C. Let q_i and p_i be the number of boxes of box size i that still need to be packed and the number of boxes which have already been packed respectively. In the following we extensively describe the three steps of the constructive heuristic.

- *Step 0: Initialization*

  $\mathcal{S} = \{C\}$ — set of empty maximal spaces.

  $\mathcal{B} = \{1, 2, ..., m\}$ — set of box sizes still to be packed.

  $q_i = n_i$ — number of boxes of size i to be packed, for all $i = 1, ..., m$.

  $p_i = 0$ — number of boxes of size i packed, for all $i = 1, ..., m$.

- *Step 1: Choosing the maximal space in $\mathcal{S}$.* The constructive heuristic follows an order in which first the corners are filled, followed by the sides and finally the inner space. To order the empty spaces, a measure of distance is introduced. Consider two points $a = (x_1, y_1, z_1)$ and $b = (x_2, y_2, z_2)$ in $\mathbb{R}^3$, where we define a distance vector $d(a, b)$ with the components $|x_1 - x_2|$, $|y_1 - y_2|$ and $|z_1 - z_2|$ ordered in non-decreasing order. Let $a = (5, 3, 1)$ and $b = (0, 3, 4)$, then $d(a, b) = (0, 3, 5)$, as a result from ordering the differences 5, 0, 3 in non-decreasing order. For every maximal space S, we compute the distance vector from every corner of the space to the corner of the container closest to it and keep the minimum:

$$d(S) = \min\{d(a, c) : a \text{ corner of } S, c \text{ corner of container } C\} \tag{1}$$

At each iteration, the maximal space is chosen with the minimum distance to a corner of the container and where at least one of the remaining box types fits in the maximal space. In case two spaces have an equal distance, the space is chosen with the bigger volume. Let S^* denote the chosen maximal space. The layer that is packed in S^*, is placed in corner a of S^* for which $d(S^*)$ was minimal. A space is represented by its corners with minimum and maximum coordinates. As an example, the container is represented by $\{(0, 0, 0), (180, 75, 75)\}$, where the elements represent the height, width and depth respectively.

• *Step 2: Choosing the boxes to pack.* Having chosen an empty maximal space, we need to construct a block. Parreño et al. (2008) did extensive research and concluded that the constructive heuristic performs best, when the blocks constructed are layers instead of only columns. For each box size i, we consider all possible layers that fit into S^* and for which the number of boxes does not exceed q_i. Each box can be rotated in six directions. Figure 4.1 gives an example of the possible directions in which a box can be rotated.

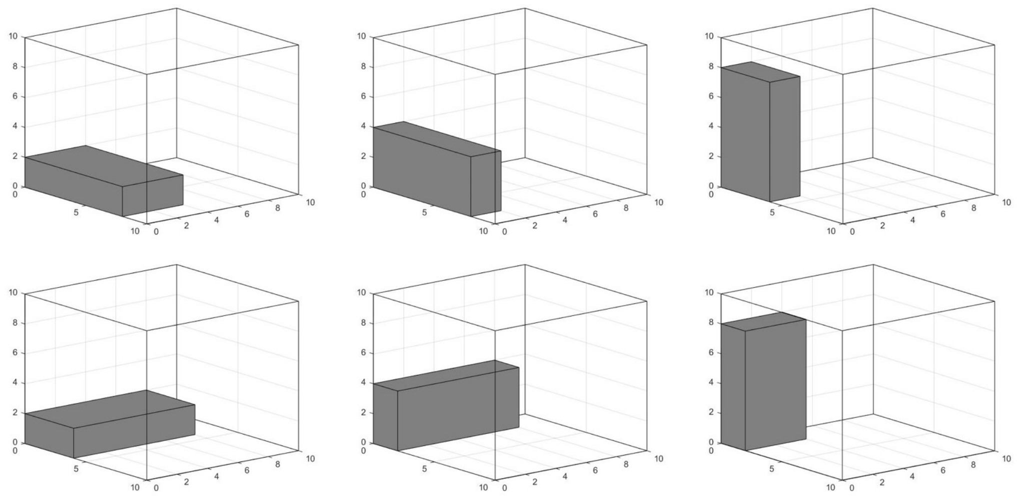
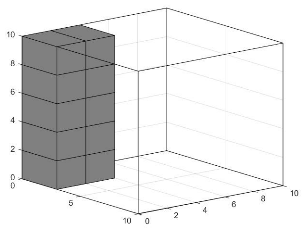
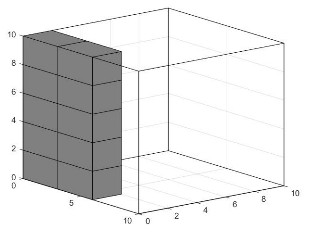


Figure 4.1: Six possible box rotations.

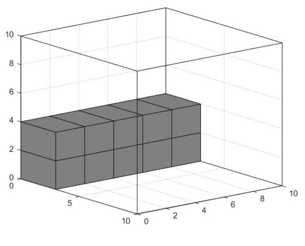
For each box rotation, there are six alternatives for assembling a layer. Figure 4.2 shows the alternatives for constructing layers for a certain box rotation where $q_i = 12$. The layer in Figure 4.2a is assembled by first constructing a column in the height dimension, followed by adding copies of this column in the width dimension, such that the total number of boxes does not exceed q_i. In short, for one box type, at most 36 layers are considered to be packed in S^*.



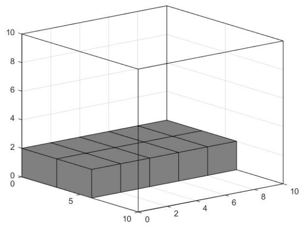
(a) Axis XY



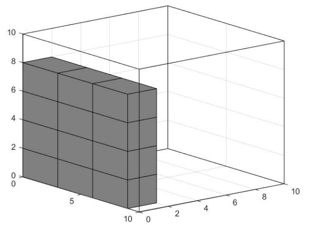
(b) Axis XZ



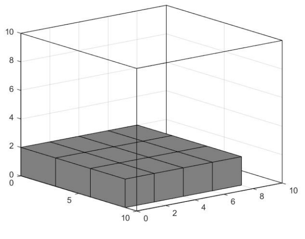
(c) Axis YX



(d) Axis YZ



(e) Axis ZX



(f) Axis ZY

Figure 4.2: Six alternatives for constructing a layer.

Having constructed all feasible layers, we need to order them based on a certain criterion and the best layer is chosen accordingly. We consider two criteria:

(i) **Volume**: Choose the layer that produces the largest increase in volume occupied.

(ii) **Best-fit**: Choose the layer of boxes that fit best in S^*. For each block we can construct a distance vector with components being the distance to each side of the maximal space. These components are ordered in non-decreasing order and using a lexicographical order, we choose a layer. In Figure 4.3 four examples are given to illustrate the best-fit criterion. In Figure 4.3a the layer completely fills the (10,10,10) empty space. In Figure 4.3b the layer fills the height and width dimension. The distance to the depth dimension is 8. In Figure 4.3c only the height completely matches the corresponding space dimension and in Figure 4.3d none of the dimensions matches the space dimensions.

For both criteria it holds that in case of a tie, the layer with the minimum number of boxes is selected. When a layer is selected with r_i boxes, we set $p_i = p_i + r_i$ and $q_i = q_i - r_i$. If $q_i = 0$, we remove i from set $\mathcal{B}$.

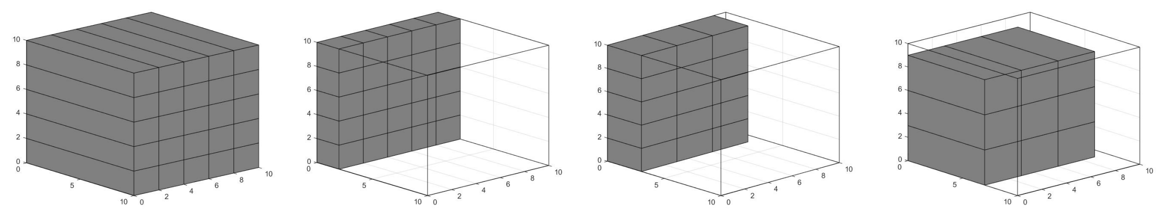

(a) (0,0,0) **(b)** (0,0,8) **(c)** (0,1,7) **(d)** (1,1,3)

Figure 4.3: Examples of the best-fit criterion distance measure.

• *Step 3: Updating the set $\mathcal{S}$.* Unless the layer fits completely in S^*, new empty maximal spaces are created which replace S^*. Note that at most three new spaces are created to replace S^*, namely a height, width and depth space. Which spaces are created, depends on the corner in which the layer is placed. In Figure 4.4 we see that the layer is placed in one of the lower corners, thereby creating a new height space above the layer. If the layer were placed in one of the upper corners, the height space would be under the layer. The same reasoning holds for the width and depth spaces. Remember that a space is represented by its minimum and maximum coordinate.

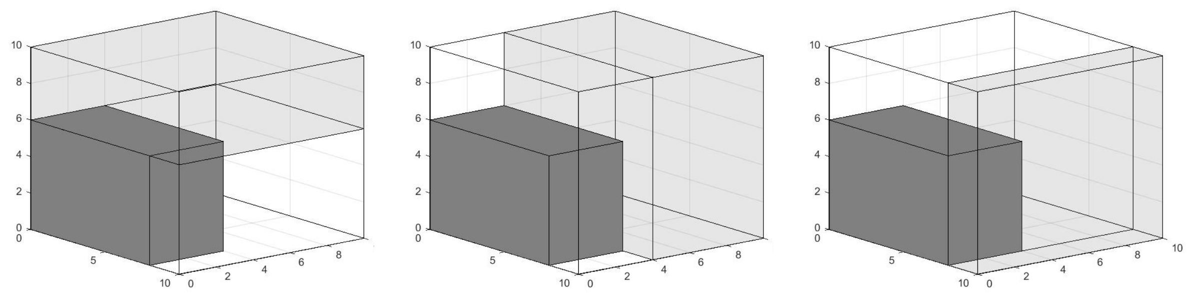
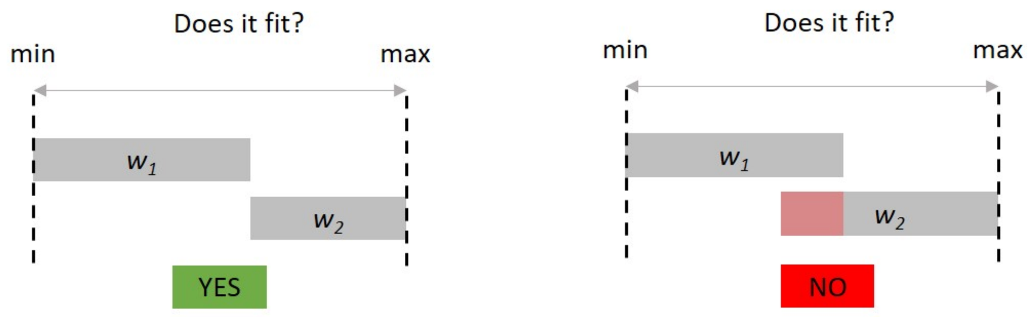

(a) New height space **(b)** New width space **(c)** New depth space

Figure 4.4: New spaces generated by packing a layer in an empty space.

