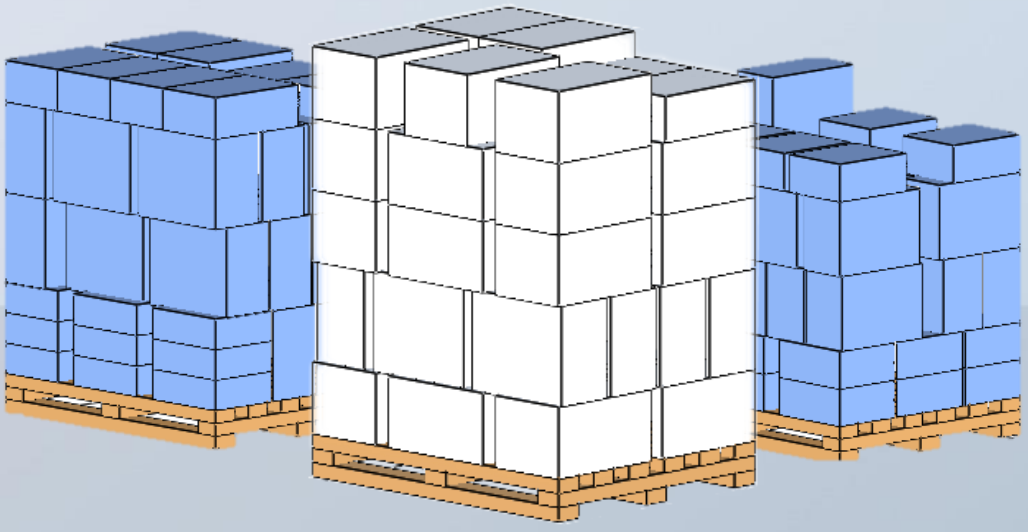
# A LAYER-BASED APPROACH FOR THE PALLET LOADING PROBLEM USING A GENETIC ALGORITHM

ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics
Master Thesis

Analytics and Operations Research in Logistics
Supervisor: Wilco van de Heuvel
Second assessor: Rick Willemsen



**NATHAN STERK**
624054
March 2, 2023

# Abstract

This thesis proposes a solution approach to the Pallet Loading Problem for the beverage industry, with the aim of creating a feasible loading plan while minimizing total handling time. Various constraints are met such as a volume, stability, stackability, (multi-)customer demand and a load-bearing strength constraint. Data is provided by Coca-Cola and two truck-types are considered: Bay Trucks and Standard Trucks. We present a threefold procedure in which horizontal layers are created, allowing items to be stacked vertically within a layer. Subsequently, layers are palletized after which a genetic algorithm combines and improves different solutions. The results are promising for Standard Trucks, in which optimal solutions are found within typically one-tenth of a second. However, the algorithm does not perform as good for nearly full Standard Trucks, as it is not always able to find a feasible solution in these cases. Regarding Bay Trucks, the layer-based approach succeeds in finding feasible solutions within a minute, regardless of the fill rate. However, the presented layer-based algorithm results in a considerable handling time for Bay Trucks, indicating that Bay Trucks are less suitable for a layer-based approach.

# Contents

# 1 Introduction

The transportation of goods is an important aspect within several industries. Businesses are keen to distribute their goods efficiently to reduce their total expenditures. Freight transportation can be divided into three categories that are well-known in the today's current literature. Namely the lot-sizing problem Billington et al. (1986); Karimi et al. (2003); Glock et al. (2014) the vehicle routing problem Toth and Vigo (2002); Baker and Ayechew (2003); Montoya-Torres et al. (2015) and the Container Loading Problem Chen et al. (1995) Bortfeldt and Wäscher (2012); Bischoff et al. (1995).

In this thesis we consider the Container Loading Problem (CLP), specifically for the beverage industry. The CLP seeks to find a feasible loading plan in which items are packed inside a container, in this case a truck. These items will be delivered to one or more customers, in a predetermined order. At every stop, items are unloaded from the truck. This research focuses on creating a feasible loading plan, while minimizing the total handling time. The handling time is the delay that is taken into account when an item has to be moved in order to reach the item that needs to be unloaded. The data used in this thesis is provided by Coca-Cola, the third largest beverage company by 2021 M.Ridder (2021). The data is gathered in Japan and concerns real-life cases. This thesis considers two different truck-types. Namely Standard Trucks, which are unloaded from the back, and Bay Trucks, which consists of compartments that are accessible through the sides of the truck.

There are multiple conventional constraints regarding the CLP, which are summarized in Bortfeldt and Wäscher (2013). First, there is a volume restriction for the pallets, as well as for the trucks. Second, stability constraints make sure that the items are safely stacked on each other. Third, all items have an individual load bearing strength, which accounts for the maximum weight that can be stacked onto these items. Fourth, weight distribution constraints need to be met inside the truck to make sure that the truck is stable. Fifth, stackability constraints do not allow certain items to be stacked on each other. Finally, we introduce orientation constraints which ensure pallets and items are not on their side. Crucial is that these constraints need to hold after every stop where the truck is (partially) unloaded.

The Pallet Loading Problem (PLP) is a specific case of the CLP. In the PLP, products first need to be packed on pallets, after which the pallets are assigned to a location inside a truck. This thesis focuses specifically on the PLP, since pallet packing is mainly used in the beverage industry. The beverage industry encounters some specific constraints that are not showcased explicitly in current research. The orientation and stackability constraints, together with the fact that the items need to be stacked on relatively small pallets - instead of using the full loading size of a truck - are very typical constraints for this industry. Together with the fact that more than one customer may be served with the use of one truck makes the loading process more complex. These specialities require a different approach than algorithms that focus on the 'standard' CLP, such as Bischoff et al. (1995); Zhao et al. (2016).

The beverage industry faces unique challenges in pallet loading, as the main concern is optimizing the unloading plan by determining the placement of a tray of cans. Despite the varying shapes of cans and bottles, there is typically a limited variety of tray shapes, providing an opportunity to improve existing heuristics. This thesis builds upon the work of Júnior et al. (2019), who address the issues that current algorithm encounter when it comes to packing for the beverage industry. In their work, horizontal layers of items of homogeneous height are formed and stacked, with 'incomplete layers' consisting of items of varying height placed on top of the stacked layers. First arranging items to layers and subsequently to the truck simplifies the problem by dividing it into two sub-problems, reducing the size of the problem and potentially decreasing solving time. However, this approach is only effective when item height variations are limited. To address this limitation, this thesis combines the forming of layers with the approach of Elhedhli et al. (2019) to allow two small items to be stacked on top of each other - called a superitem - within a layer, resulting in higher quality solutions.

The research question of this thesis is: To what extend is a layer-based approach able to solve the PLP? To answer this question, an algorithm is created combining different techniques of known literature, together with newly designed aspects. The approach used in this thesis is threefold. Firstly,

layers are made - possibly containing superitems - with the use of a greedy random adaptive search procedure (GRASP). Secondly, the formed layers are placed onto pallets. The rest items, the items that remain as left-overs in the first step, are subsequently placed on top of the pallets. This results in a single loading plan: a solution is obtained. Lastly, to improve the solution quality, various different loading plans are created by iterating over step 1 and 2. These solutions are then combined to form new solution with the use of a second genetic algorithm.

To main research question - to what extend is a layer-based approach able to solve the PLP - is divided into 4 subquestions. These four subquestions provide insight into the performance of the Layer-based Algorithm, showing its behaviour under different circumstances. The subquestions are formulated as follows:

1. How does the 'Palletizing Algorithm' perform compared to the MIP approach?
2. What is the influence of different truck types on the performance the Layer-based Algorithm?
3. How does the Layer-based Algorithm scale depending on the fill rate of trucks?
4. To what extend is the Layer-based Algorithm able to construct feasible solutions, regarding load bearing strength, stability and stackability?

This thesis answers the main research question through the above formulated subquestions. To provide extensive and well-substantiated answers to these questions, the following chapters are covered in the thesis. Firstly, the current literature regarding Pallet Loading Problems is reviewed in section 2. Secondly, a problem description is provided in section 3 which all aspects of an instance are showcased, together with an overview of the made assumptions. Thirdly, the data section 4 shows how the specific data of Coca-Cola is structured. Fourthly, in the section 5 the methodology of the Layer-based Algorithm, the two genetic algorithms, the rest-item algorithm and the formulation of the MIP are described in depth. Fifthly, the result section 6 gives an overview of the outcomes regarding the research question, together with the three subquestions. Finally, the results are analysed and interpreted in section **??**, after which follow-up research is advised.

# 2 Literature Review

The Container Loading Problem consists of placing multiple items inside a container, satisfying a specific set of packing constraints. The first study about this problem is done by Bischoff et al. (1995), forming a feasible loading plan where all items are required to be 'stable'. This paper forms the basis of most current CLP literature. Within this area of research, many different variations and approaches are investigated subsequent to this approach. Different variations lie in the use of the constraints. The most basic variant of the CLP only considers volume constraints - in which items do not exceed the containers volume - and constraints disallowing items to be stacked inside each other. More general constraints are weight distribution, stability, and orientation of the items. A brief overview of these constraints, together with an explanation about the challenges per constraints is given in Bortfeldt and Wäscher (2013). This research also takes stackability constraints into account, which is mentioned in Bódis (2015); Alonso et al. (2016).

Another important aspect of the data used in this thesis is that there are multiple customers, who require the items to be dropped-off at different locations. Therefore the order in which items are stored plays an important role. This problem is referred to as the multi-drop Container Loading Problem. Junqueira et al. (2012) uses a Mixed Integer Program (MIP) to solve the multi-drop CLP, considering only basic constraints for volume and stability. Christensen and Rousøe (2009) additionally includes load bearing strength constraints and proposes a heuristic approach for the multi-drop CLP. Ceschia and Schaerf (2013) includes the same constrains and uses a local search metaheuristic.

Regarding truck types, there is many research available for the CLP for Standard Trucks, which can be unloaded from the back of the truck. Only Júnior et al. (2019) mentions the specific characteristics of Bay Trucks, which contain multiple compartments that can be reached through the side of the truck. The packing of Bay Trucks can be seen as a Multi-Container Packing Problem, where the compartments for the Bay Truck are considered as containers. Ceschia and Schaerf (2013) solves the CLP with multiple trucks using a local search metaheuristic. Alonso et al. (2019) proposes an IP approach for the multi-truck CLP, only cases with one customer. To solve for Bay Trucks, one can also solve the PLP, including the fact that all pallets are reachable. Research which makes use of the principle of small compartments inside a Bay Truck are Sciomachen and Tanfani (2003); Pacino and Jensen (2012); Araujo et al. (2016). However, these papers are not about trucks, but about ship loading. Therefore many constraints used for the CLP are not taken into account, such as load bearing strength and stackability. There is also no maximum height for these ship containers. In conclusion, current literature about packing Bay Trucks is limited, and can be extended by solving the multi-drop PLP on the condition that all pallets are reachable at a stop.

The CLP has been proven to be NP-hard (Silvano Martello and Vigo (2000)). Therefore large instances are not solvable in reasonable time. Martello et al. (2000); Hifi (2004) solve for optimality using an exact model, both applying branch and cut. These papers only include volume constraints, neglecting stability and stackability constraints.

When considering heuristics, a distinction can be made between constructive, tree-search and meta heuristics. A constructive heuristic is used by Bischoff et al. (1995), building pallets from the bottom upward using layers. Araújo and Armentano (2007) proposes a constructive heuristic in which items are placed one-by-one on a pallet choosing the next item probabilistically. Metaheuristics are widely studied for the CLP. Kang et al. (2012) constructively stacks items in a bin and improves the solution with the use of a genetic algorithm to maximize the total amount of items packed in a bin. Bortfeldt and Gehring (2001) uses a genetic algorithm differently by creating vertical layers of items. A tabu search is performed by Bortfeldt et al. (2003) using a distributed-parallel approach including stability constraints. Liu et al. (2011) also takes the load bearing strength into account, maximizing the total amount of used space in a bin. Another promising metaheuristic is simulated annealing. Wang et al. (2012) use a multi-stage search based simulated annealing algorithm to maximize the total volume utilization of a single container. Mostaghimi Ghomi et al. (2017) includes stability constraints, and combines the algorithm with a MILP.

The CLP can also be seen as a Pallet Loading Problem (PLP). Instead of a widely spaced truck, the available placement area is reduced to the size of a pallet. Pallets are subsequently placed inside the truck. Although CLP and PLP share a lot of similarities, some algorithms perform better at CLP than PLP. An example is Scheithauer and Terno (1996), who uses the concept of layering in which items are first assigned to a layer of items with the same size, before being placed on a pallet. Júnior et al. (2019) extend this approach by also allowing the top layer to be 'incomplete', containing items of different height. Incomplete layers are formed with the principle of order stacking of Egeblad et al. (2010).

The use of layers, which is used in this thesis, is firstly studied by Scheithauer and Sommerweiß (1995). Here the two dimensional rectangle problem is solved to obtain layers. Dividing the PLP in 2 sub-problems, namely creating layers and assigning layers to pallets, reduces the complexity of the problem. In this thesis the creation of layers is combined with a genetic algorithm, as in Bortfeldt and Gehring (2001). Instead of creating vertical layers, the algorithm proposed in this thesis uses horizontal layers as its input, as inspired by Júnior et al. (2019). Besides, this thesis accounts for cases including multiple customers instead of a single customer. Also, load bearing constraints for individual items are taken into account, using the approach proposed by Elhedhli et al. (2019). The combination of these approaches suit the data of Coca-Cola used in this research, taken all important constraints regarding the beverage industry into account. How these approaches are implemented in practice is described in more detail in the methodology section.

# 3 Problem Description

This thesis solves the multi-drop PLP, which involves the transportation of items belonging to multiple customers on a single truck. As a result, the unloading of the truck must be done in phases, with the customers served one by one. To precisely define the PLP, the following notation is used. Let $T = \{t_1, t_2\}$ denote the set of available truck types, each with its own capacity and specific characteristics. The information about pallets is represented by quadruple $(m, W, D, H)$ consisting of the number of pallets $m$, width $W$, depth $D$ and maximum height $H$, which does not include the height of the pallet itself. Let $I = \{1, ..., n\}$ be the set of items that need to be loaded onto the pallets. Each item $i \in I$ has a specific width, depth, height, weight, load bearing strength and customer, denoted by $w_i$, $d_i$, $h_i$, $g_i$, $b_i$ and $c_i$, respectively. Note that $c_i$ represents the customer and therefore the order in which items are unloaded, where $c_i$ is unloaded before $c_j$ in case $i < j$. Furthermore, the set of items contains a stackability matrix $M = [a_{i,j}]_{n \times n}$, indicating whether item $i$ can be placed on top of item $j$ with $i, j \in I$ and $a_{i,j} \in \{0, 1\}$. The algorithm proposed in this thesis makes use of layers $l \in L$ with height $h_l$ such that $h_l = h_i \ \ \forall \ i \in l$.

Typically, the objective of the PLP is to pack all items in a feasible manner while minimizing the number of pallets used. However, in the dataset used for this thesis, feasibility is guaranteed and the number of pallets that can fit in a truck is fixed. Consequently, the number of pallets used is not a relevant factor in this particular PLP. Therefore, this thesis minimizes the total handling time, which is represented by the time $t$. The handling time consists of a fixed penalty $CP$ for opening truck compartments, and a variable relocation penalty $p_i$. This penalty is incurred when item $i$ has to be moved during unloading to reach item $j$, with $c_i < c_j$, and is defined as $p_i = \alpha + w_i \cdot \beta$. Where parameters $\alpha$ and $\beta$ are predetermined and represent the fixed penalty per relocated item and the weight-dependent penalty, respectively.

In this thesis, two types of trucks are considered. Namely, $t_1$: a Bay Truck, and $t_2$: a Standard Truck. For Bay Trucks, pallets are placed in compartments, which are all accessible through the side of the truck, while for Standard Trucks pallets can be only reached from the back of the truck. For both trucks, a relocation penalty $p_i$ is charged for moving item $i$ during unloading, while the compartment penalty $CP$ only applies to Bay Trucks. The compartment penalty is calculated for each compartment to be opened, which may involve multiple compartments per stop. Despite the fact that pallets have to be unloaded from the back of the truck in the case of a Standard Truck, shown in Figure **??**, it is still possible to access every pallet during unloading. However, if the pallets are located at the back of the truck, it may be necessary to first remove other pallets that are in front, resulting in a relocation penalty $p_i$ for each item $i$ on the pallet. In order to solve the PLP, there are several constraints that must be obeyed. These constraints are necessary to ensure feasibility of the final loading plan:

*Volume constraint:* The total volume of the items on a pallet cannot exceed its accessible volume. That is, $\sum V_i \leq WDH$, for all items $i$ on the same pallet, with $V_i = w_i d_i h_i$ representing the volume of item $i$. In addition, items may not overlap with each other, and also remain within the dimensions of the pallet.

*Customer demand constraint:* All items need to be part of the loading plan exactly once. Only one truck is used per instance, and all items have to be part of the loading plan of this truck.

*Stability constraint:* All items need to be stable on a pallet. This implies that items require sufficient support when stacked.

*Load bearing strength constraint:* All items have a maximum weight which they can hold, $b_i$. The sum weights $w_j$ of items $j$ resting on top of item $i$ should be lower or equal than $b_i$.

*Stackability constraint:* Some items can not be placed on certain other items. Stackability matrix $M_{ij}$ represents which items can be stacked on top of other. If items $i$ and $j$ are not stackable, then any overlap of the two items results in an infeasible loading plan.