For all three new spaces in Figure 4.4, it holds that the maximum coordinate equals the maximum coordinate of S^* and only the minimum coordinate needs adjusting. Let $(H^{S^*}_{\min}, W^{S^*}_{\min}, D^{S^*}_{\min})$ be the minimum coordinate and let the dimensions of the layer be given by H_l, W_l and D_l. Then, the minimum coordinates are adjusted to $(H^{S^*}_{\min}+H_l, W^{S^*}_{\min}, D^{S^*}_{\min})$, $(H^{S^*}_{\min}, W^{S^*}_{\min}+W_l, D^{S^*}_{\min})$ and $(H^{S^*}_{\min}, W^{S^*}_{\min}, D^{S^*}_{\min}+D_l)$ respectively.

As the empty maximal spaces are non-disjoint, some spaces might coincide. Therefore, it is possible that a layer L is partially packed in another empty space S', which consequently, needs to be reduced. In order to determine whether a layer L coincides with an empty space S', we need to verify if there is at least one point that is in the interior of both layer L and empty space S'. An empty space is represented by its minimum and maximum corner. By means of these two corners, we can construct a range with a lower and upper bound for each dimension of S'.

For a point to be in the interior of two spaces, if must hold that there is an x that is strictly in between the lower and upper bound of both spaces, a y that is strictly in between the left and right bound of both spaces and a z that is strictly in between the back and the front of both spaces. In other words, for each dimension the ranges of both spaces need to overlap. For all of the above to hold, each edge of a space must be parallel to an axis, which is the case in our problem at hand. Figure 4.5 graphically depicts the condition that needs to hold, in order for two ranges to overlap. Consider the two ranges with width w_1 and w_2. Then, the two ranges overlap if and only if $w_1 + w_2 > \max - \min$.



(a) $w_1 + w_2 \leq \max - \min$. so, **no overlap**

(b) $w_1 + w_2 > \max - \min$. so, **it overlaps**

Figure 4.5: Illustrative example on how to verify if two ranges overlap.

Figure 4.6 depicts the process when a box is packed and new maximal spaces are created in 2 dimensions. First, one box is packed in the upper left corner, thereby creating maximal spaces 1 and 2. Then, a second box is packed in space 2. Maximal space 2 is reduced to two new spaces 3 and 4. The last box is packed completely in space 4, hence space 4 is removed. Spaces 1 and 3 are reduced to spaces 5 and 6, respectively. As a last step, it might be that an empty space is completely included in another empty space. Therefore, we need to remove any inclusions from $\mathcal{S}$.

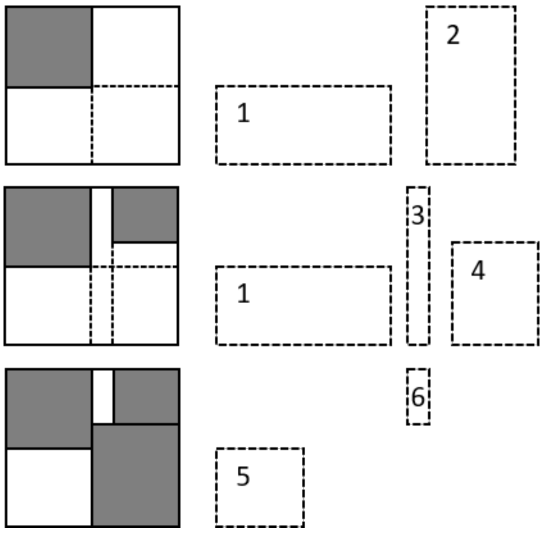


Figure 4.6: Maximal spaces in two dimensions.

4.1.2 GRASP algorithm

In Section 4.1.1 we described the constructive block heuristic which forms the basis for the reactive GRASP. A GRASP algorithm constructs starting solutions in a constructive phase by means of a constructive heuristic, followed by an improvement phase. In order to obtain different starting solutions, we randomize the selection procedure for the block configuration. All feasible layers for S^* are ordered according to one of the two criteria (volume or best-fit). Instead of selecting the best layer, we randomly pick a layer among the $100\delta\%$ best layers, where $0 \leq \delta \leq 1$.

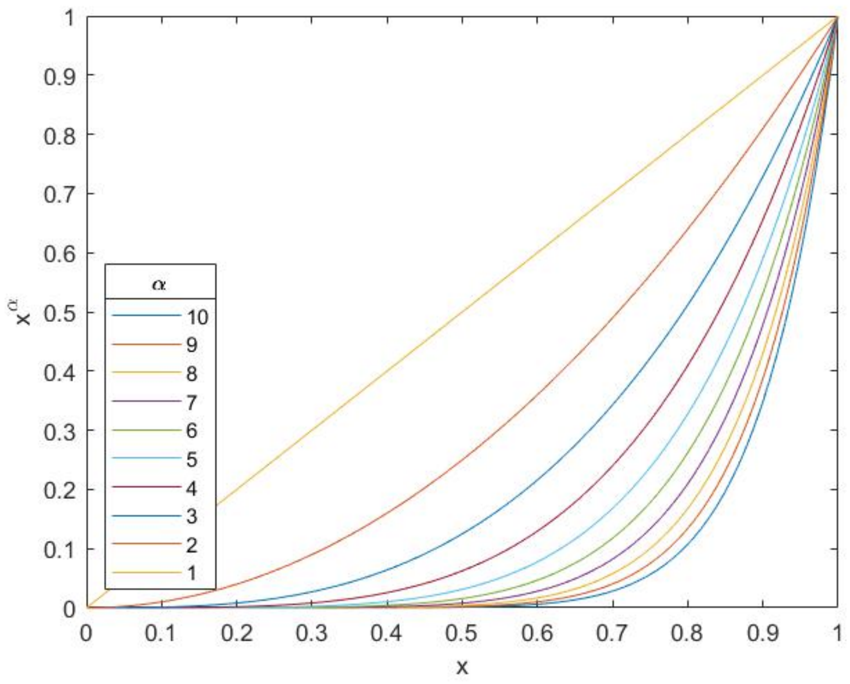


Figure 4.7: Behaviour of x^α for different values of α.

In order to determine the value for δ which gives the best results, Parreño et al. (2008) implemented a reactive GRASP. This implies that the GRASP considers several values for δ, initially all with equal probability. Then, after a certain number of iterations — *numReactive* — the values for δ which resulted in better solutions obtain higher probabilities and vice versa. In other words, the algorithm finds the best values for δ. The reactive GRASP is given in Algorithm 1, following Delorme, Gandibleux, and Rodriguez (2004). Prais and Ribeiro (2000) suggested to fix parameter α at 10. By definition it holds that $V_{best} \geq mean_\delta$, which means that the value of $\left(\frac{mean_\delta - V_{worst}}{V_{best} - V_{worst}}\right)$ is between 0 and 1. Therefore, $eval_\delta$ is close to zero for more values, as the value of α increases. This behaviour of $eval_\delta$ is illustrated in Figure 4.7. Choosing a lower value for α, results in more variation in values of δ.

Algorithm 1 Reactive GRASP

```
Initialization:
$\mathcal{D} \leftarrow \{0.1, 0.2, ..., 0.9\}$                    ▷ set of possible values for $\delta$
$V_{best} \leftarrow 0$ ; $V_{worst} \leftarrow \infty$
$n_{\delta^*} \leftarrow 0$                                  ▷ number of iterations with $\delta^*$, $\forall \delta^* \in \mathcal{D}$
$Sum_{\delta^*} \leftarrow 0$                                ▷ sum of values of solutions obtained with $\delta^*$
$P(\delta = \delta^*) \leftarrow p_{\delta^*} \leftarrow 1/|\mathcal{D}|$                       ▷ $\forall \delta^* \in \mathcal{D}$
$numIter \leftarrow 0$
While $numIter < maxIter$ do
    Choose $\delta^*$ from $\mathcal{D}$ with probability $p_{\delta^*}$
    $n_{\delta^*} \leftarrow n_{\delta^*} + 1$
    $numIter \leftarrow numIter + 1$
    Apply constructive phase with $\delta^*$ obtaining solution $S$ with objective value $V$
    If $V \geq V_{worst} + 0.5(V_{best} - V_{worst})$ then
        Apply improvement phase obtaining solution $S'$ with value $V'$
        $V \leftarrow V'$
    If $V > V_{best}$ then $V_{best} \leftarrow V$
    If $V < V_{worst}$ then $V_{worst} \leftarrow V$
    $Sum_{\delta^*} \leftarrow Sum_{\delta^*} + V$
    $mean_{\delta^*} \leftarrow \frac{Sum_{\delta^*}}{n_{\delta^*}}$
    If mod($numIter$, $numReactive$) == 0 then
        $eval_\delta \leftarrow \left(\frac{mean_\delta - V_{worst}}{V_{best} - V_{worst}}\right)^\alpha, \forall \delta \in \mathcal{D}$
        $p_\delta \leftarrow \frac{eval_\delta}{(\sum_{\delta' \in \mathcal{D}} eval_{\delta'})}, \forall \delta \in \mathcal{D}$
end While
```

In the improvement phase, we start by removing $k\%$ of the layers from the constructed starting solution. This is done in a backwards order, starting at the layer which was packed last. Then, the deterministic constructive heuristic is implemented where in each iteration the layer is selected deterministically according to one of the criteria. We need to first completely load the container before removing $k\%$ of the last packed layers, instead of stopping when the first $(1-k)\%$ layers are packed, since it is not known how many more layers are going to be constructed and packed after each iteration.

The improvement phase is only performed if the solution of the constructive phase is such that, $V \geq V_{worst} + 0.5(V_{best} - V_{worst})$, where V_{worst} and V_{best} correspond to the worst and best objective function values of solutions obtained in previous GRASP iterations. Parreño et al. (2008) did extensive research and concluded that it is best to remove the last 50% of the layers in the initial solution. Also, in the improvement phase the constructive heuristic is executed two times, i.e. one time with each criterion.

4.2 A greedy packing heuristic for solving an online variant of the 3D-CLP

In this section we present a greedy packing heuristic for solving an online variant of the 3D-CLP. The heuristic is based on the reactive GRASP proposed in Section 4.1. The algorithm should mimic the loading process at a single loading dock in a bol.com FC. Iteratively, boxes become available and are loaded subsequently. To model the iterations in which the boxes are loaded, we first present an adjusted version of the constructive heuristic in Section 4.2.1. Then, in Section 4.2.2 we present an improvement step, based on the improvement phase of the reactive GRASP. In Section 4.2.3, we discuss how to enable the heuristic to load multiple containers in parallel with boxes accumulating at one loading dock. Finally, in Section 4.2.4 we describe how to construct the order in which boxes accumulate at the loading dock, based on the distribution of different box sizes.

4.2.1 Adjusted constructive heuristic

To mimic the loading of a container, we implement an adjusted version of the constructive heuristic proposed in Section 4.1.1. The constructive heuristic consists of four steps, where step 1, 2 and 3 are repeated until no boxes remain to be packed, or there are no empty maximal spaces left which can fit at least one box. Step 0 is executed once at the beginning to initialize all variables. In the online variant of the 3D-CLP, the set of box sizes to be packed and the number of boxes to be packed, $\mathcal{B}$ and q_i, are not completely known at the start. The adjusted constructive heuristic can be described by five steps:

- *Step 0: Initialization*
  $\mathcal{S} = \{C\}$ — set of empty maximal spaces.
  $\mathcal{B} = \{\}$ — set of box sizes still to be packed.
  $q_i = 0$ — number of boxes of size i to be packed, for all $i = 1, ..., m$.
  $p_i = 0$ — number of boxes of size i packed, for all $i = 1, ..., m$.
- *Step 1: Update.* Boxes have accumulated and are ready to be packed. $\mathcal{B}$ and q_i are updated consequently.
- *Step 2: Choosing the maximal space in $\mathcal{S}$.*
- *Step 3: Choosing the boxes to pack. Update q_i, p_i and $\mathcal{B}$ if needed.*
- *Step 4: Updating the set $\mathcal{S}$.*

The adjusted constructive heuristic starts by initializing the variables. Then, in each loading iteration — after new boxes have accumulated — $\mathcal{B}$ and q_i are updated. Next, step 2 to step 4 are repeated until no boxes remain to be packed, or there are no empty maximal spaces left which can fit at least one box. When this is the case, new boxes become available and $\mathcal{B}$ and q_i are updated. This continues until there is a loading iteration in which no boxes were packed.

Another alteration, in step 2 — *choosing the maximal space in* $\mathcal{S}$ — is that all eight corners are considered for packing the next layer. However, in real-life it is not possible to pack a layer in one of the upper corners, if no other packed boxes offer support. We need to adjust the algorithm, such that only the lower four corners are considered in Equation 1.

4.2.2 Improvement step

The reactive GRASP constructs starting solutions, which are improved in the improvement phase. An improvement phase at the end, when the container is completely filled, would suggest a removal of $k\%$ of the last loaded layers. In reality, it is not reasonable to unload such a quantity of boxes, in order to reload them in an optimized manner. Rather, we implement an improvement step in each loading iteration.

Boxes in layers packed in previous iterations, could be combined with boxes of the same box size in the current iteration, if any have accumulated. As an example, consider a layer of two boxes of box size i which was packed in the last iteration. If in the current iteration, another three boxes of box size i accumulate, then the workman should consider the possibility of constructing a layer of size five. It is important to note, that only layers are considered from previous iterations, which do not have another layer placed on top of it.

To check if a layer does not have any layers packed in the empty space above, we iterate over all empty maximal spaces and check if there is an empty space which covers the entire layer up to the height of the container. Consider a layer l represented by its minimum and maximum coordinates $(H^l_{\min}, W^l_{\min}, D^l_{\min})$ and $(H^l_{\max}, W^l_{\max}, D^l_{\max})$, an empty space S with coordinates $(H^S_{\min}, W^S_{\min}, D^S_{\min})$ and $(H^S_{\max}, W^S_{\max}, D^S_{\max})$ and the container with its maximum coordinate $(H^C_{\max}, W^C_{\max}, D^C_{\max})$. Then, the following statements must hold:

$$\begin{array}{lll} H^S_{min} = H^l_{max} & W^S_{min} \leq W^l_{\min} & D^S_{\min} \leq D^l_{\min} \\ H^S_{max} = H^C_{max} & W^S_{max} \geq W^l_{\max} & D^S_{\max} \geq D^l_{\max} \end{array}$$

The improvement step is incorporated in step 1 of the adjusted constructive heuristic. After updating $\mathcal{B}$ and q_i, it is checked if there are any layers on top with a box size in $\mathcal{B}$. In this case, the layer with box size i is removed and the number of boxes in the layer is added to q_i.

4.2.3 Parallel loading

As described in Section 2.2 we consider three factors that might be of influence to the average number of boxes that are loaded in a container, namely (a) the number of boxes that accumulate in each packing iteration, (b) the number of containers that are loaded in parallel and (c) the distribution of different box sizes. We will investigate what happens with the performance of the greedy packing heuristic, when multiple containers are loaded in parallel. Assume there are n containers being loaded. Then, we need to adjust step 0, such that $\mathcal{S} = \{C_1, ..., C_n\}$. Also, we initialize a variable for the number of boxes of size i in container j, namely $p_{ij} = 0$ for all $i = 1, ..., m$ and $j = 1, ..., n$. Variable p_i can be disregarded.

4.2.4 Box order

In the online variant of the 3D-CLP, boxes iteratively become available for packing. Hence, we need to generate an order of boxes. To construct an order of boxes, we make use of the same problem instances that are solved by the reactive GRASP. Let q_i be the number of boxes that need to be packed per box size i, let $N = \sum_i q_i$ be the total number of boxes in an instance and let $P_i = q_i/N$ be the probability that the next box is a box of size i. Then, iteratively a box size is chosen with probability P_i. If the next box size is i, then $q_i = q_i - 1$ and for all i the relative probabilities, $P_i = q_i/(N-1)$, are updated accordingly.

4.3 Generation of the BRD problem classes

As described in Section 2.3 we use a well-established benchmark set to test and compare the performance of the proposed algorithms. In this section, we describe the process of constructing those benchmark problem instances as was proposed by Bischoff and Ratcliff (1995). The procedure for generating the problem instances is reproducible. This is due to a standard random number generator, capable of reproducing the same number streams on most platforms based on non-random seed numbers. The procedure for generating an instance uses the following input variables:

T_c — Cargo target volume, which is equal to the total volume of the container C.
m — Number of different box types.
$a_j, b_j, j \in \{1, 2, 3\}$ — Lower and upper limits on box dimensions.
L — Box stability limit.
s — Seed number.

Figure 4.8 depicts the complete generation process of one problem instance. The random number generator (RNG) used, is proposed in Park and Miller (1988) and is a special variant of the multiplicative congruential method. The RNG is defined by a recursive formula:

$$z_{n+1} = az_n \mod r \tag{2}$$

where $a = 16807$, $r = 2^{31} - 1$ and where we initialize the recursion with seed $z_0 = s$. To obtain uniformly distributed random numbers in the open interval (0,1), we divide z_n by r:

$$\text{rand}_n(0{,}1) = z_n/r, n = 1, 2, ... \tag{3}$$

It might happen that a product of a and z_n exceeds the maximum value for a 32-bit integer. To deal with this, one could implement a Lehmer generator. However, as we write our code in Java, we can overcome this problem by assigning z_n to a **long** variable instead of an **int** variable.

For the generation of the BRD problem instances we set the cargo target volume equal to the capacity of the container $H \times W \times D$, where $H = 220$, $W = 233$ and $D = 587$. As mentioned before, we set $m = 3, 5, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90$ and 100. The limits on box dimensions are $a_1 = 20$, $b_1 = 80$, $a_2 = 25$, $b_2 = 100$, $a_3 = 30$ and $b_3 = 120$, and the stability limit L is set to 2. The definition of the stability limit was explained in Section 2.1. Finally, the seed s depends on the problem instance number p. This is done, to enable the reproduction of a single problem instance. The seed value is given by $s = 2502505 + 100(p - 1)$. Table 4.1 shows an example of an instance generated, namely the 49th instance for $m = 8$. For each dimension the column *Vert.* indicates whether the corresponding dimension can be placed vertical, based on the stability threshold described in Section 2.1.

Table 4.1: Exmaple of BRD problem instance.

$m = 8$, $p = 49$, $s = 2507305$

Box size	Height	Vert.	Width	Vert.	Depth	Vert.	Quantity
1	45	Yes	54	Yes	91	No	13
2	72	Yes	77	Yes	105	Yes	15
3	48	Yes	78	Yes	79	Yes	10
4	59	Yes	76	Yes	109	Yes	12
5	30	Yes	37	Yes	48	Yes	13
6	27	Yes	37	Yes	44	Yes	9
7	54	Yes	76	Yes	79	Yes	17
8	20	Yes	78	No	116	No	16

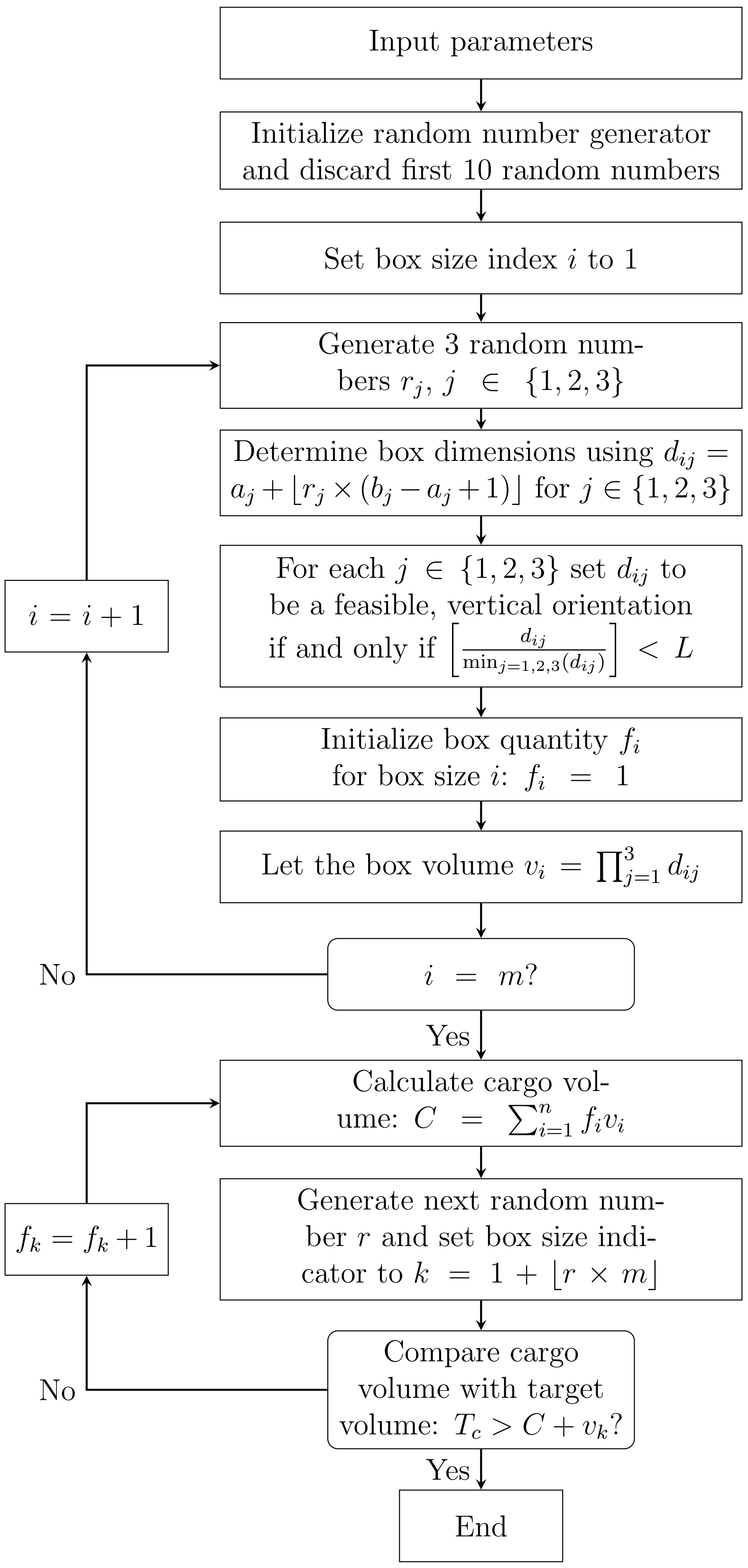

Input parameters
Initialize random number generator and discard first 10 random numbers
Set box size index i to 1
Generate 3 random numbers r_j, $j \in \{1,2,3\}$
Determine box dimensions using $d_{ij} = a_j + \lfloor r_j \times (b_j - a_j + 1) \rfloor$ for $j \in \{1,2,3\}$
For each $j \in \{1,2,3\}$ set d_{ij} to be a feasible, vertical orientation if and only if $\left\lceil \frac{d_{ij}}{\min_{j=1,2,3}(d_{ij})} \right\rceil < L$
$i = i + 1$
Initialize box quantity f_i for box size i: $f_i = 1$
Let the box volume $v_i = \prod_{j=1}^{3} d_{ij}$
$i = m$?
No
Yes
Calculate cargo volume: $C = \sum_{i=1}^{n} f_i v_i$
Generate next random number r and set box size indicator to $k = 1 + \lfloor r \times m \rfloor$
$f_k = f_k + 1$
Compare cargo volume with target volume: $T_c > C + v_k$?
No
Yes
End

Figure 4.8: Generation of a single BRD problem instance.