To satisfy all constraints of the PLP, assumptions need to be made. These assumptions are necessary to ensure that the problem can be mathematically modeled and solved. This thesis applies the following assumptions:

*Item Stability assumption:* In order to meet the stability constraint, all item need to be stably placed. We assume that an item is stable if at least 70% of its lower surface is supported. Additionally, an item is considered stable when all four of its corners are supported. Finally, we assume that when layers have a fill rate of 70%, the stability constraints of each item can be satisfied, as also used by Gzara et al. (2020).

*Load bearing strength assumption:* We assume that the weight of an item is distributed evenly on the supporting items based on the fraction of support.

*Stackability assumption:* In this thesis, we present a method to account for stackability, as described in section 5.5.3. We assume that 2 layers will not results in stackability issues in case $\alpha \leq 0.4$, where $\alpha$ gives an indication of the amount of non-stackable items in those layers, as described in detail in section 5.5.3.

*Accessibility assumption:* When unloading Standard Trucks, the assumption is made that filled pallets need to stay side-by-side at any moment during unloading, due to stability. Besides, we assume that pallets are not stable in case one side of the truck is unloaded, while the other side is still filled. We illustrate this using Figure 1. For Standard Trucks, pallets 1 and 2 are accessible at the start of unloading. In case pallet 1 if emptied, we are still allowed to unload items from pallet 3, without a penalization being involved. However, when pallet 3 is fully unloaded and pallet 2 still contains items, we assume that we are not able to freely reach pallet 5, without violating stability constraints. In this case, a relocation penalty $p_i$ will be encountered for each item $i$ on pallet 2.

*Relocating assumption:* For unloading item $i$, we assume that all items $j$ placed above item $i$ have to be moved in order to reach it, resulting in a relocation penalty $p_j$. In this case, item $j$ does not have to be directly above item $j$, being at the other side of the pallet, but above item $i$, still results in a penalty. Furthermore, we assume that after an item is relocated, it is placed back on its original pallet, without concerning stability or stackability constraints.

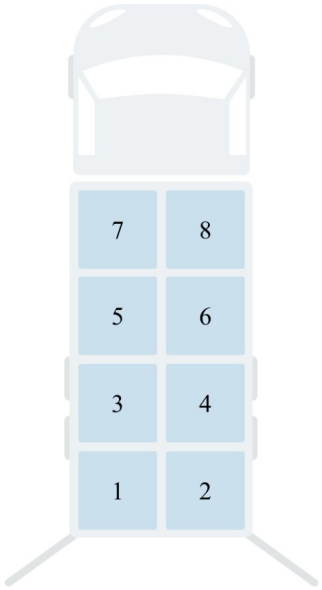



Figure 1: Visualisation of a Standard Truck. The pallets, which are numerated, can only be reached from the back of the truck.

As mentioned, the PLP is solved by minimizing over the total handling time $t$. We can confidently say that a loading plan is optimal when the total handling time is equal to 0. However, a solution having $t > 0$ does not necessarily mean that the optimal solution is not yet reached. An example of this is shown in Figure 2. In this simplified version of the PLP, a penalty has to be encountered to ensure feasibility. Therefore, we can not state if an solution is not yet optimal in case $t > 0$.

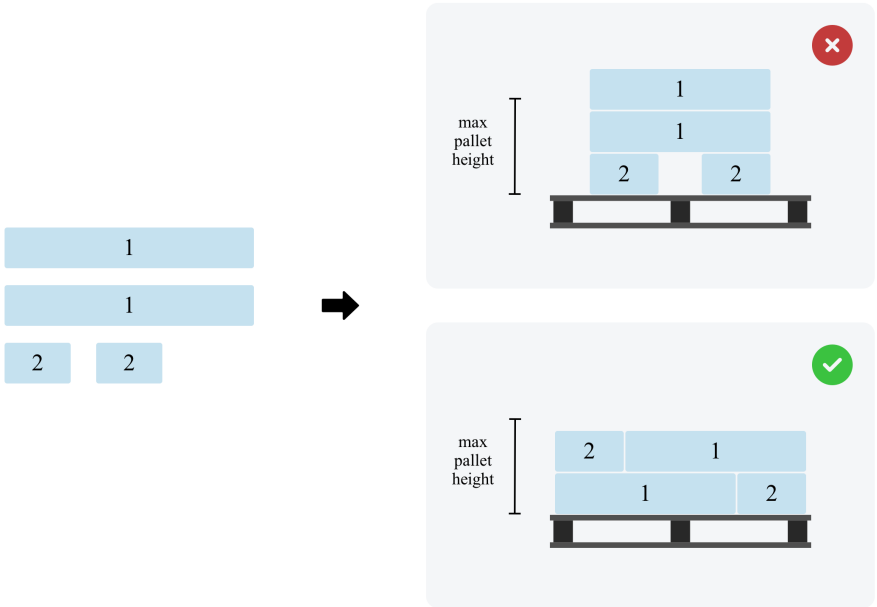



Figure 2: Items are labeled by customer. In this example, a relocation penalty for the item of customer 2 is inevitable in order to reach feasibility.

# 4 Data

The data that is used is gathered in Japan by the beverage company Coca-Cola, one of the biggest competitors in its working field in the world. The total data set exists of over 100,000 different real-life instances. Every instance contains information about the truck, pallets, items and customers. For each instance, it is guaranteed that the items and pallets can be stacked inside the available trucks to form a feasible solution. In this section we evaluate the data, with the aim of finding out the structure regarding the different truck types. Having a clear understanding of the data allows us to choose a suitable approach to solve for the PLP.

When observing the data, one can notice a substantial difference between the instances for Standard Trucks and Bay Trucks. First of all, Bay Trucks are considered in 74.9% of the instances, while Standard Trucks are only used in 25.1%. Furthermore, both trucks differ in the composition of their items. Figure 3 shows the number of customers per instance for both truck types. From the more than 100.000 instance we can determine the percentage of cases in which a certain amount of customers is present. The results show that in 93% of the cases for Standard Trucks there is only one customer, while for Bay Trucks this value is 47%, representing a much wider variation on the number of customers. Having only one customer for Standard Truck implies that in the vast majority cases the handling time is not taken into account. Namely, when there is only one customer to be served, items will never be given a relocation penalty. This means that optimality is already guaranteed when reaching feasibility. On the contrary, this does not hold for Bay Trucks. For a Bay Truck delivery with only one customer we still need to pay attention to the location of the items, because compartment penalties are involved. Taking this into account, together with the fact that Bay Trucks contain a higher amount of customers on average, we state that loading pallets is more complex for Bay Trucks than for Standard Trucks.

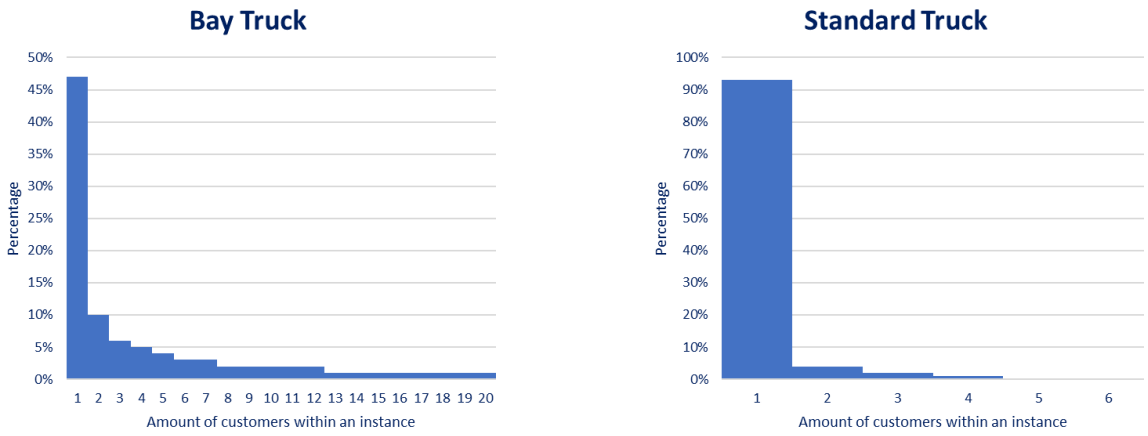



Figure 3: Percentage of instances in which the amount of customers is equal to a certain value, for Bay Trucks (left) and Standard Trucks (right).

To get a better understanding of the data of both truck types, we take a closer look at the items that need to be packed. Note that all items can be rotated sideways. However, just a small proportion of items can be rotated upwards, i.e. put on their side. This is the case for 1.8% of all the items. For Standard Trucks only 0.04% of items can be put on their side, for Bay Trucks this is 6.0%. Moreover, it is useful to identify the amount of different item types per truck type. An item type represents an items' customer and its measurements. Two identical item types therefore belong to the same customer, while also sharing the exact same dimensions. Having a small variety of item types usually lowers the complexity of the problem, since there is less variability in the packing possibilities. Furthermore, a layer-based approach, as used in this thesis, benefits from a low amount of item types, since it makes it more likely that layers can be formed. Therefore, an overview of the number of different item types can provide an insight in the both complexity as well as in the suitability for a

layer-based approach. The composition of item types for both trucks are shown in Figure 4. What is noteworthy is the amount of variation in the data for Bay Trucks, while in nearly half of the instances of Standard Trucks at most 6 different item types are involved. From this we state that Standard Trucks are likely to be suitable for a layer-based approach. Only in 4% of the instances there are more than 35 different item types included. For Bay Trucks, it seems more of a challenge to create layers. However, observing just the number of different item types does not give us an inclusive insight in the possibility to create layers. For that, it is important to look at one more component.

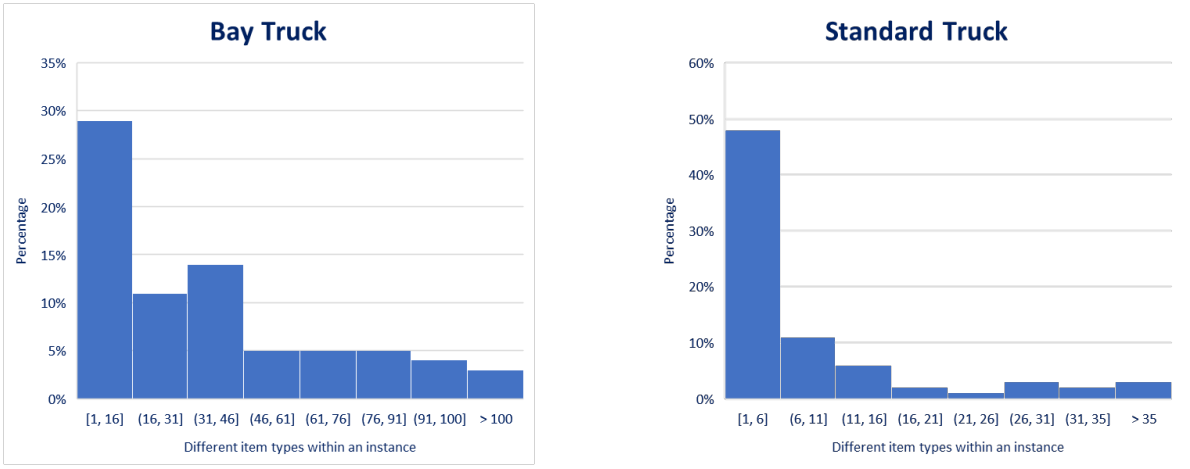



Figure 4: Percentage of instances in which the number of different item types is within a certain interval, for Bay Trucks (left) and Standard Trucks (right).

A requirement for a layer-based approach is the presence of items of the same height. A high variety in item heights makes it likely that only a small proportion of layers can be made, which may cause infeasiblity. To calculate the variety in item heights, we divide the heights in groups of 10 cm, with group 1 containing 0-10cm, group 2 containing 10-20cm etc. Then, for each instance, we check how many different height groups are present. The results show that a different amount of heights of 5 is the most common for Bay Trucks, with a percentage of 17%. Besides, in almost 90% of the instances there are not more than 10 different height groups present. For Standard Trucks almost all instances have at most 10 different item heights, from which a large portion of 25% only has one item height. These results are promising for a layer-approach since it is plausible that the low variety of item heights makes it possible to create sufficient layers so that a feasible loading plan can be obtained.

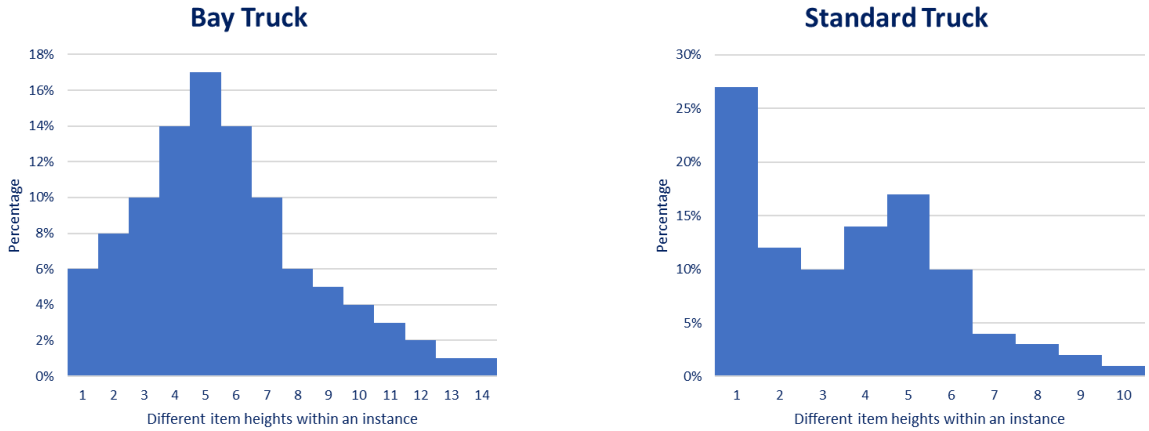



Figure 5: Percentage of instances in which the number of different item heights is a certain amount, for Bay Trucks (left) and Standard Trucks (right)

We have now obtained a comprehensive understanding of the composition of items for both truck types. In the PLP, items are first loaded onto pallets prior to being placed into the truck. It is therefore crucial to also examine the different pallet variations and their dimensions. While in one truck only one pallet variation is used, different instances may have different pallet sizes, depending on the size of the truck. Figure 6 shows the composition of pallets per truck type. The pie chart demonstrates that there are 2 distinct pallet types for Bay Trucks, of which the pallet with dimensions 0.9 x 1.1 x 1.4 is the most prevalent, constituting for 80.3% of the instances, with the dimensions given in meters. For standard Trucks 4 different pallet variations are present, from which pallets with dimensions dimensions 0.9 x 1.1 x 1.4 is the most common with a occurrence of 86.7%.

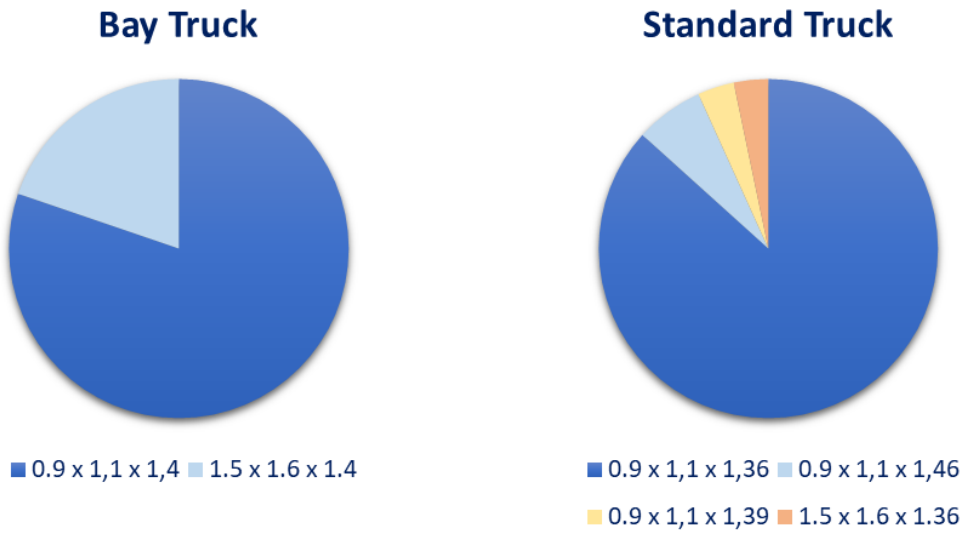



Figure 6: Pie charts of the proportion of the different pallet dimensions. Per instance, only one type of pallet is used. The measurements (L x W x H) are given in meters.

To get a better understanding of the size of the instances, it is also useful to examine the size of the truck. This is accomplished by determining the number of pallets that can be placed in the truck. Figure 7 provides an overview of the number of pallets per truck type. The results indicate that Standard Trucks contain a significantly higher number of pallets compared to Bay Trucks. Regarding Bay Trucks, we notice that the amount of pallets is always 10. For Standard Trucks, 67% of the trucks contain 20 pallets, making it the most prevalent truck. The variety in the number of pallets for a Standard Truck is caused by the fact that there exists a truck in which 2 different pallet sizes fit. Note that per truck only 1 pallet size is used.