4.4 Linear regression model

As described in Section 1, the main goal of this research is to develop a linear model, which accurately computes the average number of boxes that are packed in a container at a single loading dock in a bol.com FC. In Section 4.2 we derived a greedy packing heuristic to solve an online variant of the 3D-CLP introduced in Section 2.2. In Section 4.4.1 we explain how this heuristic is used to generate synthetic data. Then, in Section 4.4.2 we concisely introduce the dependent variable and input variables of the linear regression model. Finally, in Section 4.4.3 we elaborate on the generation of problem instances used for constructing synthetic training and test data.

4.4.1 Generation of synthetic data

The proposed greedy packing heuristic is designed to mimic the loading process at a single loading dock in a bol.com FC. We are going to use the heuristic to pack many containers and generate observations of Y_i, i.e. the average number of boxes packed in a container corresponding to an input group X_i. Remember that an input group X_i consists of (a) the number of boxes that accumulate in each loading iteration, (b) the number of containers that are loaded in parallel and (c) the distribution of different box sizes.

The greedy packing heuristic processes as input a predefined order in which boxes arrive at the loading dock. The hyper parameter settings of the heuristic consist of (a) the number of boxes that accumulate in each loading iteration and (b) the number of containers that are loaded in parallel. The order in which the boxes arrive, is randomly generated based on the distribution of different box sizes. The output of the heuristic is one packed container, or multiple containers if the containers are loaded in parallel.

The idea is that we generate many box orders per group of inputs and let the heuristic process these orders. This way, we can generate an observation for Y_i, by averaging the number of boxes in a container over all containers packed, corresponding to the specific group of inputs X_i. This idea follows from the law of large numbers, which states that the average of the results obtained from a large number of trials should be close to the expected value and tends to become closer to the expected value as more trials are performed. In other words, if we let the heuristic pack many containers by processing box orders which are generated based on the distribution of X_i, then eventually the average number of boxes over all containers should be close to expected value corresponding to input group X_i.

4.4.2 Ordinary least squares model

In order to construct and assess the performance of a linear regression model, we need synthetic training and test data. To generate synthetic data, we need problem instances which can be converted to distributions of box sizes where the sum of all shares equals one. We construct many instances and distributions. For each input group X_i consisting of a distribution and a hyper parameter setting, we produce an observation for Y_i.

Finally, when all synthetic training data is generated we construct the linear regression model by regressing Y on the distribution of box sizes and two dummy variables, $D_{containers}$ and D_{boxes}. We will do computational experiments in Section 5 to determine the settings for the two dummy variables.

4.4.3 Generation of the bol.com problem instances

For the generation of bol.com problem instances, we use an adjusted version of the process explained in Figure 4.8. As described before in Section 2.3, we consider one problem class consisting of 14 different box sizes, given in Table 2.1, and a container as was shown in Figure 1.2.

With respect to the input variables, we consider T_c, m and s. Cargo target volume T_c is set equal to $H \times W \times D$, where $H = 180$, $W = 75$ and $D = 75$. The number of different box sizes m is set to 14, as we only consider the bol.com problem class. Let m_s and $m_b = m - m_s$ be the number of smallest and biggest box sizes. We divide the set of 14 box sizes evenly in 7 smallest and 7 biggest box sizes. We use the same random number generator as was used for the generation of the BRD instances. We initialize the box quantity for each box size i by setting $f_i = 1$ and we set v_i equal to the volume of box size i. Let L_s be the share of m_s in the total distribution of box sizes. In each iteration we generate a random number r_1 in the open interval (0,1). If $r_1 < L_s$, the next box size is randomly chosen out of the set of smallest boxes. If $r_1 \geq L_s$, the next box size is randomly chosen out of the set of biggest boxes. The complete process is depicted in Figure 4.9.

Table 4.2: Example of bol.com problem instance.

$L_s = 2/3$, $p = 5$, $s = 2502905$

Box size	Height	Vert.	Width	Vert.	Depth	Vert.	Quantity
Soap box	20	Yes	14	Yes	6	Yes	4
Match box	20	Yes	12	Yes	8	Yes	5
Parfumebox	24	Yes	16	Yes	8	Yes	2
Sound box	28	Yes	20	Yes	8	Yes	1
Lunch box	24	Yes	20	Yes	12	Yes	5
Sandbox	31	Yes	23	Yes	14	Yes	4
Outbox	28	Yes	24	Yes	20	Yes	3
Colorbox	40	Yes	32	Yes	12	Yes	2
PA-8 Juke box	36	Yes	28	Yes	22	Yes	2
Pillowbox	54	Yes	40	Yes	12	Yes	2
Shoebox	48	Yes	38	Yes	20	Yes	2
Pandora's box	45	Yes	35	Yes	30	Yes	2
Toolbox	64	Yes	48	Yes	22	Yes	3
PA-14 Sky box	70	Yes	50	Yes	35	Yes	3

In Table 4.2 an example of a bol.com problem instance is given. Notice that each dimension of every box can be placed vertically. We do not consider a stability constraint, since in reality the containers are loaded with enough boxes, such that the sides of the wheeled container and the other boxes offer sufficient support and stability.

We construct three different problem classes. We consider one where the distribution between smallest and biggest box sizes is evenly distributed, one where the smallest box sizes have $2/3$ of the total share, and one where the biggest box sizes have $2/3$ of the total share. For the synthetic training data we construct 100 problem instances per problem class, thus in total 300 instances. For this process we use the same seed s as was used for the generation of the BRD instances. For the synthetic test data we construct 50 problem instances for each value of L_s but we used another seed $s = 1996 + 100(p-1)$.

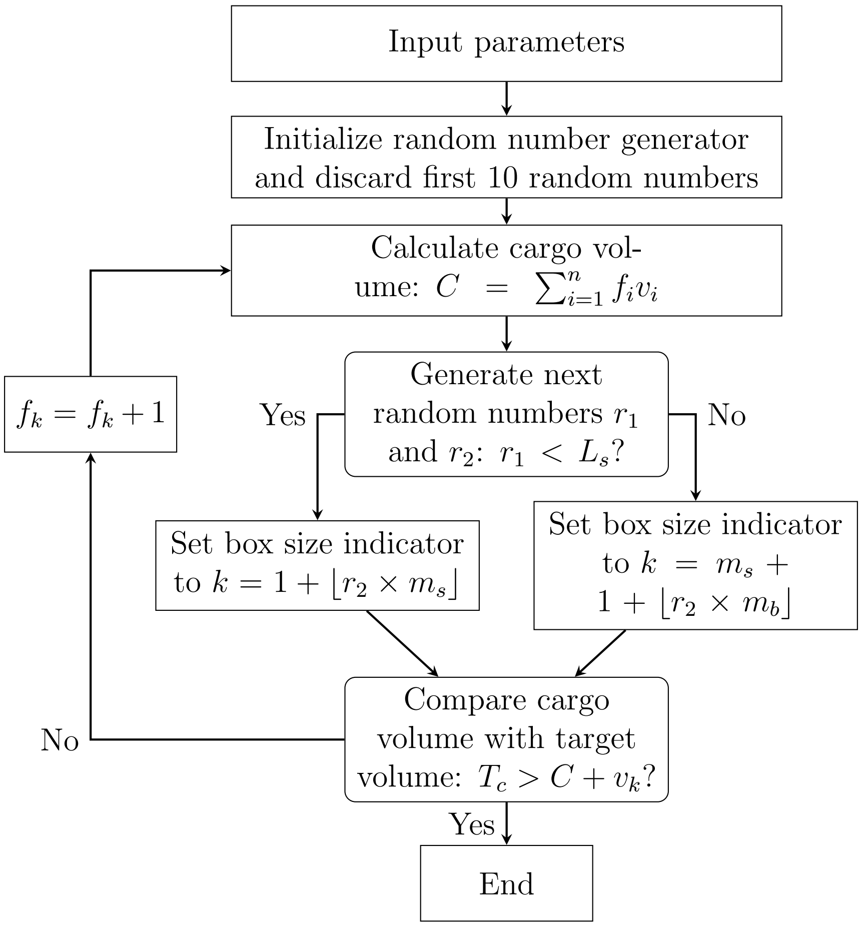

Input parameters
Initialize random number generator and discard first 10 random numbers
Calculate cargo volume: $C = \sum_{i=1}^{n} f_i v_i$
Generate next random numbers r_1 and r_2: $r_1 < L_s$?
Yes
No
$f_k = f_k + 1$
Set box size indicator to $k = 1 + \lfloor r_2 \times m_s \rfloor$
Set box size indicator to $k = m_s + 1 + \lfloor r_2 \times m_b \rfloor$
Compare cargo volume with target volume: $T_c > C + v_k$?
No
Yes
End

Figure 4.9: Generation of a single bol.com problem instance.

5 Computational Experiments

First, in Section 5.1 we do several computational experiments to examine if some parameter settings can be adjusted in order to improve the performance of the reactive GRASP. Then, in Section 5.2 we present a sensitivity analysis on the greedy packing heuristic to get insight into the behaviour and performance of the derived heuristic. Finally, in Section 5.3 we construct and test a linear regression model for predicting the average number of boxes that are packed in a container at a loading dock of a bol.com FC. We also show the difference in solution quality, between the solutions obtained by the greedy packing heuristic and the GRASP approach. The algorithms proposed in this paper are coded in Java 1.8.0 and run on an Intel Core i5-6200U at 2.30 GHz with 8.00 GB of RAM.

5.1 Hyper parameter tuning for reactive GRASP

In Parreño et al. (2008) the authors made conclusions with respect to several (parameter) settings for the reactive GRASP. The constructive phase builds layers in step 2 and as an objective the increase of volume is used. The value of δ is chosen by the reactive GRASP approach. In the improvement phase the last 50% of layers are removed and the deterministic constructive heuristic is applied two times, once with each objective.