Finally, we consider two aspects that influence the feasibility of a solution. These are the load bearing strength and the stackability of the items. The items that need to be loaded are mainly cans and bottles. These items have the characteristic of having a high load bearing strength. Only in very rare scenarios it may occur that the load bearing strength of one of the items in the truck is exceeded. Conservative tests turned out that for less than 1% of the items load bearing could play an issue, which is very unlikely to result in infeasibility in practice. However, due to the sake of completeness, load bearing strength is taken into account in this thesis. Regarding stackability, we calculated the percentage of arrangements in which stacking two items is not feasible due to stackability. This is done by examining all combinations of 2 items and, for each combination, considering the 2 possibilities for stacking the items on top of each other. This results $2n$ different stacking arrangements. Subsequently, for each stacking arrangement of 2 items, it is checked whether it is possible to stack the items in this manner. For Bay Trucks, 8.3% of all arrangements are not stackable, while this is 3.4% for Standard Trucks. It should be noted that two items of the same category, e.g. two trays of cans or two packages of bottles,

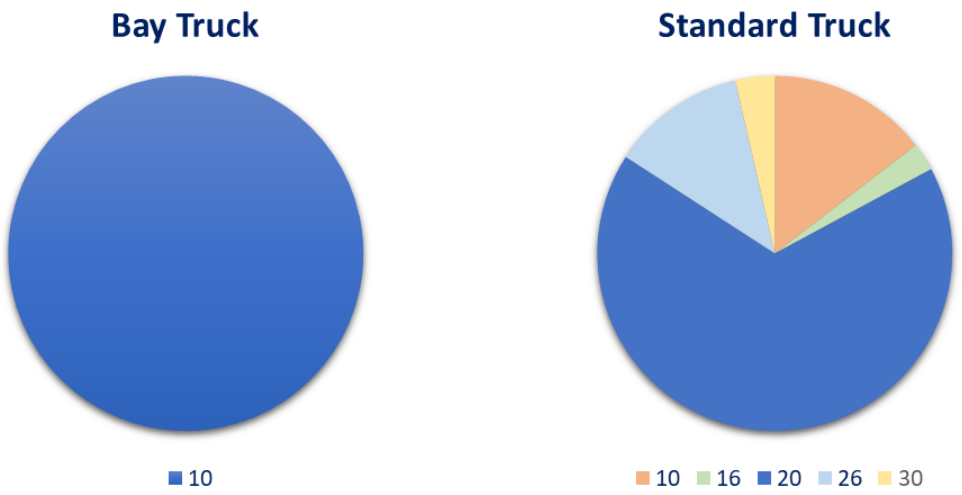



Figure 7: Pie chart of the number of pallets that fit within a truck. For Bay Trucks, pallets are placed within compartments, of which there are always 10.

can always be stacked on top of each other. This partly explains why the value for Standard Trucks is lower than that for Bay Trucks, since Standard Trucks more often contain items of the same category.

One of the four sub-questions is: How does the Layer-based Algorithm scale depending on the fill rate of trucks? To distinguish between different fill rates, instances are subdivided. For Bay Trucks, we consider fill rates of 15%, 35%, and 55%, with the latter being a truck that is nearly 'full'. The fill rate is the total amount of volume of all items, compared to the available volume inside the truck. For standard Trucks, we examine the same percentages, and additionally add a 75% fill rate. Since Standard Trucks primarily consist of identical items, they can be packed more efficiently, making it possible to consider this higher fill rate as well. All instances are carefully selected and provide a representative view of the entire dataset.

# 5 Methodology

When choosing for a suitable method it is important to note that the Pallet Loading Problem requires a different approach than the general 3D Bin Packing Problem (3D-BPP). Loading items on compact pallets instead of loading items directly in the container is more likely to violate the stability constraints. The load on pallets need to be stable, independent of the load plan of other pallets. Where the 3D-BPP can acquire stability by the support of all adjacent items, the PLP, by definition, only allows items to be supported by fellow items on that pallet.

Since stability is a crucial aspect of the PLP, it would be convenient to place two items sharing the same height next to each other on a pallet, which makes it possible to stably place a new item on top. This principle is used by creating horizontal layers: a group of items with homogeneous height that fit within the length and width of a pallet. By creating these layers, and subsequently stacking these layers on top of each other, the size of the problem is reduces significantly. This principle was used to develop the Layer-based Algorithm. In this chapter, we explain the steps that the Layer-based Algorithm performs, the aspects that are taken into account and how its features can contribute in generating a good loading plan.

The Layer-based Algorithm consists of 5 different parts, represented by Figure 8, which are elaborated in 5 subsections of this chapter. At first, section 5.1 gives an overview of the pre-processing steps of the data. Section 5.2 describes the Layer-creating Algorithm, which consists of an iterative process of creating layers, where each iteration has, among other things, a different 'superitem' composition, which is explained in 5.1.2. For each created set of layers, the Palletizing Algorithm is used to place the layers and rest items onto pallets, as explained in section 5.3. In the Palletizing Algorithm, a distinction is made between Standard Trucks (5.3.2) and Bay Trucks (5.3.3) . For Standard Trucks, we present a heuristic, while for Bay Trucks, we present both a GRASP algorithm combined with Genetic Algorithm 1 (5.3.5) and an MIP approach (5.3.6) to place layers on pallets. Subsection 5.4 describes the process of combining the found solutions, which is done by Genetic Algorithm 2. Finally, three methods are described in section 5.5 which are able to showdown different feasibility aspects of the found solution. All steps combined form a single loading plan.

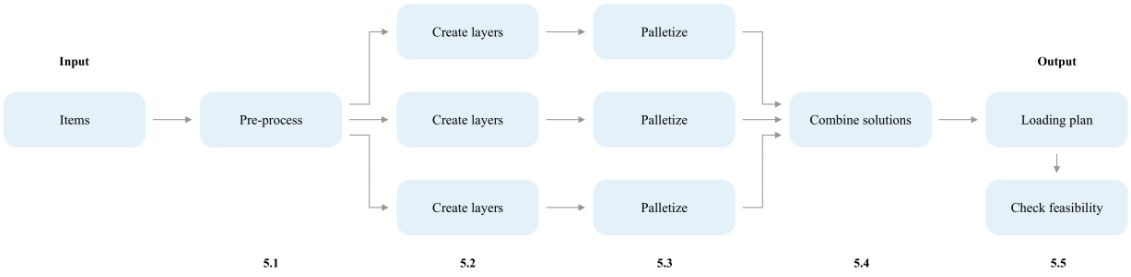



Figure 8: Representation of the proposed methodology

## 5.1 Pre-process

Pre-processing is critical step of the layer-based algorithm, as it ensures that the data is ready to be used by the Layer-Based algorithm. Besides, pre-processing the data contributes to efficiency of the layer-based algorithm.

The first step of pre-processing the data is to create a set of merged items: $M = \{i \in I\ , \exists j \in I \mid w_i = w_j,\ d_i = d_j,\ h_i = h_j,\ c_i = c_j,\ i > j\}$. We then subtract $M$ from $I$ to form a new set of items. Note that the amount of items does not change, since the item amounts of the merged items $j$ are added to $i$. In conclusion, all items sharing the same dimensions and customer are merged, and considered to be identical. For example, a tray of Coca-Cola cans is considered to be identical to a tray of Fanta cans, in case both items belong to the same customer. In this way, the total amount of different item types is reduced, which reduces the complexity of the instance.

After adding identical items together, all different orientations are added per item type. Each item can be horizontally rotated, but only some items are allowed to be rotated vertically. During pre-processing, horizontal rotations are not considered. The possibility to rotate an item horizontally is considered during the creation of layers, since this does not affect the height of an item. However, a vertical rotation affects the height of an item, which plays an important when creating layers. Therefore, all items that can be rotated vertically are added twice to the item pool, once for each orientation. At the stage of creating layers, when the item is placed inside a layer, not only the amount of items left to pack of that orientation is updated, but also that of the vertically rotated counterpart.

### 5.1.1 Cartonize

An important step of the pre-process is cartonizing. Cartonizing, also known as case packing or case filling, is the process of arranging and packaging items inside a box. Some instances contain very small items, e.g. packages of tea. Those items may cause stability issues when packing them individually on a pallet. In order to stably place those items, cartons are available in which the small items can be packed. During this pre-processing step all small items, that have at least two dimension below 20 cm, are considered to be placed inside cartons. The available carton sizes are given for every instance.

When cartonizing, the goal is to pack the items as efficiently as possible. Therefore, we try for each item type two different orientations inside every carton. Either all items are placed inside their original rotations, or all items are rotated by swapping their x and y dimensions. The combination of the carton and item orientation with the least empty space is chosen. It is not possible for item of different customers to be in the same carton due to inefficiency during unloading.

### 5.1.2 Superitems

Items inside a layer share the same height by definition. This means that it is only possible to create the layers if we have items of exactly the same height. However, this principle is inconvenient in many practical occasions. For example, when there is much variation in height in the items that need to be loaded, it may be the case that there are not enough items of the same height to fill a full layer. Besides, only allowing items of the same height to be on the same layer is conservative, and may result in a poor solution.

To tackle this problem, the algorithm makes use of 'superitems'. A superitem, $s \in S$, is an item consisting of multiple items on top of each other. This allows small items to be stacked together so that the height of a big item can be matched. Together they can form a layer. By introducing superitems it will be possible to construct layers consisting of a combination of tall and short items, while maintaining the homogeneous height of the layer. This provides more flexibility in the creation of layers. After all superitems are made, $S$ is added to $I$, so that $S \cup I = \{1, ..., n, n+1, ..., n+m\}$, with $m$ defined as the amount of created superitems. Note that the formulation of $I$ allows an item to occur multiple times in the set, either as an individual item or as part of one or more superitems. This will be clarified later in this section.

Superitems are defined as follows: $s = \{(i,j) \mid h_i + h_j = h_x \pm \delta,\ w_i d_i \geq w_j d_j,\ w_j d_j \geq 0.7 w_i d_i\}$. As shown, there are some restrictions when creating superitems. First, items are only chosen if they, when stacked together, match the height of an already existing item, defined as $h_x$. The height does need not match exactly, but should be within a small margin $\delta$, which is explained in the next paragraph. Furthermore, when packing a Standard Truck, superitems are only created of items $i$ and $j$ if $c_i = c_j$. For Bay Trucks at most 2 different customers can be present in a superitem. Note that the item with the highest surface is used as the bottom item. Due to stability reasons for stacking, superitems are only made when the surface of the upper item is at least 70% of the surface of the item below. The formulation of $s$ shown above regards superitems consisting of 2 subitems. In addition, we also allow for superitems consisting of 3 subitems, of which the formulation is given as $s = \{(i,j,k) \mid h_i + h_j + h_k = h_x \pm \delta,\ w_i d_i \geq w_j d_j \geq w_k d_k,\ w_j d_j \geq 0.7 w_i d_i,\ w_k d_k \geq 0.7 w_i d_i\}$

Before creating superitems, all items are grouped based on their height. When forming these height-groups, the 'height margin' should be taken into account. That is, all items in a layer share the same height including a small margin, named $\delta$. $\delta$ is set to 10 millimeters, as used in Elhedhli et al. (2019). This means that the height difference inside a height-group should at most be 10 millimeters. As a result, multiple groupings of item-heights can be formed. For example, 3 items with heights 190 mm, 200 mm and 210 mm could either be arranged as 190/200 and 210, or 190 and 200/210. The choice of how the height-groups are formed plays an important role in the creation of superitems. Therefore all possible height-group combinations are considered during pre-processing. For each combination of height-groups, all possible superitems are determined, which are all added to $I$. Not all superitems in $I$ will be actually part of the final loading plan, but this allows the iterative Layer-creating Algorithm to quickly choose which superitems are formed and which are not, without any further calculations.

## 5.2 Layer-creating Algorithm

The creation of layers is an iterative process, where each iteration differs in input. This difference lies in the composition of height-groups, the chosen superitems and their quantities. For each iteration, a subset of $I$ is chosen, in which each item is either present individually or as a part of a superitem. In addition to the random choice of the height-groups and superitems, several steps of the Layer-creating Algorithm contain randomness, allowing us to find a unique set of layers out of each iteration. Each iteration is called a 'layer-creating iteration', where $\lambda$ represents the total number of iterations. Out of all the created sets of layers, multiple loading plans are created, which are later used to find 1 solution, which is explained in section 5.4. The psuedocode of the Layer-creating Algorithm is given in Algorithm 1.

**Algorithm 1** Layer-creating Algorithm

**for** the number of layer-creating iterations ($\lambda$)
  **while** sufficient items to form a layer
    1. pick a starting customer
    2. **if** sufficient items to fill a layer for 70%
        create layers with 2D-packer (5.2.1)
        return to step 1
      **else**
        Pick (new) joining customer
          Standard Truck → pick either *preceding* or *succeeding* customer
          Bay Truck → pick customer sharing the most layers with the starting customer
        **if** no customer to choose from
          remaining items of starting customer are rest-items
          return to step 1
        **else**
          add superitems
          return to step 2

During the pre-processing of the data, all items are sorted by height. Each height-group is considered individually, and per group the Layer-creating Algorithm seeks to create as much 'good' layers as possible. To do so, the algorithm chooses a random height-group and initiates by picking a random 'starting customer', of whom we try to make new layers first. Two scenarios are possible when considering the items of this customer. First, if the items together cover a sufficient area so that they meet the 70% fill rate. In this scenario the items are packed with the help of a 2D-packing algorithm, as described in section 5.2.1. The 2D-packer tries to form as many layers as possible from these items. The second scenario concerns the case where the surfaces of the customer's items do not add up to 70% of the pallets surface, or, in rare cases, when the 2D-packer does not succeed in packing at least 1 layer while sufficient surface is available. In this case, items from another customer, but with the same height, are selected to be put together with the starting items. But, which costumer should we choose as the 'joining customer'? The answer to this question is different for Standard and Bay Trucks.

When considering Standard Trucks, pallets are unloaded from the back of the truck to the front. Therefore, it is important that items are placed in the right sequence. With this in mind, we choose the joining customer to be either the customer that is unloaded 1 stop earlier, or 1 stop later, which is chosen randomly. If none of these options contributes to reach the threshold of 70%, no layer is made and another starting customer is chosen.

Unlike Standard Trucks, Bay Truck come with the advantage that all pallets can be reached directly from the side. Therefore, customers who are unloaded in succession do not necessary need to be placed close to each other in the truck. On the other hand, having a small variety of customers in one pallet minimizes the total compartments that need to be opened, which can save handling time. When forming layers of multiple customers, it is preferred to form a layer of customers who are already put together in other layers. In this way a pallet may be formed with just a small variety of customers. Therefore, when picking the next customer it is considered which customers are already put together within a layer. The customer who shares the most layers with the starting customer is chosen as the joining customer. Otherwise, a random customer will be chosen. Note that were Standard Trucks could have a maximum of 2 different customers per layer, Bay Trucks typically include a larger variety of customers, wherefore they may include a total of 3 customers inside a layer.

Now that we discussed which customers are picked to form layers with, the question arises how the superitems fit into this procedure. The superitems consisting of just one customer don't cause any issue since they are just treated as a regular item. On the contrary, superitems of multiple customers are only inserted during the creation of multi-customer layers. That is, when multiple customers are chosen to be placed together on a layer, the superitems are added which share this customer-configuration. In this way, superitems do not increase the number of customers in a layer, while they may contribute in achieving the 70% threshold.

After all the steps have been completed, the Layer-creating Algorithm has formed a set of layers $L$, together with some rest-items. These are the items that did not succeed to be put into a layer, and are added on top of the pallets at a later stage. Rest-items are represented as $r \in R$, with $R$ being the set of rest-items. Despite not being guaranteed, we strive to minimize the size of set R by creating as many layers as possible. Set R containing many items may result in infeasibility, as we will place the rest-items at a later stage on top of pallets.

### 5.2.1 2D-packing algorithm

Once determined which items will be on the same layer, the items are packed using the 2D-packing module named Rectpacker, which is described in depth in Huang and Korf (2013). This module takes a set of items as input and returns 2 dimensional layers. Items can be packed in two dimensions in various ways, each method with its own pros and cons. Jylänki (2010) describes three different methods, namely the Shelf, Guillotine and Maximal Rectangles algorithm. When deciding for the right method it is important to focus on two factors, namely running time and the quality of the solution, which can be expressed as the fill rate of the created layers. The higher the fill rate, the better the algorithm manages to pack the items efficiently. For all three methods, different sub-variants are available which are listed in Jylänki (2010).

The Shelf Algorithm turned out to be both on running time and computational worse than the other options, and therefore only the Guillotine and Maximal Rectangles algorithm are tested thoroughly. The test results can be found in the Appendix (8) in figure 5, showing that the maximum rectangles algorithm with the Best Short Side Fit configuration is the optimal packing strategy. Therefore, only this algorithm is explained in this section.

The Maximal Rectangles Algorithm initiates by sorting the items on surface. The items will be placed one by one, starting with the item with the highest area. For each item, it is checked whether it can be placed inside an existing layer. If this is not possible, the item is added to a new empty layer. After all items have been placed, it is determined which layers meet the 70% fill rate. Layers that do not satisfy this threshold are taken down again, and their items are returned so they can be used later to create other layers.