In Parreño et al. (2008) the authors present results for 5000 iterations. Also, the number of iterations after which the relative probabilities for δ are re-evaluated is set to $numReactive = 500$. If this number is decreased, it might be that there is a quicker convergence of δ to values that produce better solutions. In Parreño et al. (2008) the authors used a value of 10 for α in the reactive GRASP. As mentioned before in Section 4.1, lower values for α will result in more variation in δ.

In the following, we present results of computational experiments performed in order to possibly improve the parameter settings of the reactive GRASP for $maxIter$, $numReactive$ and α.

5.1.1 Setting $maxIter$ and $numReactive$

Table 5.1 shows results of the reactive GRASP obtained with different values for α, $numReactive$ and $maxIter$. This table, like all other tables in this section unless mentioned otherwise, shows the percentage of container volume occupied by boxes in the solution. The percentages in Table 5.1 are the average results of the first 10 instances in the BRD_{06} problem class. We know that 5000 iterations of the reactive GRASP will result in solutions at least as good as the solutions obtained with 500 iterations. We want to test whether one parameter setting obtains significantly better results.

We start with a non-parametric analysis. A Friedman Test was conducted on nine subjects, i.e. the different values for α with 10 instances each. In addition, we do a para-

Table 5.1: Results of reactive GRASP averaged over first 10 instances of BRD_{06}.

$maxIter$	500	5000	
$numReactive$	50	50	500
$\boldsymbol{\alpha}$	①	②	③
2	91.58	**92.44**	91.98
3	91.07	**92.16**	91.98
4	91.51	**92.35**	92.17
5	91.26	**92.40**	92.35
6	91.79	**92.59**	92.20
7	91.85	**92.73**	92.46
8	91.76	92.54	**92.73**
9	91.55	**92.70**	92.51
10	91.82	**92.51**	92.40

Note. The best values appear in bold.

metric analysis of variance for repeated measures (MANOVA). These tests are done, to test whether different settings for $maxIter$ and $numReactive$ lead to statistically significant differences in volume use. Based on the Friedman test statistic $Q = 16.22$ ($p < 0.001$) and MANOVA test statistic $F = 97.00$ ($p < 0.001$), we conclude that this is the case.

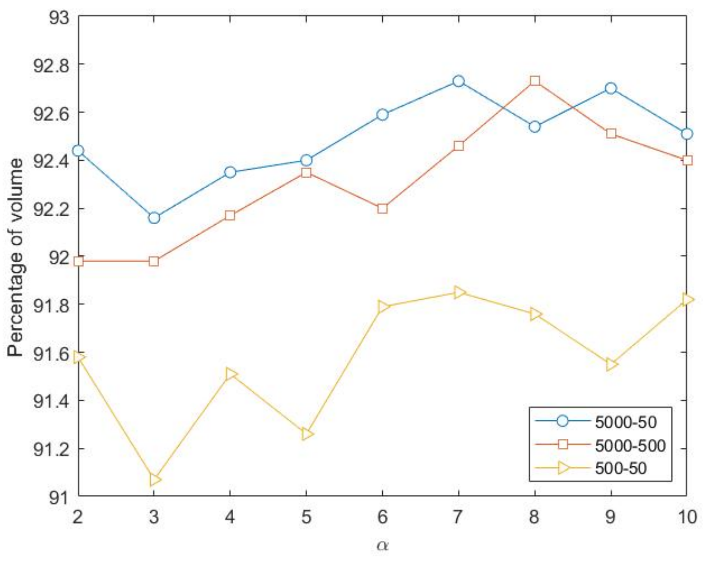


Figure 5.1: Comparing volume use and parameter settings of reactive GRASP.

For pairwise comparisons the Wilcoxon Signed Rank Test is performed. Based on the Wilcoxon Test statistics, we can conclude that setting the maximum number of iterations in the reactive GRASP to 5000 and re-evaluating the relative probabilities of δ after 50 iterations, result in significantly higher volume use ($p < 0.05$). Figure 5.1 graphically depicts the difference in performance between the three parameter settings. Comparing setting 2 and 3, we note that out of 90 subjects, setting 2 obtains better solutions in 58 cases, setting 3 obtains better solutions in 29 cases and 3 times the performance was equal.

A first consequence of this analysis, is that we can conclude that it is best to re-evaluate the relative probabilities for δ after every 50 instead of after every 500 iterations as was proposed by Parreño et al. (2008). 5000 iterations result in significantly better solutions than 500 iterations. However, in the following we set the maximum number of iterations to 500, unless mentioned otherwise, as we are going to do a limited computational study.

5.1.2 Setting α

A next step is to determine an appropriate value for α, i.e. the parameter in the reactive GRASP used for updating the relative probabilities for each δ. As described before in Section 4.1.2, lower values for α result in more variation in δ. We performed a computational study using only the first 10 instances of each BRD problem class. The goal of this analysis is to determine whether there is a significant difference in performance between settings for α. Table 5.2 shows the average volume use obtained with each value for α, ranging from 2 to 10. We again perform a Friedman, MANOVA and Wilcoxon Signed Rank test both on all BRD problem classes, as well as on the weakly and strongly heterogeneous classes separately.

Table 5.2: Results of reactive GRASP averaged over first 10 instances per BRD class.

	α								
Problem class	2	3	4	5	6	7	8	9	10
BRD$_{01}$	94.65	94.17	94.14	94.59	94.27	94.52	94.32	94.38	94.26
BRD$_{02}$	93.51	93.73	93.48	93.64	93.61	93.86	93.78	93.89	93.77
BRD$_{03}$	92.67	92.54	92.64	92.79	93.21	92.83	92.76	92.73	92.70
BRD$_{04}$	92.05	92.20	92.15	92.40	92.74	92.16	92.76	92.74	92.59
BRD$_{05}$	91.85	91.97	92.12	91.94	92.15	92.13	92.36	92.22	92.15
BRD$_{06}$	91.58	91.07	91.51	91.26	91.79	91.85	91.76	91.55	91.82
BRD$_{07}$	90.52	90.43	90.44	90.55	90.72	90.51	91.00	91.04	90.47
BRD$_{08}$	89.87	89.84	89.97	90.03	89.88	90.09	89.80	90.28	89.91
BRD$_{09}$	89.15	89.29	89.34	89.43	89.45	89.56	89.50	89.23	89.06
BRD$_{10}$	88.52	88.96	88.94	88.95	88.82	88.74	88.86	88.84	88.77
BRD$_{11}$	88.25	88.33	88.36	88.38	88.43	88.18	88.45	88.29	88.39
BRD$_{12}$	87.91	88.06	88.04	88.09	87.69	88.11	87.92	88.10	88.02
BRD$_{13}$	87.60	87.81	87.78	87.75	87.72	87.60	87.70	87.79	87.62
BRD$_{14}$	87.36	87.62	87.63	87.52	87.50	87.59	87.62	87.45	87.53
BRD$_{15}$	87.53	87.57	87.84	87.89	87.85	87.88	87.70	87.89	87.77
Mean	90.20	90.24	90.29	90.35	90.39	90.37	90.42	90.43	90.32

Note. maxIter = 500 and *numReactive* = 50.

For the complete BRD benchmark set, we can conclude that different values for α lead to statistically significant differences in volume use, with Friedman test statistic $Q = 23.91$ ($p < 0.005$) and MANOVA test statistic $F = 3.61$ ($p < 0.001$). The tests for pairwise comparisons showed that the reactive GRASP obtained significantly higher volume use with $\alpha \in \{4, 5, 6, 7, 8, 9, 10\}$ than $\alpha \in \{2\}$ ($p < 0.05$). The test concludes that $\alpha \in \{9\}$

results in significant higher volume use compared to $\alpha \in \{2, 3, 10\}$ ($p < 0.05$).

For both the weakly and strongly heterogeneous classes the Friedman and MANOVA test statistics show that there is a significant difference in performance. For BRD_{01}-BRD_{07} and $\alpha \in \{6, 7, 8, 9, 10\}$ we see that the reactive GRASP obtains significantly better volume use than $\alpha \in \{3, 4\}$ ($p < 0.05$). For BRD_{08}-BRD_{15} and $\alpha \in \{3, 4, 6, 7, 8, 9, 10\}$, the reactive GRASP resulted in significant higher volume use than $\alpha \in \{2\}$ ($p < 0.05$). Based on these results it is difficult to make a clear conclusion on the best value for α. For comparison of the performance of the reactive GRASP, we set $\alpha = 9$.

5.1.3 Reactive GRASP vs. GRASP

As described in Section 5.1.2, no unique value of α exists, which results in the highest volume use. The results do indicate a better performance with higher values for α. This suggests that it is preferred to have less variation in δ. Perhaps, it is best to not let the reactive GRASP choose a value for δ, but to set δ to a certain value in advance. Figure 5.2 shows lines per value for α with the average volume use obtained with a value for δ in the reactive GRASP. These averages are obtained on the BRD benchmark set and the first 10 instances of each problem class. We can clearly see a decreasing trend in volume use as δ increases. The best results were on average obtained by $\delta = 0.1$.

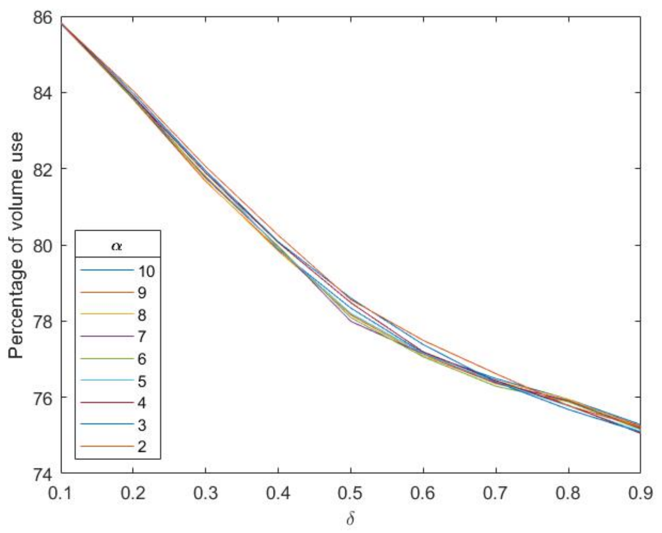


Figure 5.2: Results of reactive GRASP averaged over first 10 instances of BRD classes 1-15.

As described, it might be better to set a value for δ in advance instead of running a reactive GRASP to choose values for δ. Therefore, we test what happens with the performance, if we implement a 'normal' GRASP with $\delta = 0.1$. Hence, we do not need to consider *numReactive* and α anymore. Table 5.3 shows the average volume use of the GRASP over the first 10 instances per BRD problem class with a limit of 500 iterations. It also shows the results belonging to $\alpha = 9$ form Table 5.2.

Table 5.3: GRASP compared with reactive GRASP: results averaged over first 10 instances per BRD class.

Problem class	GRASP	reactive GRASP
BRD_{01}	94.36	**94.38**
BRD_{02}	**93.97**	93.89
BRD_{03}	**93.13**	92.73
BRD_{04}	**92.85**	92.74
BRD_{05}	**92.27**	92.22
BRD_{06}	**91.75**	91.55
BRD_{07}	91.04	91.04
BRD_{08}	90.18	**90.28**
BRD_{09}	**89.36**	89.23
BRD_{10}	88.84	88.84
BRD_{11}	**88.62**	88.29
BRD_{12}	**88.15**	88.10
BRD_{13}	**87.92**	87.79
BRD_{14}	**87.76**	87.45
BRD_{15}	**87.94**	87.89

Note. The best values appear in bold; $maxIter = 500$ and $numReactive = 50$.