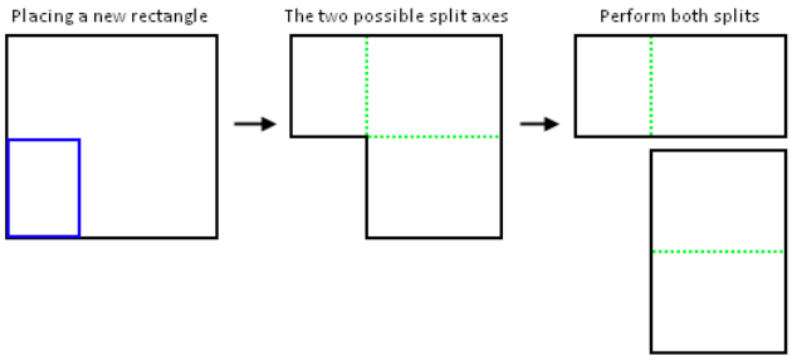



Figure 9: The splitting axis method of the Maximal Rectangles Algorithm is used to divide the available space into two boxes, Jylänki (2010)

But how does the algorithm determine at which location in the layer an item is placed, and in which orientation? The algorithm does this by dividing the free space in a layer into different sections. This splitting procedure is shown in Figure 9, and is performed before the placement of each item. Each new section, also called a 'box', forms a new option for the item to be placed. The different sub-variants of the Maximal Rectangles Algorithm differ in the way in which the algorithm chooses the box to place the item in. The Best Short Side Fit, which performed best, works as follows. When placing an item, the algorithm considers all the optional boxes, which is visualised in Figure 10. Then, for each box, the length of the longest leftover side is calculated: $max(w_b - w_i, d_f - d_i)$, where $w_b$ and $w_i$ are the width of the box and the item respectively. The depth of the box and the item are represented by $d_b$ and $d_i$. Note that an item will always be placed at the bottom-left corner of a box, and that for every box two possible rotations are calculated. After all boxes and rotations of the item are considered, the option with the minimal longest leftover side is chosen. The Maximal Rectangle Algorithm runs in $O(|B|^2 n)$ time, where $|B|$ is the set of maximal free boxes that represents the free area left in the bin at some packing step, and $n$ the number of items. This proposition is proven by Jylänki (2010).

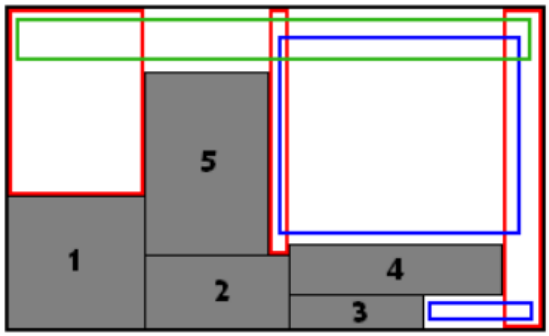



Figure 10: Possible item locations in the Maximal Rectangles Algorithm. The box is chosen in which the length of the longer leftover side is minimized, Jylänki (2010)

## 5.3 Palletizing Algorithm

For each layer-creation iteration, the Layer-creating Algorithm provides a set of layers $L$ and a set of rest-items $R$. The next step is to place all these layers and rest-items on pallets. For this process, the handling time is minimized. In order to find a solution, palletizing is split into 2 parts. Firstly, layers are placed onto pallets, while secondly the rest-items are placed on top of these pallets. Since Standard Trucks and Bay Trucks are unloaded differently, and typically have to serve a different amount of customers, the Palletizing Algorithm uses two different approaches for placing the layers onto pallets, specific on truck type. For Standard Trucks a greedy heuristic (5.3.2) is used to place the layers on pallets. This heuristic takes into account the location of the pallets inside the truck. The rest-items are then placed on top of pallets with the help of a greedy heuristic (5.3.4). On the contrary, a Bay Truck is independent on the location of the pallets since they can be accessed at all times. Furthermore, Bay Trucks face more complex scenarios since there is a much wider range of customers in the layers. To tackle this, two different palletizing algorithms are presented. The first algorithm partly consists of a greedy algorithm in which layers are placed onto pallets (5.3.3). Thereafter, the rest-items are placed on top of the pallet by the same algorithm that is used for Standard Trucks. This results in a full loading plan. Iterating over this algorithm creates multiple random loading plans, consisting of the same set of layers, which are used by Genetic Algorithm 1 (5.3.5) to improve the quality of the solution. The second algorithm we present is a MIP approach, described in section 5.3.6. The MIP approach seeks to find the optimal solution of assigning the layers and rest-items onto pallets. The two different approaches for palletizing Bay Trucks will be compared based on their obtained fitness values and corresponding running time. Based on these results we choose suitable palletizing approach to be part of the Layer-Based Algorithm. An overview of the palletizing steps for both Standard Trucks and Bay Trucks is shown in Figure 11

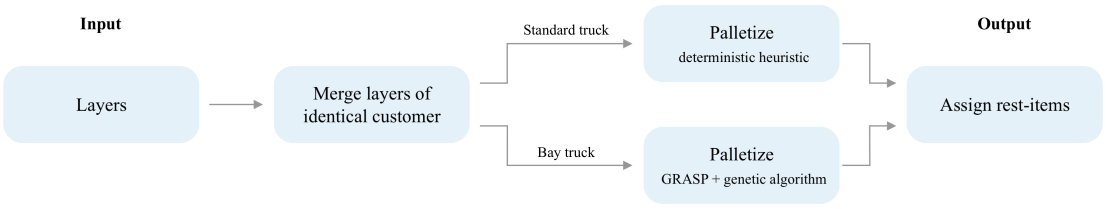



Figure 11: Visual representation of the palletizing procedure. A different heuristic is used for palletizing Standard Trucks and Bay Trucks.

The Palletizing Algorithm produces a full loading plan as output, in which the layers are assigned to pallets, with any rest-items placed on top. A loading plan is assigned a fitness value, which equals the total handling time. The penalty for opening a compartment, $CP$, is set to 15 seconds. For the relocation penalty $p_i$: $\alpha = 3$ and $\beta = 1$. That is, a fixed penalty of 3 seconds is given for each relocated item, while 1 second is added to this penalty for each kilogram of its weight. When considering Standard Trucks, only a relocation penalty is considered. This means that a final fitness value of 0 indicates optimality.In addition, Bay Trucks include $CP$. Since opening compartments is inevitable, the handling time for Bay Trucks cannot be equal to 0. To gain a better understanding of the optimality of Bay Truck solutions, we employ the following approach. For each customer involved, CP is subtracted from the fitness value for Bay Trucks, since at least one compartment must be opened for each customer. Additionally, we examine the total volume of each customer's items and subtract CP from the fitness value for the number of additional pallets required to accommodate this volume. The amount of additional pallets is calculated based on a pallet fill rate of 100%. As a result, the fitness value for Bay Trucks can also be equal to 0, which ensures that the loading plan is optimal. However, it does not necessarily mean that a fitness score above 0 is a non-optimal solution, since in some cases extra handling time may be inevitable.

### 5.3.1 Merge single-customer layers

Before the Palletizing Algorithm is used, layers consisting of the same customer are merged into one layer. This only applies to layers consisting of just one customer. As a result, the total amount of layers is reduced, bringing down the overall computational time of the subsequent Palletizing Algorithm. While merging layers, we might have to make a decision on which layers to bring together. This is the case when the total height of the layers exceeds the maximum pallet height. Then the question arises which layers will be merged together. A visualisation of the problem is shown in Figure 12. This problem is a Bin Packing Problem, where we want to minimize the empty space on a fully-loaded pallet. To solve this problem, a greedy heuristic is used. All layers are sorted by height, after which the highest layer is chosen first. For every layer, the greedy algorithm checks if it can be added to the highest pile of layers, without exceeding the maximum pallet height. Is there no pile of layers available? Then this layer will be the first layer of a new pile. This results in piles of merged layers, where every pile is considered as 1 layer in the Palletizing Algorithm.

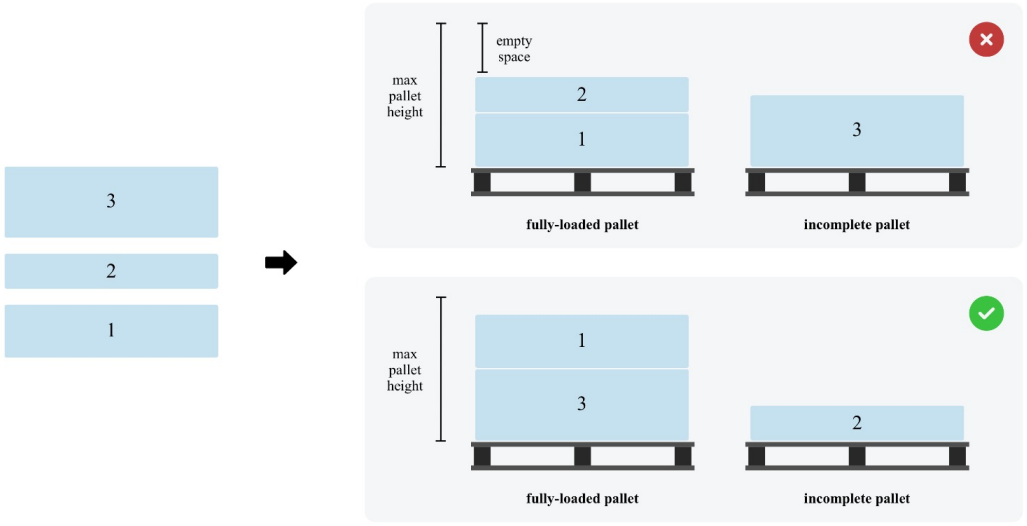



Figure 12: An example on how layers are merged in case they surpass the maximum pallet height. A greedy approach, where layers are sorted on height, minimizes the empty-space.

### 5.3.2 Layers: Standard Truck

For Standard Trucks, there are generally very few different customers in the truck. As a result, the final solution is unlikely to contain extra handling time. Therefore, a greedy heuristic is chosen for this type of problem. This enables us to quickly find a solution. To start the algorithm, layers are sorted based on customer, where layers containing two customers are sorted in between those customers. Then, starting with the latest customers, layers are placed as far in the back of the truck as possible, starting with the highest layer. This can be either on an empty pallet, a pallet containing items of the same customer or a pallet containing the customer that is unloaded one stop after. An example of this scenario is shown in Figure 13. In this example, an orange layer needs to be placed inside the truck. We prefer to place the layer on pallet 5, since this is the pallet furthest in the back that does not result in a penalty. If we cannot fit the layer on pallet 5, pallet options 4,3,2 and 1 are considered, in this order. Only if the layer is unable to fit in any of these pallet options, options 6, 7 and 8 are taken into account, in this order. In case none of the pallet options fit, auxiliary pallets are added to the truck on which these layers are packed. This results in an infeasible solution, in which every auxiliary pallet adds 1000 extra seconds to the total handling time. Infeasible solutions are punished rather than eliminated, since Genetic Algorithm 2 (5.4) may be able to use these loading plans to find a feasible solution.

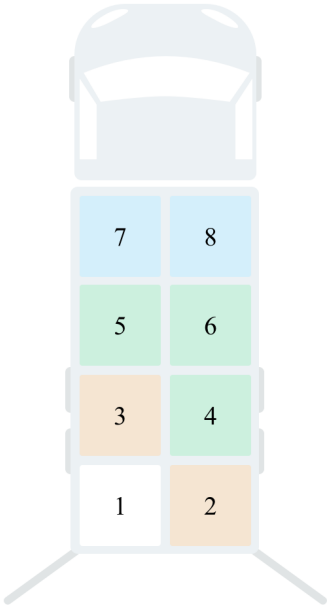



Figure 13: Standard truck, colors represent different customers. An orange layer can be placed on pallets 1-5 without penalization, while pallets 6-8 result in a penalty.

After all layers are designated to a location in the truck, rest-items are assigned to the pallets. Since Standard Trucks generally contain a small set of different item-types, $R$ is usually very small, containing only items which were not able to form a layer with a 70% fill rate with. These rest-items are placed on top of pallets, which is explained in section 5.3.4. After the rest-items are assigned to pallets, the total handling time is calculated. This calculation takes into account that the available pallets change during the unloading process of the truck.

### 5.3.3 Layers: Bay Truck

Bay Trucks come with the advantage that all pallets can be accessed at each stop. This makes their location in the truck unimportant, unlike in the case of Standard Trucks. What is important is that the items of identical customers should be distributed on as few pallets as possible, due to the extra handling time that is taken into account for opening a compartment. In this section we explain how layers are placed onto pallets for Bay Trucks, which consists of a GRASP algorithm. This algorithm involves placing layers on pallets and subsequently merging the pallets together. An overview of the algorithm is shown in Algorithm 2.

Unlike in case of Standard Trucks, the input of the Palletizing Algorithm for Bay Trucks consists generally of a lot of rest-items. To handle this large number of rest-items, all pallets are formed containing some empty-space on top. This space can be used to place the rest-items on. The height of this space is determined before the algorithm starts, and is equal to the highest rest-item, where the superitem rest-items are taken apart.

**Algorithm 2** Palletize layers

```
lower the max height of pallets by the height of the highest rest-item
place single-customer layers on unique pallets
for each multi-customer layer:
    assign layer to a pallet based on common customers
    place layer on the best location within the pallet (Layer-placement Algorithm)
for each combination of pallets:
    combine pallets if the handling time can be reduced (Layer-placement Algorithm)
while infeasible:
    for each combination of pallets:
        calculate the extra handling time (Layer-placement Algorithm)
        combine pallets which cause the lowest extra handling time =0
```

As a first step, a distinction is made between layers consisting of one customer, and layers of multiple customers. Note that the layers of one customer can in fact consist of multiple layers because of the merging procedure executed before. Then, each layer of one customer is assigned a unique pallet. The number of pallets used during this process does not have to match the available quantity of pallets in the truck. After this step, layers consisting of multiple customers are placed. The layers are randomly sorted and placed one by one on a pallet. The selection of the pallet is done in the following way. For each pallet, we first check whether the layer fits in terms of height. If this is the case, we check how many common customers there are between the layer and the pallet. Does the layer consist of customer 2 and 3, and there are already items packed of customer 2 on the pallet? Then this pallet is included as an option to be selected. When there are 2 common customers on a pallet, the probability of this pallet is twice as likely. An example of this procedure is given by Figure 14. Furthermore, when is at least one empty pallet left in the truck, this pallet is also considered as if it had 1 common customer. Out of all options, one pallet is then selected.

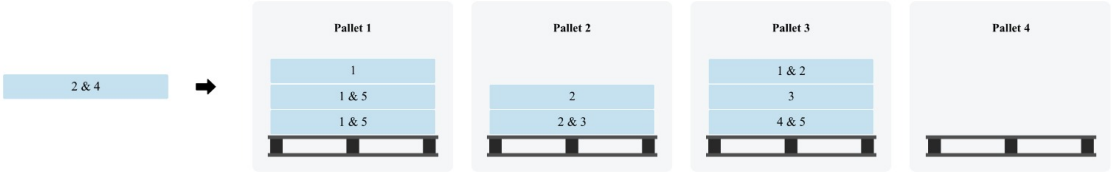



Figure 14: When selecting a pallet for the layer consisting of customer 2  4, pallets 2 and 3 are considered since they have matching customers. Pallet 3 is twice as likely to be selected, since it has two matching customers instead of one.

After a pallet is chosen, the question arises on which location on the pallet the layer will be placed. By location we mean whether the layer is placed above, below or somewhere between the already placed layers. This question is answered by the Layer-placement Algorithm. First, the algorithm checks if there is a location in which no handling time is involved. This is the case at a location where all customers below the layer are unloaded later, and all customers above are unloaded earlier. If no such location exists, then the relocation penalty is calculated for each location, after which we pick the location with the lowest penalty. During this process, not all locations are calculated. Only the locations which may contribute to a good solution are considered. For example, a layer with customer 2 will always be placed below a layer with customer 1, thus the locations above layer 1 are not considered. While calculating the handling time, we pay close attention to which items need to be moved during unloading. Consider the following example, which is visualised in Figure 15. We start with a pallet containing 2 layers. The upper layer contains items of customers 1 and 3, the bottom layer items of customer 2 and 3. In case this pallet is unloaded, a handling time is included for moving the items of customer 3 of the upper layer. This is because the items of customer 2 of the layer below are unloaded first. Then, a layer existing of customer 2 is added to the pallet. Naturally, this layer is placed in between the two layers on the pallet. Note that the inclusion of this layer does not affect the handling time of the pallet since the items of customer 3 on the upper layer are moved anyway in because customer 2 is already present beneath it. This example shows the calculations of the Layer-placement Algorithm while finding the handling time per location.

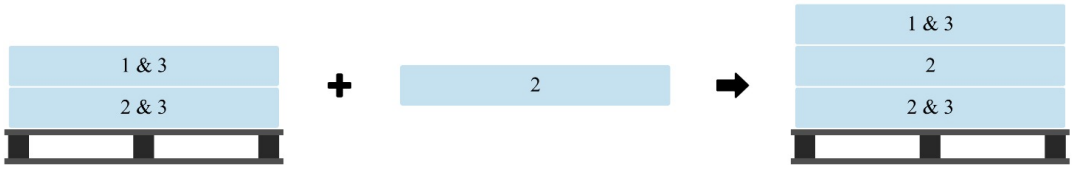



Figure 15: The handling time for this pallet does not increase when adding the layer of customer 2, since the upper layer is already impacted by this customer.

When all layers are placed, the pallets are considered for merging. That is, two pallets may be merged into one pallet to reduce the number of compartments that need to be opened during unloading. The algorithm starts by randomly picking two pallets which have at least one common customer, since this results in one less compartment to open. From these two pallets, it is firstly checked if the they can be merged without exceeding the maximum height of a pallet, minus the space that is assigned for the rest-items. Secondly, if the pallets can be merged due to height, the layers of the first pallet are combined and considered as 'one layer'. This single layer is then considered to be placed in the second pallet with the help of the Layer-placement Algorithm, which is also used before. However, the pallets are only merged if the (possible) additional handling time involved is smaller than the handling time for opening an extra compartment. In case the pallets are not merged, two new pallets are considered for merging. The process of merging pallets either terminates if a time-saving combination is found, or in case all combinations of pallets are unsuccessfully considered for merging.

When more pallets are packed than can fit in a truck, the solution is infeasible. As with Standard Trucks, an additional handling time of 1000 seconds is added for each excessive pallet. In case of infeasibility, the number of pallets could potentially be brought down by merging pallets. However, pallets are originally only merged if they cause a reduction in handling time. In case of infeasibility it is important to merge pallets at any costs. Therefore the Layer-placement Algorithm is used again, but without checking for a reduction in handling time. Instead, all combinations are calculated after which the option with the lowest handling costs is chosen. Is the solution still infeasible? Then the second-best option is also executed (including the newly formed pallet). This procedure continues until either the solution is feasible, or no more pallets can be merged. We now have allocated all layers to the trucks. The next task will be to place the rest-items on top of the pallets.

### 5.3.4 Rest-items

Placing rest-items is done by a greedy heuristic. This heuristic uses the created pallets as input, together with the set of rest-items. An overview of the full-process of assigning rest-items is shown in Algorithm 2. First, all pallets are checked for 'duplicates'. Two duplicate pallets are defined as containing the exact same customers, which means that while placing the rest-items there is no difference in the two pallets. This lowers the number of calculations required, which especially benefits Standard Trucks greatly due to the high similarity of their pallets. Before the rest-items are assigned to pallets, the superitems that are present in the set of rest-items are broken down. Then, all rest-items are sorted based on customer. For each customer, the total amount of handlings per pallet is calculated. For example, if a pallet consists of customer 1 and 2, then a rest-item of customer 4 needs to be handled twice on this pallet. All potential penalties are calculated, including $p_i$ and $CP$. The extra compartment penalty only applies to Bay Trucks, and only in the event in which a rest-item is placed on a pallet that does not include the customer of the rest-item. Then, for each item, all pallets are ranked based on the penalty that is involved when assigning the rest-item to it. Subsequently, each item is provisionally assigned to their 'favourite' pallet, the option which causes the least penalty. The algorithm then attempts to place the rest-items, starting with the pallet with the most area of rest-items allocated to it. All rest-items assigned to this pallet are sorted on surface, the biggest item will be placed first. This ensures optimal operation of the 2D-packer. The 2D-packer, the same as used during the creation of layers (5.2.1), is also tested on performance, from which we conclude that the Maximum Rectangles Algorithm with the Best Short Side Fit decision rule is the optimal packing strategy. This is the same strategy used for packing layers. In case the sum of the surfaces of all rest-items assigned to a pallet exceeds 60% of the surface area of the pallet, the packing algorithm will be utilized. Otherwise, we assume that the items can be placed on the pallet, which reduces the time consuming 2D-packing. If the surface area of the remaining items does exceed the 60% of the pallets' surface area, then the packing algorithm packs the rest-items in turn, trying to fit as many items onto the pallet. In case an item does not fit, it is assigned to its second-favourite pallet. This process is repeated for each pallet. It may be the case that there are still rest-items left to pack after all pallets are considered. This means that the found solution is infeasible, for which we assign a penalty to the solutions' fitness of 1% of the surface, given in centimeters, of the left-over rest-items. To place this penalty into perspective, a standard-sized tray of soda cans would induce a penalty of about 100 seconds in case we are not able to put in onto any of the pallets.

**Algorithm 3** Palletize rest-items

```
break superitems apart
for each rest-item:
    calculate rest-items' penalty per pallet
    assign rest-item to its 'favourite' pallet
for the number of pallets:
    pick the pallet to which the most area of rest-items is assigned to
    if the area is larger than 60% of the pallet surface:
        sort its rest-items from smallest to largest area
        assign as many rest-items as possible to the pallet using the 2D-packer
        assign not-packed rest-items to their second-best pallet option
    else:
        assign all the rest-items to the pallet
```

The 60% threshold is used to reduce the computation time of assigning rest-items. Although unlikely, it is possible that this threshold may result in infeasibility, which occurs when it is not possible to fit the items onto the pallet, while the item surfaces occupy only 60% of the pallet surface. After the Layer-based Algorithm is completed and a loading plan has been created, the rest-items are yet again assigned to pallets. This time, the 60% threshold is not included, to guarantee that if a feasible solution is found now, the items will physically fit on the pallets. Additionally, we obtain the coordinates for each item through the use of the 2D-packer. It may happen that during this final check, the configuration of the rest-items is found to be infeasible. In case of this unlikely event, it is possible to select not the best, but the second-best solution from Genetic Algorithm 2.

### 5.3.5 Genetic Algorithm 1: Bay Truck

The algorithm to palletize layers for Bay Trucks (5.3.3) is a Greedy Random Adaptive Search Procedure (GRASP), which may cause large variation in the quality of the solution. Besides this variation, it may also be possible that the GRASP is not able to find the optimal solution. Since the outcome of the Palletizing Algorithm may play a big role in the quality of the final solution, we propose a genetic algorithm which combines different randomly generated loading plans created by the GRASP.

The genetic algorithm works in the following way. First, a selection of the original population size is taken with the use of a tournament. In this tournament, 20% of the original population survives. This is done by organizing multiple completions in which one parent wins. For example, for a population size of 100, 20 completions are organized each containing 5 parents, resulting in a new populations of 20 parents. In the second step, 2 parents are taken from the new population randomly. Note that both parents have the same sets $L$ and $R$. From the first parent a pallet is chosen from which a group of layers is picked, e.g. $l_2$ and $l_4$. A group of layers is defined as multiple layers which are stacked on top of each other within the parents' loading plan. Both the choice of the pallet and the size of the group of layers is done randomly. The found layer group will be later used to create offspring with. Note that the orientation within the group of layers does not chance. The group of layers will be stacked in the same way within the offspring. After the group of layers from parent 1 is chosen, the algorithm repeats the process for parent 2. Note that from parent 2 we can now choose layers from the set $L \backslash l_2, l_4$. The process of picking a group of layers from a parent is repeated until all layers are chosen. Then, the layers, including the fixed groups of layers, are placed on a pallet using the original GRASP to place layers on pallets (5.3.3). After a loading plan for the layers is made, the rest-items are assigned to the pallets (5.3.4). This results in a full loading plan, with a possibly improved composition of layers per pallet. Each newly found loading plan is checked for uniqueness. If a found loading plan is already part of the population, it is rejected and 2 new parents are chosen. In case this scenario occurs 3 times in a row, the creation of offspring is aborted, and the genetic algorithm continues with the current pool of parents.

The survivors from the tournament create new offspring, which together form the next generation. This process has 2 different stopping criteria, which are determined based on the results. The first stopping criterion is the maximum number of generations, i.e. the depth of the algorithm. The algorithm terminates when this number of generations is reached, using the best known solution obtained at that point. Next to the depth, another stopping criterion is used which is able to terminate the prematurely. That is, the algorithm stops if a certain amount of generations have occurred in which no improvement in solution value is found. This stopping criterion may have a positive impact on the running time of the algorithm.

### 5.3.6 MIP approach

The Palletizing Algorithm assigns both layers and rest-items to pallets. This algorithm runs in reasonable time, which is essential since it is used frequently within the Genetic Algorithm 2. However, this may come at the expense of the quality of the loading plan. Therefore, we present a MIP approach for palletizing $L$ and $R$. Since the MIP approach solves to optimality, it can additionally used to give an insight in the quality of the solutions of GRASP and the Genetic Algorithm. For Standard Trucks, preliminary results show that the GRASP together with the Genetic Algorithm is frequently able to solve to optimality, which we know if the handling time is equal to 0. For Bay Trucks on the other hand, this is not the case. In order to find the optimal solution for Bay Trucks, and therefore find a lower bound for the Palletizing Algorithm, we use a MIP approach. Just as the GRASP, the MIP has a $L$ and $R$ as input. The output is a loading plan containing the exact locations of all items, together with the corresponding penalty. This penalty is minimized in the objective function of the MIP.

Inside the MIP-formulation the following sets are used. $L$ is the set of layers and $P$ the set of pallets. $I = \{0, ..., i_{max}\}$ is the set of indices on a pallet. The index denotes the location on the pallet. For example, when a layer is placed at $i = 0$, it is located at the the bottom of the pallet. The set of customers is defined as $C$. The index indicates the location on a pallet, e.g. a layer at index 0 indicates that the layer is located at the bottom of the pallet.

The MIP-formulation contains two matrices of constants, namely matrices $a$ and $b$. Matrices $a^1_{lc}$ and $a^2_{rc}$ indicate the penalty for layer $l$, or rest-item $r$ respectively, in case an item of customer $c$ is placed below the layer or rest-item. In other words, matrix $a$ denotes the penalty that is accounted for items that have to be moved in order to reach other items. Matrices $b^1_{lc}$ and $b^2_{rc}$ exists of binary values. $b^1_{lc}$ is equal to 1 if layer l contains customer c, and 0 otherwise. $b^2_{rc}$ equals 1 if rest-item $r$ belongs to customer $c$, and 0 otherwise. Other constants inside the MIP are the compartment penalty ($CP$), the maximum pallet height ($PH$) and the pallet surface ($PS$). The height of layers and rest-items is given by $h^1_l$ and $h^2_r$ respectively, while $s_r$ represents the surface of rest-item $r$. $q$ represents the number of the to-open-compartments of which a compartment penalty is inevitable. $q$ is pre-calculated by examining per customer the number of pallets that are at least required to load the total volume of their items.

The following variables are used inside the MIP. Binary variables $x_{lpi}$ indicate if layer $l$ is assigned to pallet $p$ at index $i$. For rest-items, binary variables $y_{rp}$ are used which are 1 if rest-item $r$ is assigned to pallet $p$, and 0 otherwise. The variables $z^1_{lc}$ and $z^2_{rc}$ are binary and indicate if layer $l$, or rest-item $r$ respectively, is affected by customer c. In other words, at least one item of customer $c$ is beneath layer $l$ on the same pallet. Binary variable $d_{cpi}$ indicates if an item of customer $c$ is packed at pallet $p$ up to, and including, index $i$. Furthermore, binary variables $k_{cp}$ are used to determine how many compartments need to be opened during unloading. That is, $k_{cp}$ equals 1 if an item of customer $c$ is packed on pallet $p$, and 0 otherwise. The variable $m_p \in N_0$ represents the highest rest item on pallet $p$.

$$\min_{z^1, z^2, k} \quad \sum_{l \in L} \sum_{c \in C} z^1_{lc} \cdot a^1_{lc} + \sum_{r \in R} \sum_{c \in C} z^2_{rc} \cdot a^2_{rc} + (\sum_{c \in C} \sum_{p \in P} k_{cp} - q) \cdot CP \tag{1}$$

$$s.t. \quad \sum_{p \in P} \sum_{i \in I} x_{lpi} = 1 \qquad \forall l \in L \tag{2}$$

$$\sum_{p \in P} y_{rp} = 1 \qquad \forall r \in R \tag{3}$$

$$z^1_{lc} + 1 \geq x_{lpi} + d_{cpi\text{-}1} \qquad \forall l \in L,\ \forall c \in C,\ \forall p \in P,\ \forall i \in I \setminus 0 \tag{4}$$

$$z^2_{rc} + 1 \geq y_{rp} + d_{cpi_{max}} \qquad \forall r \in R, \forall c \in C,\ \forall p \in P \tag{5}$$

$$d_{cpi} \geq d_{cpi\text{-}1} + \sum_{l \in L} x_{lpi} \cdot b^1_{lc} - 1 \qquad \forall c \in C, \forall p \in P,\ \forall i \in I \tag{6}$$

$$M \cdot k_{cp} \geq d_{cpi_{max}} + \sum_{r \in R} y_{rp} \cdot b^2_{rc} \qquad \forall c \in C, \forall p \in P \tag{7}$$

$$m_p \leq y_{rp} \cdot h^2_r \qquad \forall r \in R, \forall p \in P \tag{8}$$

$$\sum_{l \in L} \sum_{i \in I} x_{lpi} \cdot h^1_l + m_p \leq PH \qquad \forall p \in P \tag{9}$$

$$\sum_{r \in R} y_{rp} \cdot s_r \leq PS \qquad \forall p \in P \tag{10}$$

$$\sum_{l \in L} x_{lpi} \leq 1 \qquad \forall p \in P, \forall i \in I \tag{11}$$

$$\sum_{l \in L} x_{lpi} - x_{lpi+1} \geq 0 \qquad \forall p \in P, \forall i \in I \setminus i_{max} \tag{12}$$

The first step when breaking down each of the constraints of the MIP-formulation is to describe the objective function (1). The first part of the objective, including $z^1$ and $a^1$, penalizes for items that have to be moved in order to reach items underneath. These penalizations are for items that are part of a layer. The second part on the other hand, including $z^2$ and $a^2$, penalizes similarly, but for rest-items. The last part of the objective function, including $k$ penalizes for the total amount of compartments that are opened. Note that $q \cdot CP$ is subtracted from the objective function, acknowledging the compartment penalty that is made regardless, which is not part of the final objective value.

Constraints (2) and (3) make sure that every layer and rest-item is placed on a pallet. In constraint (4) and (5) $z^1$ and $z^2$ are defined, keeping track of which customers are located underneath the layers/ rest-items. Constraint (6) identifies which customers are present up to a certain index. Note that $d_{cpi}$ also has values for $i = -1$, which are 0 by definition. Constraint (7) keeps track of the customers that are present on a pallet. Constant $M$ is set to 1 + the number of rest items, since this is the maximum value that the right hand side of the equation can obtain. The highest rest item per pallet is defined by constraint (8). Constraint (9) ensures that the maximum height of a pallet is not surpassed, taking into account the height of the layers together with the highest rest-item on the pallet. Constraint (10) ensures that the area of all rest items does not exceed the surface of the pallet. Constraint (11) states that an index is not used more than once. Finally, constraint (12) prevents gaps in between layers, e.g. when index 0 and 2 are filled, but no layer is placed at index 1. Important to note is that experiments have shown that constraint (12) contributes to an increase in running time. Therefore, we choose to leave out this constraint and instead remove any gaps between the layers after a solution is found.

An important side-note to the MIP-formulation is that it does not guarantee feasibility. The reason for this lies in constraint 8. Here we assume that rest-items can be packed on a pallet not more than the pallets surface is being used. This constraint involves the following drawback. Namely, it can not be guaranteed that all rest items fit on a pallet, without overlapping each other. The 2D-packer described in section X could determine if the combination of rest-items can be feasibly packed, but this 2D-packer can not be implemented in a MIP-formulation. Therefore we choose to only restrict the rest items on the pallet surface, which results in a lower bound. This lower bound is equal to the optimal solution in case all rest-items are able to fit on their designated pallet, which can be checked after the solution is obtained with the use of the 2D-packer. Despite the fact that the obtained solution value is not guaranteed to be optimal in all cases, it can either way be used as a lower bound for GA1 to give a indication about the quality of its results.

## 5.4 Genetic Algorithm 2

For making a single loading plan, it is determined in advance which item heights are able to be placed on the same layer and which superitems are created, together with their quantities. These processes are performed randomly. Also random is the process of choosing which customers will be packed together on the same layer. All these random choices have their own influence on the creation of a loading plan. This implies that creating multiple loading plans results in a pool of potentially different solutions. This pool can then be used as the input for Genetic Algorithm 2. The goal of this algorithm is to find a newly, and hopefully better, solution selecting a combination of different layers from the original pool of solutions.

The principle of Genetic Algorithm 2 is very similar to Genetic Algorithm 1. We want to combine 2 loading plans, taking layers randomly from each parent, to form a new loading plan with the potential of having a better fitness value. The big difference however, lies in the different item compositions within layers. In other words, the layers of one parent are not the same layers of the other parent. This makes the process much more complex, since the items within a layer of parent 1 may be all spread out within different pallets for parent 2. Therefore, a slightly different approach is used for Genetic Algorithm 2, from which the pseudocode can be found by Algorithm 4.

**Algorithm 4** Genetic Algorithm 2: combine full loading plans

```
for the number of generations:
    organize tournament
    while the population is not full:
        select two parents
        while still layers available to select:
            select a group of layers from parent 1 or 2 to pass on to the child
            update available layers
        create new layers from the remaining items 5.2
        palletize layers and rest-items to obtain 1 child 5.3
        if the obtained child is unique in the current population:
            add child to population
        elif found 3 duplicate children in a row:
            break: stop increasing the population
            continue to next generation
    if best solution from population did not improve for x generations in a row:
        break: stop genetic algorithm
```

The first step of Genetic Algorithm 2 is to obtain a small selection from the population, which will be used to create offspring with. Like Genetic Algorithm 1 (5.3.5), a tournament is organized in which 20% of the population survives. Subsequently, selecting a group of layers randomly from a parent is done identically as in Genetic Algorithm 1. However, there is 1 significant difference. Namely, during the process of selecting layers, we keep track off all the item amounts. These are the amount of items which need to be packed into the truck, but are not (yet) packed. When selecting a group of layers from the first parent, we update the item amounts, so that we know which layers we can select at the next iteration from the second parent. A visualisation of this process is shown in Figure 16. This process is performed iteratively between both parents, until no more layers from any parent can be chosen. At this point, we have obtained a set of layers, which we define as $L^1$, together with a set of items. From these remaining items, we attempt to form new layers using the Layer-creating Algorithm (5.2), resulting in a new set of layers $L^2$. We then combine the two sets to form a new set of layers: $L = L^1 \cup L^2$. Besides the layers, we have set $R$, representing the rest-items. $L$ and $R$ are then palletized using the Palletizing Algorithm 5.3), which results in a full loading plan. Genetic Algorithm 2 uses the same stopping criteria used for Genetic Algorithm 1, where the value of the parameter $x$ - which indicate after how many generations without improvement in a row the algorithm terminates - is determined after evaluating the results.

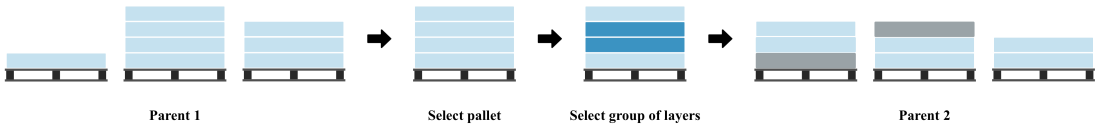



Figure 16: When generating offspring, GA2 picks a random pallet from Parent 1. A group of layers on this pallet is selected and given to the offspring. Subsequently, the available layers of Parent 2 are updated, from which a random pallet is selected again.