At first sight, it seems that the GRASP outperforms the reactive GRASP. We perform a paired t-test and a Wilcoxon Signed Rank test. Based on the test statistics of both tests, we conclude that the normal GRASP obtains significantly higher volume use than the reactive GRASP ($p < 0.025$).

5.1.4 Comparison with the original algorithm

We have shown that the normal GRASP outperforms the reactive GRASP proposed in Section 4.1, for which we adjusted the hyper parameter setting. Finally, we present several outcomes in Table 5.4 in order to compare the performance of the GRASP with the results of the original reactive GRASP proposed in Parreño et al. (2008).

In previous sections we did limited computational studies where the maximum number of iterations was set to 500. This table shows the results of the normal GRASP with a maximum of 500 and 5000 iterations. It also makes a distinction between, when the stability constraint is and is not enforced. Lastly, it shows the results from Parreño et al. (2008), which are the average results of five runs of the reactive GRASP with 5000 iterations. All results are obtained over the first 10 instances of each BRD problem class.

We want to examine whether the performance of the GRASP improves significantly when the maximum number of iterations is increased, both with and without the stability constraint. We run a paired t-test and a Wilcoxon Signed Rank test on parameter setting one and two and on setting three and four. We conclude that the GRASP obtains significantly higher volume use with 5000 iterations, both with and without the stability constraint ($p < 0.005$).

Table 5.4: Results of GRASP averaged over first 10 instances per BRD class compared with results from Parreño et al. (2008).

Stability constraint	without		with		
Algorithm	GRASP		GRASP		Parreño
maxIter	500	5000	500	5000	5000
Problem class	①	②	③	④	⑤
BRD_{01}	94.36	94.52	92.68	**92.79**	92.71
BRD_{02}	93.97	94.80	93.20	**94.06**	94.00
BRD_{03}	93.13	93.97	92.65	**93.63**	93.58
BRD_{04}	92.85	93.71	92.44	**93.44**	93.25
BRD_{05}	92.27	93.26	92.15	**93.20**	93.00
BRD_{06}	91.75	92.89	91.40	92.61	**92.72**
BRD_{07}	91.04	91.83	90.75	**91.86**	91.70
BRD_{08}	90.18	90.76	89.99	**90.93**	90.81
BRD_{09}	89.36	90.19	89.32	90.34	**90.45**
BRD_{10}	88.84	89.63	88.91	89.59	**89.70**
BRD_{11}	88.62	89.43	88.68	89.15	**89.36**
BRD_{12}	88.15	88.84	88.16	88.81	**88.95**
BRD_{13}	87.92	88.69	87.86	**88.50**	88.34
BRD_{14}	87.76	88.35	87.73	**88.24**	**88.24**
BRD_{15}	87.94	88.44	88.01	**88.49**	88.33
Mean 1-8	92.44	93.22	91.91	**92.81**	92.72
Mean 9-15	88.37	89.08	88.38	89.02	**89.05**
Overall mean	90.54	91.29	90.26	**91.04**	91.01

Note. The best values with stability constraint appear in bold.

Also, we want to test whether the results of the GRASP with 5000 iterations are significantly better than those in Parreño et al. (2008). For the complete BRD benchmark set as well as the strongly heterogeneous problem classes, we cannot reject the null hypothesis that the means of parameter setting four and five are equal. For the weakly heterogeneous problem classes BRD_{01} to BRD_{08}, only based on the t-test we can conclude that the GRASP outperforms the reactive GRASP proposed by Parreño et al. (2008) ($p < 0.05$).

In short, based on the test results we cannot conclude that a significant difference in performance exists between the normal GRASP and the reactive GRASP proposed by Parreño et al. (2008) for any type of problem instance. However, we do conclude that for weakly heterogeneous problem instances ranging from three to thirty different box types, it is best to use the GRASP for maximizing the volume use of a container.

5.2 Sensitivity analysis on the greedy packing heuristic

In the previous section we have done computational experiments on the reactive GRASP. We concluded that it is better to use a normal GRASP with $\delta = 0.1$ for weakly heterogeneous problem instances. In this section, we do a sensitivity analysis on the greedy packing heuristic used for solving an online variant of the 3D-CLP.

As described before in Sections 4.2.3 and 4.2.4, there are two parameters for which the value can be adjusted in order to properly represent a certain setting of an online 3D-CLP. Namely, the number of boxes that becomes available for packing in each loading iteration and the number of containers that are loaded in parallel. In the following we may refer to these values with *#boxes* and *#containers*.

For each problem instance in the BRD classes we construct a distribution of box sizes. Based on each distribution, we generate 10 different box orders. We only consider the first 10 instances of each BRD class. Thus, in total 1500 box orders per parameter setting are processed by the greedy packing heuristic. Table 5.5 reports the average volume use per problem class for each parameter setting.

Table 5.5: Results of greedy packing heuristic averaged over first 10 instances per BRD class.

#boxes	1			5			10		
#containers	1	2	3	1	2	3	1	2	3
Problem class	①	②	③	④	⑤	⑥	⑦	⑧	⑨
BRD_{01}	81.31	83.44	86.72	84.80	86.61	87.57	83.69	87.17	86.93
BRD_{02}	79.32	83.88	86.40	81.57	86.15	87.59	82.02	86.37	87.45
BRD_{03}	76.18	82.58	85.14	79.94	84.08	86.68	80.18	84.97	86.93
BRD_{04}	75.23	81.28	84.27	79.32	82.95	85.88	80.08	83.27	86.27
BRD_{05}	75.57	80.01	83.41	79.17	82.30	84.94	79.57	82.68	85.21
BRD_{06}	74.79	78.54	81.78	78.82	81.46	83.55	79.31	81.73	84.37
BRD_{07}	74.57	77.52	80.29	78.74	80.35	82.63	79.06	81.05	82.95
BRD_{08}	73.95	76.45	78.38	78.72	80.26	81.69	79.27	80.66	81.86
BRD_{09}	72.73	75.59	77.41	78.12	80.00	81.07	78.38	80.21	81.27
BRD_{10}	73.14	75.17	76.99	78.41	80.03	80.77	78.86	80.41	81.18
BRD_{11}	72.39	76.11	76.26	78.35	80.02	80.94	78.65	80.49	81.04
BRD_{12}	71.78	75.18	76.04	78.75	80.05	80.80	79.11	80.41	81.13
BRD_{13}	72.37	74.14	75.84	78.66	80.12	80.84	79.17	80.17	80.97
BRD_{14}	73.31	74.74	75.94	78.91	80.13	80.91	79.06	80.35	81.04
BRD_{15}	72.01	73.65	74.85	78.81	80.31	80.89	79.22	80.62	81.19
Mean	74.58	77.88	79.98	79.41	81.66	83.12	79.71	82.04	83.32

Note. For each problem instance 10 box orders are processed by greedy packing heuristic.

From Table 5.5 and the heat map displayed in Figure 5.3 we can see that the results are as expected. We see that the volume use increases as the number of boxes that iteratively becomes available is increased. Also, we see an increase in volume use when the number of containers that are loaded in parallel is increased.

We are interested to examine whether the relative increase in volume use varies with respect to the type of problem class and the parameter setting. We first examine the effects when *#containers* is incremented, while keeping *#boxes* equal in Section 5.2.1. Then, we look into the effects when *#boxes* is increased, while keeping *#containers* equal in Section 5.2.2.

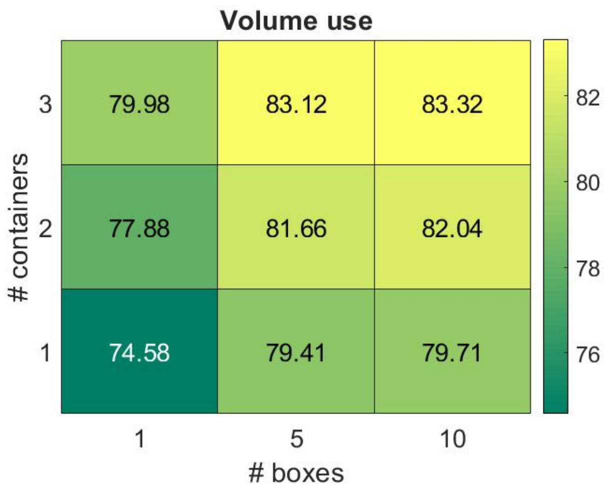


Figure 5.3: Heat map of the mean volume use in percentages from Table 5.5.

5.2.1 Marginal effects of *#containers*

In this section, we examine the effects on the results of the greedy packing heuristic, when the number of containers that are loaded in parallel is increased, while keeping the number of boxes that becomes available in each loading iteration constant. Table 5.6 shows the relative changes in percentages in volume use for each problem class, when *#boxes* is kept constant and *#containers* is increased.

Table 5.6: Relative changes in percentages of the results of the greedy packing heuristic.

#boxes	1			5			10		
#containers	1→2	2→3	1→3	1→2	2→3	1→3	1→2	2→3	1→3
Problem class	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
BRD_{01}	2.61	3.94	6.65	2.14	1.11	3.27	4.17	-0.27	3.88
BRD_{02}	5.75	3.00	8.93	5.62	1.66	7.38	5.30	1.26	6.62
BRD_{03}	8.40	3.10	11.76	5.19	3.09	8.43	5.97	2.31	8.42
BRD_{04}	8.05	3.67	12.02	4.57	3.53	8.26	3.97	3.60	7.72
BRD_{05}	5.87	4.25	10.37	3.95	3.21	7.28	3.91	3.06	7.09
BRD_{06}	5.01	4.12	9.34	3.36	2.56	6.01	3.05	3.23	6.38
BRD_{07}	3.95	3.57	7.67	2.05	2.83	4.94	2.52	2.33	4.91
BRD_{08}	3.38	2.53	5.99	1.95	1.78	3.76	1.76	1.48	3.27
BRD_{09}	3.94	2.41	6.44	2.40	1.35	3.78	2.33	1.32	3.69
BRD_{10}	2.77	2.42	5.26	2.06	0.93	3.00	1.96	0.96	2.94
BRD_{11}	5.13	0.21	5.35	2.13	1.15	3.31	2.34	0.68	3.04
BRD_{12}	4.73	1.15	5.94	1.65	0.94	2.60	1.64	0.90	2.56
BRD_{13}	2.45	2.29	4.80	1.85	0.90	2.77	1.25	1.00	2.27
BRD_{14}	1.95	1.60	3.58	1.55	0.97	2.53	1.63	0.86	2.50
BRD_{15}	2.28	1.63	3.94	1.91	0.72	2.64	1.78	0.70	2.49

Note. Based on results from Table 5.5.

Figure 5.4 shows three lines. Each line represents the average of three columns in Table 5.6, namely (a) columns 1, 4 and 7, (b) columns 2, 5 and 8 and (c) columns 3, 6 and 9. We see that the relative change in volume use is not linear with the number of containers. In general, the highest increase is seen from one to two containers. A paired t-test proves that the increase from one to two containers is significantly higher than the increase from two to three containers. This suggests that there is a tipping point where adding an extra container results in a negligible increase in volume use.