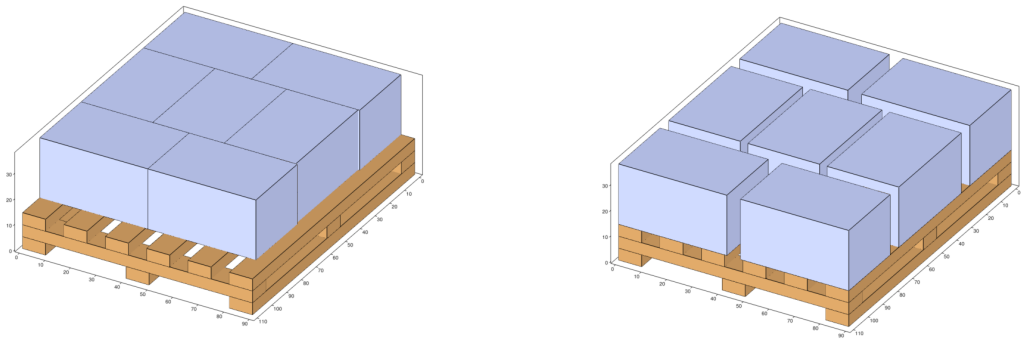
During the creation of off-spring, we want to avoid creating children that are already present in the population, as this can negatively impact the diversity of the population. To prevent this, the existing parents must be monitored. While this may not pose a problem in GA1, checking for duplicates in GA2 is more complicated. In GA1, it is evident if two parents are identical, since all parents contain the same set of layers. However, in GA2, the sets of layers differ. One could keep track of the location of each item in the loading plan, but this is a time-consuming process and would also allow for two nearly identical parents to be present in the population. To prevent this, duplicates are checked as follows. For each parent in the population, we keep track of the different item types per pallet. When a new child with an existing fitness value is produced, the item configuration per pallet is compared. If they match, the child is considered a duplicate solution and new offspring is generated.

## 5.5 Feasibility check

Using the Layer-based algorithm, we are able to find a loading plan in which the exact location of each item is determined in the truck. However, how do we know if this solution is feasible? To answer this question, we look at 4 different components for feasibility. Namely, stability, load bearing strength, stackability and occupied volume. The last-mentioned can simply be checked by looking at the number of used in the obtained loading plan. If more pallets are required than available in the truck, the solution is infeasible. For the other three components, a more comprehensive analysis is required to determine feasible. In this chapter, we describe methods that check for the stability, load bearing strength and stackability of items, allowing us to identify the feasibility of the obtained solutions. This provides a broader understanding of the quality of the solution, and can contribute to the tuning of the parameters regarding feasibility.

### 5.5.1 Stability

An item is considered to be stable if at least 1 of 2 requirements is met. First, at least 70% of an items surface should be supported, as also used by Gzara et al. (2020). If this does not hold, is it also sufficient if an item is supported at all its four corners. Here, each corner needs to be supported by at least 5cm to ensure stability. During the process of creating layers, a minimum fill rate of 70% is required by which we assume stability Gzara et al. (2020). Although unlikely, it may be the case that two layers can not be stacked stably on top of each other, despite both having a fill rate of at least 70%. To see if this is the case for an obtained loading plan, we check for stability for each individual item. To do so, we first have to distribute the items evenly within layers. That is, the 2D-packer, used to create the layers, places items without taking the distribution of a layer into account, as shown in Figure 17a. Therefore, to test for stability for each individual item, we first have to spread the items evenly within layers, as shown in Figure 17b



(a) (b)

Figure 17: On the left, items are placed with the help of the 2D-rectpacker. On the right, the items are spread out

To distribute items evenly on a pallet, we use a approach in which the items can be shifted within their layer. This approach uses linear constraints, allowing it to be solved with a solver. It is important that items can move without their x and y coordinates being taken into account simultaneously, which - together with the interaction of other items - would result in non-linearity.

Therefore, a LP-formulation is used in which items can only be moved in the positive direction, while maintaining their relative position to the other items. Within this LP-formulation, the following sets, constants, parameters and variables are used. Note that this program is used for each layer independently. The only set present is $I = \{1, 2, ..., n\}$, which includes all items on the layer. Constants $W$ and $D$ represent the width and depth of pallet, i.e. the maximum dimensions of a layer. Parameters $w_i$ and $d_i$ are the width and depth of item $i$, while $q_i^{\mathrm{x}}$ and $q_i^{\mathrm{x}}$ represent the initial x and y coordinates of item $i$. Furthermore, binary parameters $a_{ij}^{\mathrm{x}}$ and $a_{ij}^{\mathrm{y}}$ are predetermined to be 1 if item i and j are 'neighbours', and 0 otherwise. That is, item i and j are neighbours if j is on the right side of i, or above respectively. Note that this means that $a_{ij}^{\mathrm{x}}$ is only 1 if j is also within the depth of item i, as can be seen in figure 18. Variables $x_i$ and $y_i$ define the new x and y coordinates of item $i$. Finally, $d_i^x$ and $d_i^y$ describe the distance between item $i$, and its closest original neighbour, in x and y direction respectively.

$$\max_{d^{\mathrm{x}}, d^{\mathrm{y}}} \quad \sum_{i \in I} d_i^{\mathrm{x}} + d_i^{\mathrm{y}} \tag{13}$$

$$s.t. \quad d_i^{\mathrm{x}} \leq a_{ij}^{\mathrm{x}}(x_j - x_i - w_i) + W(1 - a_{ij}^{\mathrm{x}}) \qquad \forall i, j \in I \tag{14}$$

$$d_i^{\mathrm{x}} \leq a_{ji}^{\mathrm{x}}(x_i - x_j - w_j) + W(1 - a_{ji}^{\mathrm{x}}) \qquad \forall i, j \in I \tag{15}$$

$$d_i^{\mathrm{y}} \leq a_{ij}^{\mathrm{y}}(x_j - x_i - w_i) + D(1 - a_{ij}^{\mathrm{y}}) \qquad \forall i, j \in I \tag{16}$$

$$d_i^{\mathrm{y}} \leq a_{ji}^{\mathrm{y}}(x_i - x_j - w_j) + D(1 - a_{ji}^{\mathrm{y}}) \qquad \forall i, j \in I \tag{17}$$

$$x_j \geq a_{ij}^{\mathrm{x}}(x_i + w_i) \qquad \forall i, j \in I \tag{18}$$

$$y_j \geq a_{ij}^{\mathrm{y}}(y_i + d_i) \qquad \forall i, j \in I \tag{19}$$

$$x_i \geq q_i^{\mathrm{x}} \qquad \forall i \in I \tag{20}$$

$$y_i \geq q_i^{\mathrm{y}} \qquad \forall i \in I \tag{21}$$

$$x_i \leq W - w_i \qquad \forall i \in I \tag{22}$$

$$y_i \leq D - d_i \qquad \forall i \in I \tag{23}$$

The objective function (1) sums over the distances to each items closest neighbour in x and y direction. The objective function maximizes the minimum distance between the items, ensuring that items are placed as far apart from each other as possible. Constraints 2 - 5 define variables $d_i^x$ and $d_i^y$. Furthermore, constraints 6 and 7 make sure that items keep their relative positions to each other. Constraints 8 and 9 allow items to only move in positive directions, while constraints 10 and 11 make sure that all items stay within the pallets' dimensions.

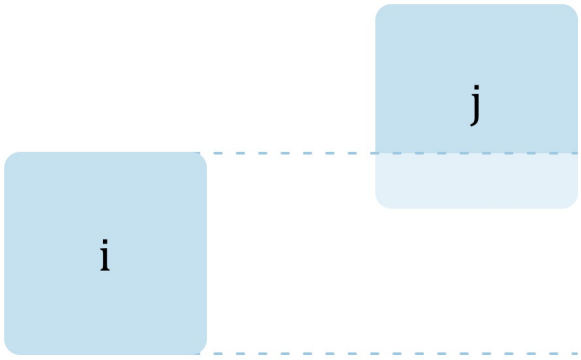



Figure 18: Parameter $a_{ij}^{\mathrm{x}} = 1$ since item $j$ is to the right of item $i$

Using the above LP-formulations results in evenly distributed layers, as can be seen in . Now, for each item it is possible to check for stability, by considering the support of its surface. However, it should be taken into account that this approach does not guarantee stability, since the relative orientation of items in between layers is not included in the LP-formulation. Therefore, examining stability results rather in an approximation than a hard conclusion. This also holds for the the other feasibility components as load bearing strength and stackibility. Allocating items to a location in a layer independently of these components provides a conservative results on feasibility. Nevertheless, the newly-distributed layers are used to provide a rough impression about the solutions' feasibility.

### 5.5.2 Load bearing strength

Each individual item has a maximum weight that it can bear, also referred to as the load bearing strength. In the final solution each item is checked on the weight that it carries. If at least one item carries more weight than it can bear, then the total solution is considered to be infeasible. In order to calculate the weight on each item, we create a graph of nodes and arcs, as used by Gzara et al. (2020). In this graph, each node represents an item, while an arc describes the distribution of its weight. The weight of each item is distributed based on its surface area by which it is supported. An example of a graph representation is shown in Figure 19. In this figure, 30% of the supported surface of item 5 is carried by item 3. Therefore, item 3 carries 30% of the weight of item 5. Since item 3 is supported by item 2, the weight of item 5 is then also transferred to item 2. Repeating this process for each nodes results in a full graph, by which the loaded weight per item can be found.

The data used in this thesis mainly exists of items used by the beverage industry, such as cans and bottles. These items come with the specific characteristic of having a high load bearing strength. Since it is unlikely that the load bearing strength of any item is violated, the Layer-based Algorithm does not take the load bearing strength into account when placing items on top of each other. However, to ensure feasibility, the final loading plan is checked on violations of the maximum load bearing strength.

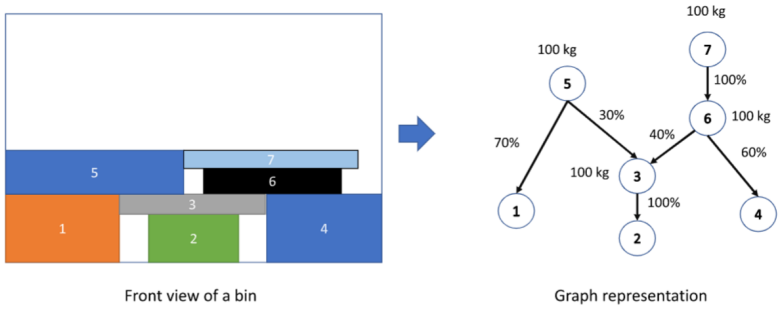



Figure 19: Graph representation of the items in a pallet, Gzara et al. (2020)

### 5.5.3 Stackability

Stackability plays an important role in the Layer-Based Algorithm. Namely, not being able to stack certain items can cause major limitations during palletizing. Ideally, to ensure feasibility, we only stack two layers on top of each other if we know with certainty that the non-stackable items do not have any surface overlap. However, during the process of palletizing, we do not know the exact locations of items within a pallet. At this stage, it is only known which items are present in the layer. The ultimate location of items within a pallet is only determined when the final loading plan is obtained. Therefore, we have to make an approximation whether 2 layers can be stacked validly on top of each other during palletizing. We do this by identifying the items that may cause stackability issues when placing the layers on top of each other. For each layer, we calculate how much of the total item surface consists of these problem-causing items. Summing the fractions of both layers indicates the likelihood that these two layers can be stacked on top of each other, without causing stackability issues. The value of this likelihood is referred to as $\alpha$. In case $\alpha = 2$, we are certain that the layers will cause stackability issues. On the contrary, $\alpha = 0$ assures that both layers can be stacked validly. For any value of $\alpha$ in between we cannot say with certainty whether the layers can be stacked on top of each other..

During palletizing, two layers are rejected to be stacked on top of each other if $\alpha > 0.4$. The value of 0.4 is chosen based on experiments. This threshold makes it unlikely for the layers to cause stackability issues, while allowing layers with a small $\alpha$ to be stacked freely each other. When a layer is rejected, the Palletizing Algorithm picks a new location, starting with the second-best option on the same pallet. The second-best location is also checked for stackability, and is only accepted if its penalty is within 25% of the penalty of the original option. If not, then the remaining pallet options are considered again.

# 6 Computational results

In this section, we present and analyse the computational results regarding the Layer-based Algorithm. In section 6.1, we show process of tuning the parameters of Genetic Algorithm 1 for the Palletizing Algorithm for Bay Trucks. The performance of Genetic Algorithm 1 is compared with the MIP approach in section 6.2. Thereafter, we tune the parameters for Genetic Algorithm 2 in section 6.3, after which the Layer-based Algorithm can be tested. The computational results of the Layer-based Algorithm are showcased in section 6.4.

All algorithms are programmed using Python 3.8.8, where CPLEX 20.1 is used as a solver for both MIP approaches. As an interperter, PyPy 3.11 is used. The computational outcomes were derived using an Intel(R) Core(TM) i7-8650U CPU processor equipped with 16 GB of RAM.

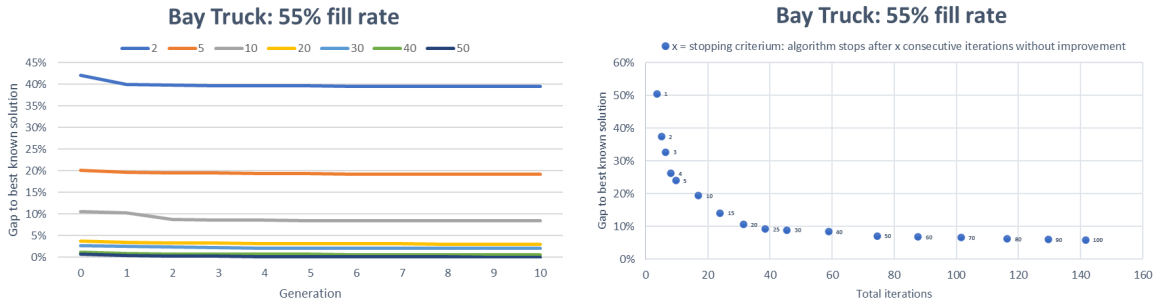
## 6.1 Tuning Genetic Algorithm 1

Both Genetic Algorithm 1 and Genetic Algorithm 2 require tuning of specific parameters in order to achieve optimal results. These parameters play a crucial role in determining the behavior of the algorithm and its convergence towards a solution. Without proper tuning, genetic algorithms may suffer from slow convergence or may not converge at all. Note that tuning Genetic Algorithm 1 only applies to Bay Trucks, as a different greedy heuristic is used for Standard Trucks. For Genetic Algorithm 1, we tune 2 different parameters. These are the population size and the number of generations, which are determined simultaneously. This is done by running the algorithm for a high number of generations, in this case 100, for various population sizes. For each chosen population size, we examine the percentage gap in fitness value relative to the best known solution over the number of generations. This gap, as any other percentage gap referred to in this section, is calculated as shown in Equation 1. In the numerator, +1 is added to allow calculations where the fitness value is zero. The best known solution is the result obtained after 100 iterations, i.e. generations, have been performed using the largest tested population size.

$$\text{Gap to best known solution} = \frac{\text{fitness found solution} - \text{fitness best known solution}}{\text{fitness best known solution} + 1} \cdot 100 \qquad (1)$$

In total, 50 different instances are tested. For each instance $\lambda = 10$, resulting in 500 tests, from which the average result per generation is determined. Figure 20a shows the obtained results for Bay Trucks with a 55% fill rate. From the figure, we observe that the percentage gap to the best known solution does not or barely change over the number of generations. The ultimate height of the percentage gap is almost exclusively determined by the population size. This holds for all 3 fill rates. From this, we conclude that Genetic Algorithm 1 fails to make a positive contribution to the fitness score of the obtained solution. Therefore, we decide not to use Genetic Algorithm 1 and instead focus on its preceding greedy heuristic.

The preceding algorithm used to place layers on pallets is a GRASP, meaning that multiple iterations are performed, resulting in a varied selection of solutions. To determine the optimal number of iterations needed to find a suitable solution, we introduce a stopping criterion the iterations of the GRASP. The GRASP terminates after $x$ solutions have been found in a row without an improvement in the best-found fitness value. To determine the optimal $x$, various values are tested. For each value of $x$, the GRASP terminates at a certain amount of iterations, which results in a loading plan. The fitness value of this loading plan is then compared to the fitness value of the best known-solution, which is obtained after 1000 iterations of the GRASP are performed. Figure 20b shows the average results on the same 500 test-cases that were used before. From these results, we determine the optimal $x$, taking the required iterations into account together with the percentage gap to the best known solution. It is important to note that the determination of $x$ is trade-off between computational time and quality of the solution. We select $x$ at the point where we only observe a small improvement in the percentage gap relative to the number of iterations. As a result, we select a $x$ of 4, 10 and 20 for Bay Truck fill rates of 15%, 30% and 55%, respectively. A complete overview of the computational results for all Bay




(a) (b)

Figure 20: In Figure a, the coloured lines represent population sizes showing the convergence rate of Genetic Algorithm 1. In Figure b, multiple values for the stopping criterion are analysed.

Truck fill rates can be found in the Appendix (8) in Figure 23.

## 6.2 Palletizing Algorithm vs MIP approach

In this thesis, we proposed two distinct methods for palletizing layers: the Palletizing Algorithm, comprising a GRASP, and a MIP approach. In order to assess the quality of the Palletizing Algorithm and determine which of the two approaches is most suitable, we compare the computational results of both algorithms. Only for Bay Trucks the heuristic is compared to a MIP approach, because initial experiments show that the greedy heuristic for Standard Trucks is already capable of creating optimal solutions. For each of the 3 different fill rates for Bay Trucks, namely 15%, 30%, and 55%, we examine 5 instances with $\lambda = 1$. We have selected only 5 instances because this approach allows us to gain specific insights into the characteristics of these instances. Besides, potentially long running times of the MIP approach prevent us from testing a large amount of instances. Additionally, we let the MIP run for a maximum time of 30 minutes. The optimality gap is calculated as shown in Equation 1, where the 'best known solution' is the obtained result of the MIP approach, which is optimal in case the MIP is not terminated prematurely. In other cases, both the current solution and the, possibly infeasible, lower bound solution is provided. The gap to the lower bound is calculated as the percentage gap between the found solution from the MIP approach, and its provided lower bound. The computational results are shown in Table 1.

Table 1: Comparison in fitness and time of the Palletizing Algorithm and the used MIP-formulation for different Bay Truck fill rates.

| | | 15% fill rate | | | | | 35% fill rate | | | | | 55% fill rate | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| heuristic | fitness | 59.7 | 164.6 | 15.0 | 15.0 | 15.0 | 131.9 | 90.0 | 193.9 | 51.4 | 157.7 | 470.6 | 3260.7 | 237.2 | 45.0 | 158.8 |
| | time (s) | .001 | .004 | .000 | .000 | .000 | .004 | .009 | .062 | .003 | .017 | .115 | .029 | .023 | .002 | .0327 |
| MIP | fitness | 49.8 | 132.0 | 0.0 | 15.0 | 15.0 | 96.2 | 74.8 | 92.4 | 49.8 | 115.8 | 131.5 | 102.7 | 120.7 | 45.0 | 130.7 |
| | LB* | - | - | - | - | - | - | - | - | - | 91.9 | 65.9 | 53.8 | - | - | 70.6 |
| | time (s) | .694 | 68.7 | .092 | .039 | .055 | 1250.2 | 25.3 | 805.5 | 2.5 | 1800 | 1800 | 1800 | 961.3 | .08 | 1800 |
| | gap (%) | 19.5 | 24.5 | 15.0 | 0.0 | 0.0 | 36.7 | 20.1 | 108.7 | 3.1 | 35.9 | 255.9 | 3045.3 | 95.7 | 0.0 | 21.3 |
| | gap to LB* (%) | - | - | - | - | - | - | - | - | - | 25.7 | 98.1 | 89.2 | - | - | 83.9 |
| | total items | 91 | 86 | 123 | 148 | 103 | 208 | 253 | 223 | 195 | 205 | 409 | 321 | 394 | 319 | 331 |
| | unique items | 45 | 62 | 42 | 6 | 17 | 80 | 63 | 92 | 63 | 94 | 100 | 81 | 50 | 298 | 86 |
| | customers | 2 | 16 | 1 | 1 | 1 | 4 | 5 | 12 | 3 | 7 | 14 | 6 | 12 | 1 | 5 |

* *Lower bound is given in case the MIP terminated prematurely.*

The computational results show the difference between the heuristic and MIP approach, with the most notable difference being the runtime. The MIP approach requires significantly more time, and for fill rates of 55%, it is not uncommon for the computations to be terminated prematurely after 30 minutes. For a low fill rate of 15%, the heuristic is occasionally able to find the optimal solution, but for higher fill rates, and cases with a high number of customers/unique items, we observe large optimality gaps on average. However, besides the quality of the result, a crucial aspect of the Palletizing Algorithm is its speed, as the algorithm is frequently used, mainly within Genetic Algorithm 2. Therefore, despite the frequent high optimality gaps, the heuristic is chosen over the MIP approach within the Palletizing Algorithm for Bay Trucks.

## 6.3 Tuning Genetic Algorithm 2

For Genetic Algorithm 2 we tune a total of 4 parameters. In addition to the population size and the number of generations we tune the number of layer-creating iterations $\lambda$, and the amount of parents taken per approach $\theta$. As an example: performing 100 layer-creating iterations with $\theta = 3$ results in 300 parents as an input for Genetic Algorithm 2. This number is then reduced to the population size by picking the best solutions, i.e. the solutions with the lowest fitness value. Note that $\lambda$ is defined as the fraction of the population size so that when $\lambda = 1$ exactly the population size is generated as input for Genetic Algorithm 2.

Tuning parameters for Genetic Algorithm 2 is a time-consuming process, as various values need to be tested for all 4 parameters. These parameters all impact each other, hence we prefer to tune them simultaneously. To do this in an efficient matter, we first focus on finding a rough estimate for each parameter, after which a more specific search can be performed per parameter. This allows us to search for the optimal value on a finer scale, while the risk of getting stuck in a local optimum is reduced.

To obtain an initial estimate of the optimal parameter settings a ranking system is used in which different configuration settings are compared and bad options are eliminated early on. The ranking system operates as follows. For each parameter a high and a low value is tested, resulting in 16 different parameter configurations. These values have been determined by conducting preliminary small tests to gain an understanding of suitable values. The 16 configurations are tested on an instance, obtaining the different fitness values and the corresponding computational times. Based on the obtained results, the configurations are ranked. Naturally, we aim for the lowest possible fitness value, but this should not come at the cost of a high running time. An optimal solution strikes a balance between these two factors. The different configurations are ranked by comparing each configurations' outcome with each other once. The comparison determines which of the two configurations is the best. This configuration receives a point. After all configurations have been compared, they are ranked based on the number of times they emerged as the best from the comparison.

The comparison of two results works as follows. When one of the two solutions has both a lower fitness value and a lower running time, it is undoubtedly better. However, for all other cases, the relative difference in time and fitness is considered. When one solution runs three times as long as the other, there is a relative difference of 200%. The solutions are considered to be equally good if the relative difference in fitness is a tenth of the difference in time, in this example 20%. If the other solution runs within this time, it is considered the best of the two. The factor of 10 is arbitrarily chosen and depends on the preference for a solution with a low fitness or short running time. As a side note, all solutions are artificially set to have a minimum of 3 seconds of running time. This ensures that we can make a fair comparison even if the configurations terminates almost instantly. In this, we assume that it does not matter for us whether a solution is found within or at exactly 3 seconds.

The ranking of solutions is performed in different rounds, each round consisting of 10 instances. After ranking the configurations 10 times, we add up all the rankings. If a configuration is scored best for an instance, it is awarded 16 points. The second-best configuration is awarded 15 points, and so on. This creates a new ranking that lists all configurations in order, based on their performance over 10 instances. The top 50% of configurations are then taken forward to the next round, for testing on 10 additional instances. This process is repeated 4 times until only one configuration remains. It is important to note that previous rounds are taken into account during the ranking process. After

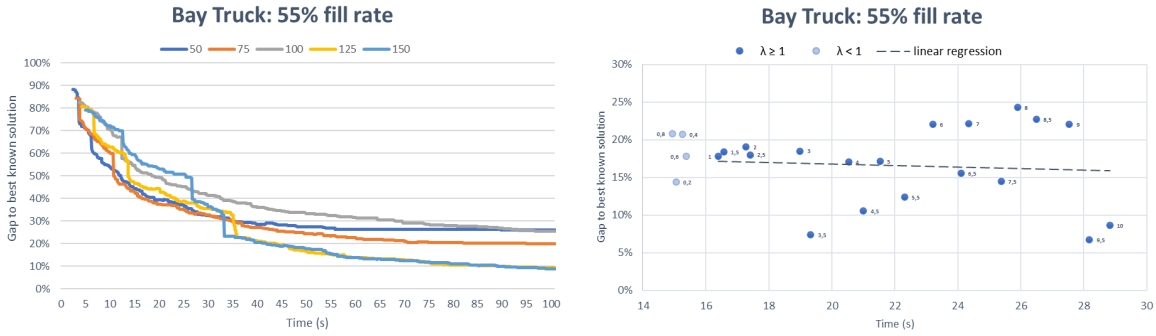
each round, the 50% worst configurations are eliminated and the scores of the remaining 50% are compared once again with each other to produce a new ranking. These scores are then carried over to the next round. This means that when reaching the last round, which consists of of only two configurations, the winner is determined by looking at the ranking across 40 instances, rather than just the 10 instances of that round. The configuration of the winner is used to further analyse the values of the parameters. The results from the ranking procedure are displayed in the Appendix (8) in Table 6.

After obtaining the initial parameters for the different trucks and fill rates, we refine these parameters more precisely. Firstly, we tune the population size and the number of generations, fixing $\lambda$ and $\theta$. Namely, we test how the gap to the best known solution varies over time for different population sizes. The best-known solution is determined by running four times the number of iterations that performed best in the ranking procedure, using the largest tested population size. For Standard Trucks, we tested 100 instances per fill rate, while for Bay Trucks, due to the longer running time, we tested 50 instances per fill rate. The results of Bay Trucks with a 55% fill rate are given in Figure 21a. The results of both Trucks, including all fill rates, can be found in the Appendix (8) in Figure 24.

It is important to note that there is no optimal point to stop the genetic algorithm, and that this is a trade-off between time and quality of the solution. We choose to stop the algorithm when the curve flattens and only a relatively small percentage of progress can be made over time. It is worth noting that in case time is less a significant factor, a higher value for the number of generations can be chosen.

Based on the obtained results, we pick a value for the population size together with a suitable running time. The number of generations is then determined by observing the average number of iterations performed by Genetic Algorithm 2 during the selected running time. The chosen parameters are shown in Table 2. For Standard Trucks with a fill rate of 15%, Genetic Algorithm 2 was not executed in any of the 100 instances, since an optimal solution was always found beforehand. We conclude that a genetic algorithm is not suitable in this case.

Subsequently, we determine $\lambda$ by running the instances again, but now for different values of lambda. The results for Bay Trucks with a 55% fill rate are shown in Figure 21b. From these results, we observe high variation for $\lambda < 1$. For $\lambda \geq 1$, a linear regression line is drawn. The regression shows a straight or slightly decreasing line for Bay Trucks, while for Standard Trucks we observe a slightly increasing regression. This may be due to higher values of $\lambda$ resulting in a lower diversity of the population. Since all tests indicate that higher values for $\lambda$ have little to no effect on the value of the solution, while runtime increases, $\lambda = 1$ is chosen for all trucks. The results show that this is the lowest value of $\lambda$ for which consistent results can be obtained.




(a) (b)

Figure 21: In Figure a, the coloured lines represent population sizes showing the convergence rate of Genetic Algorithm 1. In Figure b, the results for different numbers of layer-creating iterations $\lambda$ are shown.

Finally, we tune $\theta$. The ranking procedure shows that $\theta = 1$ is always preferred over $\theta = 5$. Testing $\theta$ more in depth reveals that increasing $\theta$ has a negative impact on the solution value. This is most likely due to the fact that increasing $\theta$ results in lower diversity in the population. Therefore, we decide to use $\theta = 1$ for all cases. Figure 2 shows the final configuration settings of all parameters.

Table 2: Final configuration settings of Genetic Algorithm 2 for the different Bay and Standard Truck fill rates

| | fill rate | population size | number of generations | layer-creating iterations ($\lambda$) | parents per iteration |
|---|---|---|---|---|---|
| Bay Truck | 15% | 50 | 85 | 1 | 1 |
| | 30% | 50 | 32 | 1 | 1 |
| | 55% | 125 | 16 | 1 | 1 |
| Standard Truck | 15% | - | - | - | - |
| | 35% | 30 | 15 | 1 | 1 |
| | 55% | 30 | 29 | 1 | 1 |
| | 75% | 40 | 13 | 1 | 1 |

## 6.4 Final results

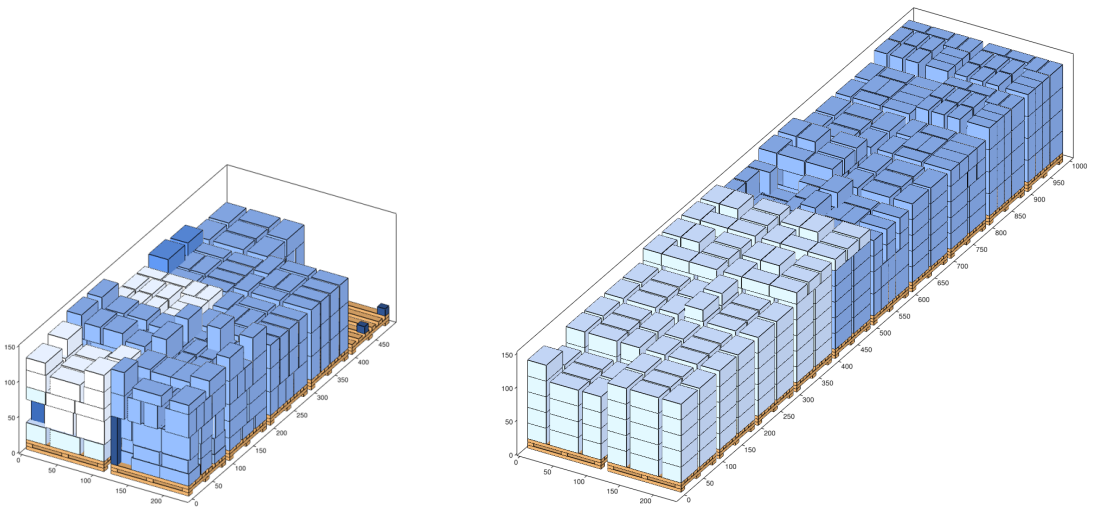
Now that a suitable palletizing algorithm has been selected and Genetic Algorithm 2 has been tuned, the layer-based algorithm can be tested. To do so, we examine 5 different instances for each fill rate. For each instance, multiple specifications are given, such as the number of unique items and the number of customers. This allows us not only to observe the fitness value and running time, but also to examine how these other factors affect the performance of the Layer-based Algorithm. A solution is considered to be feasible if all items are packed within the Truck. Besides, the specific feasibility of individual items is given regarding stability, stackability and load-bearing strength. First, we analyse the results for Bay Trucks, which are shown in Table 3. Thereafter, the results for Standard Trucks are presented in Table 4.

Table 3: Final results of the Layer-based Algorithm for Bay Trucks. For each fill rate, 5 instances are tested.

| | 15% fill rate | | | | | 35% fill rate | | | | | 55% fill rate | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| fitness | 0.0 | 9.5 | 64.9 | 0.0 | 0.0 | 50.8 | 15.0 | 15.0 | 30.0 | 4.1 | 191.3 | 190.7 | 145.8 | 58.3 | 69.5 |
| time | .78 | 10.21 | 5.43 | 5.26 | 7.12 | 25.43 | 4.19 | 4.31 | .04 | 13.67 | 35.21 | 59.76 | 53.83 | 44.01 | 44.22 |
| feasible | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| items | 102 | 189 | 97 | 112 | 142 | 276 | 232 | 209 | 248 | 213 | 321 | 349 | 312 | 352 | 335 |
| unique items | 36 | 116 | 73 | 40 | 49 | 95 | 37 | 26 | 4 | 77 | 81 | 181 | 154 | 100 | 115 |
| pallets | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| customers | 1 | 21 | 12 | 1 | 1 | 12 | 1 | 1 | 1 | 11 | 6 | 12 | 13 | 7 | 6 |
| superitems | 6 | 28 | 1 | 2 | 6 | 44 | 0 | 13 | 0 | 2 | 7 | 1 | 1 | 9 | 2 |
| moved items | 0 | 11 | 9 | 0 | 0 | 6 | 0 | 0 | 0 | 1 | 4 | 13 | 4 | 3 | 1 |
| not-packed items | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| opened comp. | 2 | 24 | 14 | 2 | 3 | 15 | 7 | 5 | 6 | 13 | 15 | 22 | 22 | 12 | 13 |
| infeasible items: | | | | | | | | | | | | | | | |
| stability | 9 | 15 | 7 | 14 | 12 | 28 | 23 | 13 | 2 | 14 | 21 | 46 | 36 | 31 | 30 |
| stackability | 0 | 4 | 2 | 0 | 3 | 3 | 1 | 2 | 0 | 2 | 3 | 7 | 2 | 2 | 4 |
| load bearing | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Based on the results, we observe that the Layer-based Algorithm is able to generate a feasible solution for all tested cases for Bay Trucks. However, for the majority of these cases, there is a handling time involved. Despite the handling time, it may still be possible that in certain cases the optimal solution is reached. Furthermore, the number of unique items and the number of customers impact the quality of the solution. Additionally, we see that superitems were used for almost every instance. That the use of superitems may play an important role in pallet loading is shown in Figure 22a, in which customer are represented by color. In this visualisation of instance 1 of the 55% filled Bay Trucks, 7 superitems are used on the pallet most in front. As a result, the dark blue item can be stacked within a layer and can therefore be placed at the bottom of the pallet.

Another remark of Figure 22a is that the before mentioned dark blue item could potentially be moved to the pallet on the far-right, thereby reducing the opening of an extra compartment. This highlights a limitation of the proposed Layer-based Algorithm. Layers are stacked up to the maximum pallet height minus the height of the tallest rest-item. This might make the Layer-based Algorithm perform poorly when there are tall, elongated items among the rest-items. Such objects are rare in the beverage industry but can have a significant impact on the algorithm's performance.



(a) Bay Truck 55% fill rate

(b) Standard Truck 75% fill rate

Figure 22: Visualisation of the final loading plan for a Bay and Standard Truck corresponding to their first instance in the results. Shades of blue represent customer order. Pallet measurements are shown in centimeters.