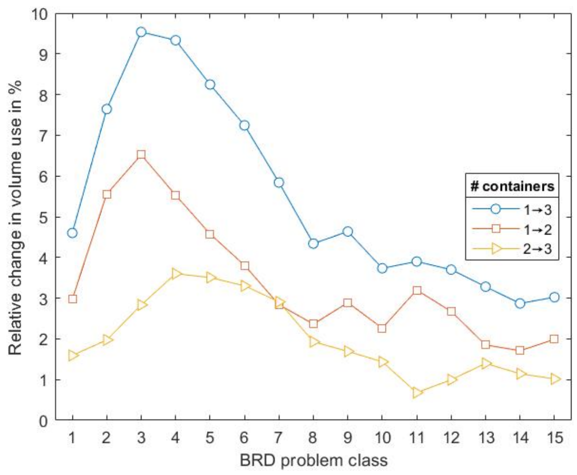


Figure 5.4: Relative change averaged over all values for *#boxes*.

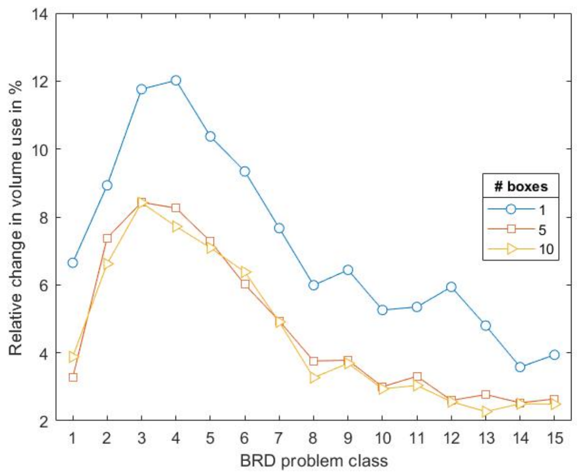


Figure 5.5: Relative change of $1 \rightarrow 3$ containers for each value of *#boxes*.

Figure 5.5 contains a line for each value of *#boxes*, with the total relative change in volume use when incrementing the number of containers from one to three. We note that the effect of increasing the number of containers is much higher when *#boxes* equals one compared to the other two lines. Based on a paired t-test, we can conclude that there is no significant difference between the other two lines.

From columns 3, 6 and 9, we also see that the effect of adding an extra container generally decreases as the number of box sizes in a problem instance increases. A possible explanation for this could be, that the probability decreases that an improvement step in the greedy packing heuristic can be be performed. As the number of box sizes increases, it is less likely that one of the boxes that became available for packing, is of the same box size as those on top in the containers.

5.2.2 Marginal effects of *#boxes*

In this section, we examine the effects on the results of the greedy packing heuristic, when the number of boxes that iteratively becomes available for packing is increased, while keeping the number of containers that are loaded in parallel constant. Table 5.7 summarizes the relative changes in percentages. In Figure 5.6 three lines are shown. Each line represents the average of three columns in Table 5.7, namely (a) columns 1, 4 and 7, (b) columns 2, 5 and 8 and (c) columns 3, 6 and 9. We can clearly see that the relative change in volume use caused by incrementing *#boxes* from five to ten, is negligible compared to the increase caused by incrementing from one to five.

Table 5.7: Relative changes in results of greedy packing heuristic.

#containers	1			2			3		
#boxes	1→5	5→10	1→10	1→5	5→10	1→10	1→5	5→10	1→10
Problem class	①	②	③	④	⑤	⑥	⑦	⑧	⑨
BRD_{01}	4.29	-1.31	2.92	3.81	0.65	4.48	0.98	-0.73	0.25
BRD_{02}	2.84	0.55	3.41	2.71	0.25	2.97	1.38	-0.16	1.22
BRD_{03}	4.94	0.30	5.26	1.82	1.05	2.89	1.81	0.29	2.11
BRD_{04}	5.45	0.96	6.46	2.06	0.38	2.44	1.91	0.45	2.37
BRD_{05}	4.76	0.50	5.29	2.87	0.46	3.34	1.84	0.32	2.16
BRD_{06}	5.38	0.63	6.04	3.72	0.33	4.06	2.17	0.98	3.17
BRD_{07}	5.59	0.41	6.02	3.65	0.87	4.56	2.91	0.39	3.31
BRD_{08}	6.46	0.69	7.20	4.99	0.51	5.52	4.22	0.21	4.44
BRD_{09}	7.42	0.34	7.78	5.83	0.27	6.11	4.73	0.25	4.99
BRD_{10}	7.20	0.57	7.82	6.46	0.48	6.97	4.91	0.51	5.45
BRD_{11}	8.23	0.38	8.64	5.14	0.58	5.75	6.14	0.11	6.26
BRD_{12}	9.71	0.45	10.21	6.49	0.44	6.95	6.26	0.40	6.69
BRD_{13}	8.69	0.66	9.40	8.06	0.06	8.12	6.58	0.17	6.76
BRD_{14}	7.63	0.19	7.84	7.21	0.27	7.51	6.54	0.17	6.72
BRD_{15}	9.44	0.52	10.01	9.05	0.39	9.47	8.08	0.36	8.47

Note. Based on results from Table 5.5.

We can also deduct from the figure that the effect of incrementing *#boxes*, is positively correlated with the number of box sizes in a problem class. There is an explanation for this observation. Consider two problem instances, one with one box size and one with 100 box sizes and consider a scenario in which one box becomes available for packing in each iteration. For the first instance, we know that choosing to pack another box, cannot result in a higher increase in volume use. However, for the second instance it might be that boxes after the one that is available, result in a higher increase in volume use. Hence, an instance with more box sizes profits more from the increase in *#boxes*.

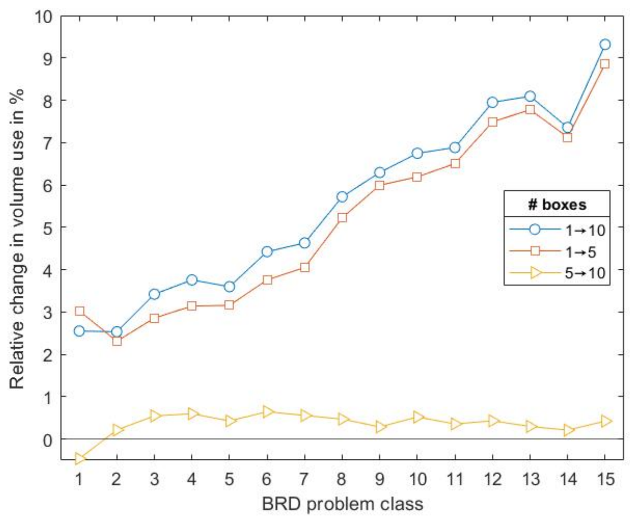


Figure 5.6: Relative change averaged over all values for *#containers*.

5.3 Analysis of linear regression model

In this section we first discuss the synthetic data that is generated for constructing the linear regression model in Section 5.3.1. Then, we present the constructed linear model in Section 5.3.2. Finally, we present the out-of-sample test results for assessing the quality of the linear model in Section 5.3.3.

5.3.1 Sythetic data generation results

In Section 5.2 we studied and analysed the greedy packing heuristic on the BRD problem classes. We have shown that the volume use increases as the number of containers that are loaded in parallel increases. It seems however that a tipping point exists, when adding an extra container has a negligible effect on the volume use. For the number of boxes that becomes available for packing, this tipping point is around five. We saw that the increase in volume, caused by incrementing *#boxes* from five to ten, was negligible compared to the increase caused by incrementing from one to five.

These outcomes are positive, considering the online variant of the 3D-CLP at the bol.com FC. In Section 2.2 we described the process in which input is released. Boxes are almost immediately taken of the conveyor belt, hence it is more reasonable to assume that input is released in groups of five than in groups of ten. For the bol.com case, we only examine the performance for one and five boxes and one and three containers. Therefore, the dummy variables can be defined as follows:

$$D_{containers} = \begin{cases} 1, & \text{if } \#containers \text{ equals } 1 \\ 0, & \text{if } \#containers \text{ equals } 3, \end{cases} \qquad D_{boxes} = \begin{cases} 1, & \text{if } \#boxes \text{ equals } 1 \\ 0, & \text{if } \#boxes \text{ equals } 5. \end{cases}$$

In Section 4.4.3 we described the three problem classes, which are identified by the share of the seven smallest box sizes L_s. Each problem class has 100 problem instances. For each problem instance we run the greedy packing heuristic 100 times. That means that we generate 100 different box orders and the packing heuristic processes every order. Hence, the greedy packing heuristic is solved 30,000 times for each parameter setting.

Table 5.8: Results of greedy packing heuristic averaged over 100 instances per bol.com problem class.

	Volume use					Boxes packed				
Algorithm	Greedy packing heuristic				GRASP	Greedy packing heuristic				GRASP
#boxes	1		5			1		5		
#containers	1	3	1	3		1	3	1	3	
Problem class	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
$L_s = 1/3$	67.77	75.13	73.60	77.76	89.22	24.59	28.30	28.82	29.74	29.21
$L_s = 1/2$	67.66	75.38	73.92	78.32	89.97	28.60	33.16	34.12	35.14	34.15
$L_s = 2/3$	66.87	75.32	74.28	79.27	90.69	34.85	40.58	42.88	44.01	42.84
Mean	67.43	75.28	73.93	78.45	89.96	29.35	34.01	35.27	36.30	35.40

Note. 100 box orders processed per instance.

The synthetic data are summarized in Table 5.8. Each problem instance was also solved by the normal GRASP from Section 5.1.3 and results are shown in Table 5.8 as well. As expected, the volume use obtained by the GRASP is much higher than the use obtained by the packing heuristic. It is interesting to note that the number of boxes packed by the packing heuristic (*#boxes*=5, *#containers*=3) is slightly higher than the GRASP. In general, the GRASP is more successful in packing the biggest boxes. Not packing a PA-14 Sky box is equal to a volume use loss of 12% of the container's capacity. In addition, it must be mentioned that the average number of boxes in a problem instance was 38.85. Hence, on average circa three boxes are not packed. The box sizes that are not packed are the cause for the difference in volume use between the packing heuristic and the GRASP.

If bol.com were to adjust the loading process, such that the input is known in advance and all boxes can accumulate before a workman starts packing, the volume use could increase to around 90%. This would translate to a reduction in used containers of more than 10%.

5.3.2 Estimates of linear regression model

In Section 5.3.1 we presented a summary of the synthetic training data that was generated. We considered in total four different hyper parameter settings and 300 problem instances. Hence, in total we generated 1200 observations of Y, i.e. the average number of boxes per container. We use this data to construct a linear regression model where we regress Y on the distribution of box types and two dummy variables, which were introduced in the previous section.

Table 5.9: Linear regression results for the relationship between the number of boxes in a container and the distribution of boxes.

Variable	Estimate	SE	**Variable**	Estimate	SE
$D_{containers}$	-2.84	(0.14)	Outbox	55.39	(1.96)
D_{boxes}	-4.11	(0.14)	Colorbox	48.87	(1.82)
Soap box	72.95	(1.86)	PA-8 Juke box	36.96	(1.98)
Match box	67.66	(2.29)	Pillowbox	32.59	(2.06)
Parfumebox	69.90	(2.06)	Shoebox	13.83	(1.90)
Sound box	72.38	(2.30)	Pandora's box	6.39	(1.93)
Lunch box	69.70	(1.88)	Toolbox	-27.15	(1.81)
Sandbox	61.44	(1.95)	PA-14 Sky box	-72.50	(1.86)
Observations	1200				
R^2	0.91				

Note. The independent variables except for dummies, are values between 0 and 1; for each estimate it holds that $p < 0.001$.

Results of the linear regression model are given in Table 5.9. We see that the values for coefficient estimates are as expected. Packing one container or loading each box instantly upon arrival at the loading dock has a negative effect on the number of boxes packed. We saw this before in Section 5.2 and in Table 5.8. Also, we see a decreasing positive effect per box size. Remember that the Soap box and the PA-14 Sky box are respectively the smallest and biggest box size.

The coefficients do require some extra intuitive interpretation. The two biggest box sizes have a negative effect on the number of boxes packed. If the distribution of box sizes would only consist of the Toolbox and the PA-14 Sky box, we would predict a negative average number of boxes per container. This is of course not a feasible prediction. However, in reality each box size will have a share in total distribution and we will not obtain negative predictions.

With respect to the quality of the regression model, we see that the model has an R^2 of 0.91. Hence, the input variables explain more than 90% of the variance in average number of packed boxes. Also, we note that each independent variable is highly significant. The standard error of regression (SE), also referred to as the root mean squared error ($RMSE$), is 2.36. The standard error is computed by $SE = \sqrt{\frac{\sum_i (Y_i - \hat{Y}_i)^2}{\mathbf{df}}}$, where Y_i is the observed value, $\hat{Y}_i$ the predicted value and **df** the degrees of freedom, calculated as the total number of observations minus total number of model parameters.

Based on a rule of thumb for confidence intervals we can compute the 95% prediction interval for predictions by $\hat{Y} \pm 2 \cdot SE$. We may assume that on average 95% of the predictions deviate less than 4.72 boxes from the actual average number of boxes packed in a container.

5.3.3 Out-of-sample test results

In order to test the performance of the model, we do an out-of-sample test. In Section 4.4.3 we constructed 150 test instances. For each instance the greedy packing heuristic processes 100 different box orders for every hyper parameter setting. We obtain 600 observations of Y_i — corresponding to 600 different input groups X_i — for testing the regression model. These are observed values given by Y in Table 5.10. We also compute predictions $\hat{Y}$ by entering the input variables in the linear model.

Table 5.10: Observations vs. predictions: test results averaged over 50 instances per bol.com problem class.

#boxes	1				5			
#containers	1		3		1		3	
Problem class	$\hat{Y}$	Y	$\hat{Y}$	Y	$\hat{Y}$	Y	$\hat{Y}$	Y
$L_s = 1/3$	24.02	24.68	26.87	28.46	28.13	28.93	30.97	29.85
$L_s = 1/2$	29.51	28.05	32.35	32.53	33.62	33.47	36.46	34.46
$L_s = 2/3$	37.23	35.08	40.07	41.06	41.34	43.64	44.18	44.77

Note. 100 box orders processed per instance.

The average absolute error over all 600 observations equals 1.84. The predictions have a SE of 2.77 boxes and the R^2 equals 0.89. Remember the assumption made in Section 5.3.2, that 95% of the observed values lie in between the interval $\hat{Y} \pm 4.72$. Figure 5.7 shows all errors between the predictions and the observations. All observations that lie outside the 95% confidence interval are colored blue. The proportion of blue lines is 4.83%. Hence, 95.17% of the predictions deviate less than 4.72 boxes from the observations. This is thus in line with the previous made assumption.

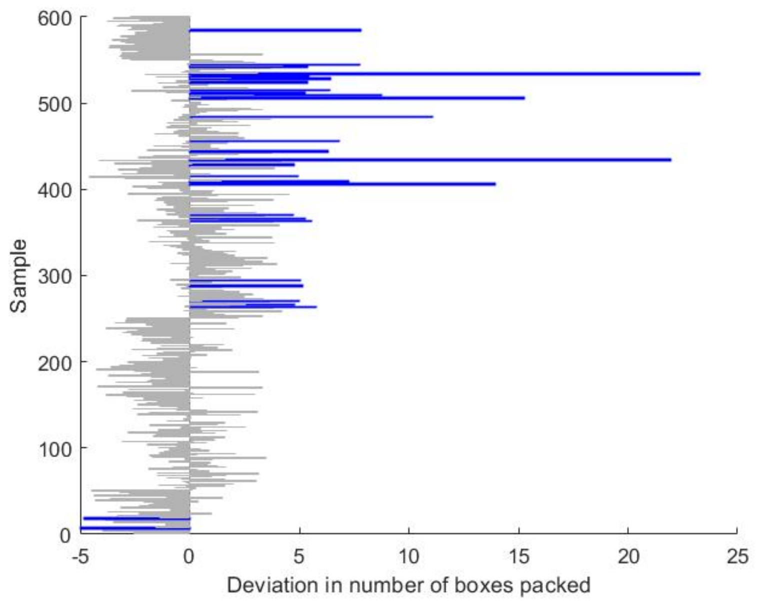


Figure 5.7: Deviations between the predictions and observations: 29 observations that fall outside the 95% prediction interval are colored blue.

It is interesting to note that out of the 29 observations that fell outside the prediction interval, 19 are of the problem class where the share L_s of the smallest boxes was $^2/_3$ (samples 400-600). Also, the magnitude of the deviations are highest for this problem class. This can be explained by the fact that the average number of boxes in a container is much higher for $L_s = {}^2/_3$ as was shown in Table 5.10. There is more room for deviation from the observed value as it increases. It is therefore insightful to not only look at the absolute deviations, but to also examine the relative deviation in percentages. The average relative deviations per problem class are 6.05%, 4.97% and 5.47% for L_s equal to $^1/_3$, $^1/_2$ and $^2/_3$ respectively. The average over all samples equals 5.49%.

Considering only the 571 observations in the prediction interval, we see an average relative deviation of 4.89%. Based on these results, we can conclude that in general the average relative deviation is less than 5% for the predictions that lie in the 95% prediction interval.

6 Conclusion and Future Research

As described in Section 1, the main goal of this research was to develop a model, which accurately predicts the average number of boxes that are packed in a container in a bol.com fulfillment center on a given day for a certain delivery flow.

We first presented a reactive GRASP proposed by Parreño et al. (2008). This algorithm is capable of loading boxes into a container with very high volume use. We suggested an adjustment to the hyper parameter setting and the structure of the reactive GRASP. Based on a comparative study we conclude that the adjusted GRASP outperforms the reactive GRASP of Parreño et al. (2008) for weakly heterogeneous problem instances ranging from three to thirty different box sizes.

Based on the GRASP algorithm we constructed a greedy packing heuristic to solve an online variant of the 3D-CLP. This online variant resembles the loading process at a single loading dock in a bol.com FC. Hyper parameter settings of the greedy packing heuristic can be adjusted, in order to best mimic the loading process at a certain loading dock. Settings consist of (a) the number of containers that are loaded in parallel and (b) the number of boxes that accumulate at the loading dock before the workman starts loading. Also, the greedy elements of the heuristic are such that they best resemble the loading logic of a workman.

To obtain insight into the behaviour and performance of the packing heuristic, we did an extensive sensitivity analysis. Among several interesting insights, one result must be mentioned. With respect to the number of boxes that accumulates in each loading iteration, we saw that the relative increase in volume use was highest when increasing *#boxes* from one to five. The relative increase from five to ten was nearly negligible.

Finally, the greedy packing heuristic was used to generate synthetic training and test data. We trained and tested a linear regression model for predicting the average number of boxes in a container. Input consisted of (a) the number of containers that are loaded in parallel, (b) the number of boxes that accumulates in each loading iteration and (c) the distribution of the different box sizes. Based on the test results we concluded that 95% of the predictions deviate less than 4.72 boxes from the actual average number of boxes packed. For the same 95% we see that the average relative deviation is less than 5%. Hence, we constructed a very accurate model, which could be really useful for planning daily operational transport at the bol.com fulfillment centers.

More (field) research could be done with respect to the loading tactics of workman. Several assumptions were made and these could be tested and adjusted if needed. This would result in an improved representation of reality. Another focus area is to get insight in the distribution of different box sizes. If distributions are drawn up, the linear model can really be used in practice. Lastly, it is interesting to investigate if total volume use can be increased if our proposed GRASP incorporates parallel loading.

References

Bischoff, E. E., & Ratcliff, M. (1995). Issues in the development of approaches to container loading. *Omega*, *23*(4), 377–390.

Bortfeldt, A., & Gehring, H. (2001). A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, *131*(1), 143–161.

Bortfeldt, A., & Wäscher, G. (2012). Container loading problems: A state-of-the-art review. *Working Paper Series*.

Davies, A. P., & Bischoff, E. E. (1999). Weight distribution considerations in container loading. *European Journal of Operational Research*, *114*(3), 509–527.

Delorme, X., Gandibleux, X., & Rodriguez, J. (2004). GRASP for set packing problems. *European Journal of Operational Research*, *153*(3), 564–580.

Fanslau, T., & Bortfeldt, A. (2010). A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, *22*(2), 222–235.

Gajda, M., Trivella, A., Mansini, R., & Pisinger, D. (2022). An optimization approach for a complex real-life container loading problem. *Omega*, *107*, 102559.

Gehring, H., & Bortfeldt, A. (1997). A genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, *4*(5-6), 401–418.

Gehring, H., & Bortfeldt, A. (2002). A parallel genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, *9*(4), 497–511.

George, J. A., & Robinson, D. F. (1980). A heuristic for packing boxes into a container. *Computers & Operations Research*, *7*(3), 147–156.

Gonçalves, J. F., & Resende, M. G. (2012). A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, *39*(2), 179–190.

Junqueira, L., Morabito, R., & Yamashita, D. S. (2012). Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations Research*, *39*(1), 74–85.

Martello, S., Pisinger, D., & Vigo, D. (2000). The three-dimensional bin packing problem. *Operations Research*, *48*(2), 256–267.

Moura, A., & Oliveira, J. F. (2005). A GRASP approach to the container-loading problem. *IEEE Intelligent Systems*, *20*(4), 50–57.

Park, S. K., & Miller, K. W. (1988). Random number generators: good ones are hard to find. *Communications of the ACM*, *31*(10), 1192–1201.

Parreño, F., Alvarez-Valdés, R., Oliveira, J. F., & Tamarit, J. M. (2010). Neighborhood structures for the container loading problem: a VNS implementation. *Journal of Heuristics*, *16*(1), 1–22.

Parreño, F., Alvarez-Valdés, R., Tamarit, J. M., & Oliveira, J. F. (2008). A maximal-space algorithm for the container loading problem. *INFORMS Journal on Computing*, *20*(3), 412–422.

Prais, M., & Ribeiro, C. C. (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, *12*(3), 164–176.

Silva, E. F., Toffolo, T. A. M., & Wauters, T. (2019). Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. *Computers & Operations Research*, *109*, 12–27.