The results for Standard Trucks are shown in Table 4. From the results, we see that an optimal solution has been found for all instances up to a fill rate of 55%. However, for a fill rate of 75%, two of the five instances are not feasible. The cause of this lies in the functioning of the 2D-packer. A fill rate of 75% requires a high efficiency of the used space in a layer, which the 2D-packer cannot always guarantee. Moreover, for Standard items, we see that superitems play a less important role. A visualisation of instance 1 of the Standard Trucks with 75% fill rate is shown in Figure 22b. Additionally, a full overview of the visualisations of the first tested instances for all fill rates can be found in the Appendix (8) in Figure 26.

Table 4: Final results of the Layer-based Algorithm for Standard Trucks.
For each fill rate, 5 instances are tested.

| | 15% fill rate | | | | | 35% fill rate | | | | | 55% fill rate | | | | | 75% fill rate | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| fitness | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2383.6 | 0.0 | 25.6 | 1673.2 |
| time | .01 | .02 | .01 | .01 | .01 | .02 | .01 | .03 | .02 | .01 | .08 | 1.43 | .05 | .06 | .07 | .52 | 7.56 | .09 | 3.11 | .84 |
| feasible | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | no | yes | yes | no |
| items | 114 | 272 | 226 | 265 | 192 | 580 | 546 | 541 | 502 | 194 | 768 | 8140 | 720 | 807 | 796 | 830 | 985 | 1172 | 852 | 1280 |
| unique items | 45 | 208 | 3 | 6 | 5 | 15 | 8 | 10 | 43 | 129 | 1 | 6 | 1 | 19 | 6 | 13 | 8 | 10 | 6 | 1 |
| pallets | 10 | 26 | 20 | 20 | 26 | 26 | 20 | 20 | 26 | 10 | 20 | 26 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| customers | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 2 | 1 |
| superitems | 9 | 10 | 0 | 0 | 0 | 8 | 43 | 17 | 2 | 2 | 0 | 0 | 0 | 103 | 0 | 0 | 64 | 0 | 0 | 0 |
| moved items | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 0 |
| not-packed items | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | 0 | 0 |
| infeasible items: | | | | | | | | | | | | | | | | | | | | |
| stability | 3 | 11 | 0 | 1 | 3 | 12 | 8 | 5 | 5 | 6 | 1 | 2 | 0 | 16 | 3 | 6 | 12 | 34 | 34 | 1 |
| stackability | 0 | 6 | 0 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| load bearing | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Finally, we observe the feasibility of the obtained solutions regarding stability, stackability and load bearing strength. Starting with stability, we assumed that all items can be placed stabily as long as layers maintain a fill rate of at least 70%. Since this is a hard restriction for the creation of layers, we assume that despite some items being placed unstably, it will not lead to infeasibility. The reason for insufficient support of items in the obtained solutions lies in the adopted MIP approach, which shifts items per layer without considering the items directly above or below the layer. This also applies to stackability. During the palletizing of layers, we take into account that only two layers can be stacked on top of each other when $\alpha \leq 0.4$. Furthermore, we assume that this threshold for $\alpha$ will always result in infeasible solutions. Therefore, even though some items may not be stacked correctly in the obtained solutions, we assume that rearranging these items within their layers lead to feasibility. Lastly, we observe that load bearing strength does not lead to infeasibility in any of the tested instances.

# 7 Conclusion and discussion

In this thesis, we presented a solution approach to the the Pallet Loading Problem, while taking specific beverage industry constraints into account. The presented Layer-based Algorithm is a threefold in which the total handling time, i.e. the required unloading time of all pallets inside the truck, is minimized. Firstly, a greedy layer-generating algorithm is presented, which creates horizontal layers, while also allowing items to be stacked vertically within one layer. Secondly, three different palletizing approaches are presented to assign layers, and subsequently rest-items, to pallets. This includes two different greedy heuristics, based on truck type, a follow-up genetic-algorithm and a MIP approach. Thirdly, a second genetic algorithm is presented to combine and improve different loading plans. Subsequently, a MIP approach is presented to stably place the items on pallets, after which 3 feasibility checks are performed, regarding stability, stackability and load-bearing strength. Finally, the genetic algorithms are tuned with a proposed ranking system, after which the obtained loading plans are evaluated based on fitness value and computational time.

The data used in this thesis is provided by beverage company Coca-Cola. The proposed methods are specifically designed for the beverage industry, taking specific constraints into account, such as stackability, and taking advantage of the favorable characteristics of the items, such as the fact that low, wide items with high load-bearing strength are ideal for layer-stacking. In the data, two types of trucks are present, namely Standard Trucks that can be unloaded from the back, and Bay Trucks, which consist of different compartments that can be opened from the side. In addition to testing the layer-based algorithm on both truck types, a distinction is made for each type between different fill rates.

Firstly, in this thesis, we have examined how the heuristic palletizing algorithm performs compared to the MIP approach. Both algorithms were tested on Bay Trucks, from which we conclude that the solution quality of the heuristic does not benefit from the genetic algorithm, after which only the GRASP is used. Comparing the GRASP with the MIP indicates that the GRASP provides solutions with comparable quality in significantly less computation time for low fill rates of 15%, while for higher fill rate of 35% and 55% the optimality gaps of the heuristics' solutions increase drastically. Further research could potentially improve the solution quality of the heuristic, albeit at the expense of slightly longer running times. Overall, we can conclude that the heuristic approach is more suitable than the MIP approach in the layer-based algorithm due to its lower running time.

Secondly, we address the impact of different truck types on the results of the layer-based algorithm. In addition, we investigated the impact of different fill rates. In conclusion, we observe a significant difference in performance between truck types. Due to the one-sided nature of the data and the limited number of customers, the Standard Trucks are highly suitable for the layer-based approach. For fill rates of Standard Trucks up to 55%, the optimal solution is found for all tested instances, typically within a tenth of a second. For higher fill rates of 75%, infeasibility is observed in two out of five instances. For high fill rates of Standard Trucks, future research could focus on efficient 2D layer packing as it is the cause of the infeasibility.

For Bay trucks, the large variation in item sizes, combined with a wide range of customers, often leads to additional handling time in the fitness value. Nevertheless, the layer-based algorithm is able to find a feasible solution for all tested fill rates within a maximum of one minute running time. Overall, we conclude that the layer-based algorithm is highly suitable for Standard Trucks. Although we cannot determine whether the obtained values for Bay trucks are optimal, the results seem promising, as they combine feasibility with a reasonable running time.

Finally, we examine the feasibility of the obtained solutions. The presented MIP approach, to place items stably, spreads out the items within each layer. The results show, despite our assumption that stability issues would not arise, unstable placement of items in almost all tested cases. Future research might solve this issue by taking items from the layer below into account within the MIP. In addition, stackability could also be involved inside the MIP. Namely, in nearly all tested instances of Bay Trucks we have encountered stackability issues, despite taking stackability into account during the placement of layers. For Standard Trucks, we encounter virtually no stackability issues due to the small variation

in items. We did not have any load-bearing issues in any of the tested cases, due to the high item-strength specific for items in the beverage industry.

Taking all results into account, we conclude that a layer-based algorithm is promising for instances in the beverage industry. Especially for Standard Trucks, the algorithm generates optimal solutions very quickly. For Bay Trucks and fully packed Standard Trucks, further research is necessary to investigate the functioning of the palletizing GRASP, the 2D-packing algorithm, and MIP to feasibly spread items within a layer. Further research on these topics could potentially benefit the running time of the algorithm, together with its solution quality.

# References


Alonso, M. T., Alvarez-Valdés, R., Iori, M., and Parreño, F. (2019). Mathematical models for multi container loading problems with practical constraints. *Computers & Industrial Engineering*, 127:722–733.

Alonso, M. T., Alvarez-Valdes, R., Parreño, F., and Tamarit, J. M. (2016). Algorithms for pallet building and truck loading in an interdepot transportation problem. *Mathematical Problems in Engineering*, 2016.

Araujo, E. J., Chaves, A. A., de Salles Neto, L. L., and de Azevedo, A. T. (2016). Pareto clustering search applied for 3d container ship loading plan problem. *Expert Systems with Applications*, 44:50–57.

Araújo, O. C. B. d. and Armentano, V. A. (2007). A multi-start random constructive heuristic for the container loading problem. *Pesquisa Operacional*, 27:311–331.

Baker, B. M. and Ayechew, M. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800.

Billington, P. J., McClain, J. O., and Thomas, L. J. (1986). Heuristics for multilevel lot-sizing with a bottleneck. *Management Science*, 32(8):989–1006.

Bischoff, E. E., Janetz, F., and Ratcliff, M. (1995). Loading pallets with non-identical items. *European journal of operational research*, 84(3):681–692.

Bódis, A. (2015). Bin packing with directed stackability conflicts. *Acta Universitatis Sapientiae, Informatica*, 7(1):31–57.

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

Bortfeldt, A., Gehring, H., and Mack, D. (2003). A parallel tabu search algorithm for solving the container loading problem. *Parallel computing*, 29(5):641–662.

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

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

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

Chen, C., Lee, S.-M., and Shen, Q. (1995). An analytical model for the container loading problem. *European Journal of operational research*, 80(1):68–76.

Christensen, S. G. and Rousøe, D. M. (2009). Container loading with multi-drop constraints. *International Transactions in Operational Research*, 16(6):727–743.

Egeblad, J., Garavelli, C., Lisi, S., and Pisinger, D. (2010). Heuristics for container loading of furniture. *European Journal of Operational Research*, 200(3):881–892.

Elhedhli, S., Gzara, F., and Yildiz, B. (2019). Three-dimensional bin packing and mixed-case palletization. *INFORMS Journal on Optimization*, 1(4):323–352.

Glock, C. H., Grosse, E. H., and Ries, J. M. (2014). The lot sizing problem: A tertiary study. *International Journal of Production Economics*, 155:39–51.

Gzara, F., Elhedhli, S., and Yildiz, B. C. (2020). The pallet loading problem: Three-dimensional bin packing with practical constraints. *European Journal of Operational Research*, 287(3):1062–1074.

Hifi, M. (2004). Exact algorithms for unconstrained three-dimensional cutting problems: a comparative study. *Computers & Operations Research*, 31(5):657–674.

Huang, E. and Korf, R. E. (2013). Optimal rectangle packing: An absolute placement approach. *Journal of Artificial Intelligence Research*, 46:47–87.

Júnior, R. R., Yanasse, H. H., Morabito, R., and Junqueira, L. (2019). A hybrid approach for a multi-compartment container loading problem. *Expert Systems with Applications*, 137:471–492.

Junqueira, L., Morabito, R., and Sato Yamashita, D. (2012). Mip-based approaches for the container loading problem with multi-drop constraints. *Annals of Operations Research*, 199(1):51–75.

Jylänki, J. (2010). A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing. *retrived from http://clb. demon. fi/files/RectangleBinPack. pdf*.

Kang, K., Moon, I., and Wang, H. (2012). A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied Mathematics and Computation*, 219(3):1287–1299.

Karimi, B., Ghomi, S. F., and Wilson, J. (2003). The capacitated lot sizing problem: a review of models and algorithms. *Omega*, 31(5):365–378.

Liu, J., Yue, Y., Dong, Z., Maple, C., and Keech, M. (2011). A novel hybrid tabu search approach to container loading. *Computers & Operations Research*, 38(4):797–807.

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

Montoya-Torres, J. R., Franco, J. L., Isaza, S. N., Jiménez, H. F., and Herazo-Padilla, N. (2015). A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, 79:115–129.

Mostaghimi Ghomi, H., St Amour, B. G., and Abdul-Kader, W. (2017). Three-dimensional container loading: A simulated annealing approach. *International Journal of Applied Engineering Research*, 12(7):1290.

M.Ridder (2021). Leading beverage companies worldwide in 2021, based on sales. *Statista*.

Pacino, D. and Jensen, R. M. (2012). Constraint-based local search for container stowage slot planning. *Lecture Notes in Engineering and Computer Science*, 2:1467–1472.

Scheithauer, G. and Sommerweiß, U. (1995). *Heuristics for the rectangle packing problem*. Citeseer.

Scheithauer, G. and Terno, J. (1996). A heuristic approach for solving the multi-pallet packing problem. *Decision Making under Conditions of Uncertainty (Cutting–Packing Problems); Mukhacheva, EA, Ed*, pages 140–154.

Sciomachen, A. and Tanfani, E. (2003). The master bay plan problem: a solution method based on its connection to the three-dimensional bin packing problem. *IMA Journal of Management Mathematics*, 14(3):251–269.

Silvano Martello, D. P. and Vigo, D. (2000). *The three-dimensional bin packing problem. Operations Research 48, 2*.

Toth, P. and Vigo, D. (2002). *The vehicle routing problem*. SIAM.

Wang, H., Wang, Z., and Luo, J. (2012). A simulated annealing algorithm for single container loading problem. In *ICSSSM12*, pages 551–556. IEEE.

Zhao, X., Bennell, J. A., Bektaş, T., and Dowsland, K. (2016). A comparative review of 3d container loading algorithms. *International Transactions in Operational Research*, 23(1-2):287–320.

# 8 Appendix

Table 5: Average results of various 2D-packer configurations. The Max Rects Bssf configuration performs best on fill rate for both truck types.

(a) Bay Truck

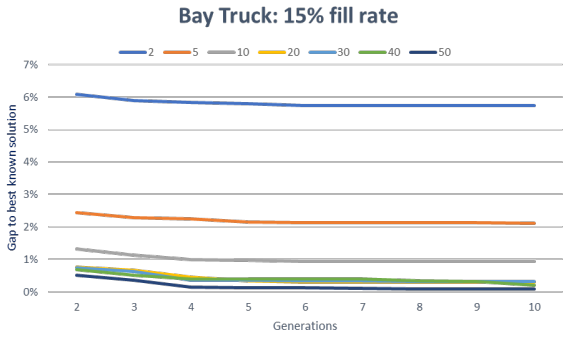
| Max Rects | | | |
|---|---|---|---|
| configuration | layers | fill rate (%) | time ($\cdot 10^{-2}$ s) |
| Bl | 19,1 | 78,5 | 5,7 |
| Bssf | 18,9 | 80,4 | 6,4 |
| Baf | 19,2 | 78,5 | 6,1 |
| Blsf | 15,2 | 75,5 | 6,8 |

| Guillotine | | | |
|---|---|---|---|
| configuration | layers | fill rate (%) | time ($\cdot 10^{-2}$s) |
| Bssf Sas | 19,5 | 76,5 | 5,9 |
| Bssf Slas | 19,6 | 76,5 | 5,9 |
| Bssf Maxas | 18,1 | 78,2 | 5,3 |
| Blsf Sas | 5,5 | 69,9 | 6,9 |
| Blsf Slas | 6,6 | 71,0 | 6,9 |
| Blsf Maxas | 4,9 | 69,0 | 7,3 |
| Baf Sas | 19,0 | 78,5 | 5,6 |
| Baf Slas | 19,1 | 78,4 | 5,6 |
| Baf Llas | 17,9 | 78,2 | 5,7 |
| Baf Maxas | 18,0 | 78,2 | 5,5 |
| Baf Minas | 19,3 | 77,8 | 5,5 |

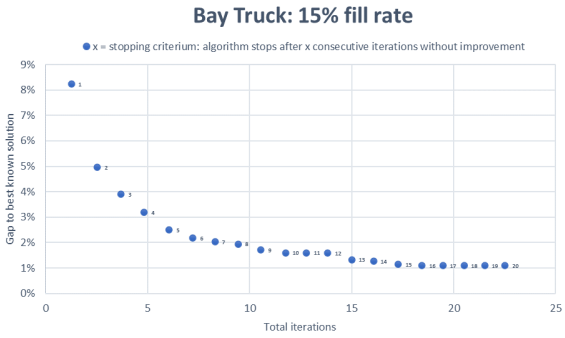
(b) Standard Truck

| Max Rects | | | |
|---|---|---|---|
| configuration | layers | fill rate (%) | time ($\cdot 10^{-2}$s) |
| Bl | 88,4 | 77,9 | 37,6 |
| Bssf | 88,8 | 79,5 | 48,1 |
| Baf | 87,8 | 78,6 | 39,4 |
| Blsf | 48,9 | 66,8 | 84,9 |

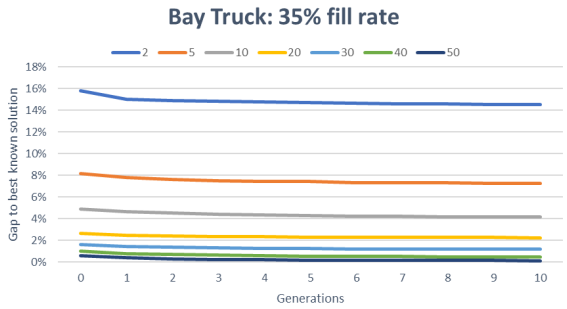
| Guillotine | | | |
|---|---|---|---|
| configuration | layers | fill rate (%) | time ($\cdot 10^{-2}$ s) |
| Bssf Sas | 89,7 | 73,5 | 59,9 |
| Bssf Slas | 89,6 | 73,4 | 61,5 |
| Bssf Maxas | 88,4 | 77,8 | 37,3 |
| Blsf Sas | 30,0 | 48,5 | 86,8 |
| Blsf Slas | 30,6 | 48,7 | 95,3 |
| Blsf Maxas | 25,7 | 47,6 | 93,5 |
| Baf Sas | 87,6 | 78,8 | 40,4 |
| Baf Slas | 87,8 | 78,8 | 41,2 |
| Baf Llas | 87,6 | 78,8 | 39,1 |
| Baf Maxas | 87,5 | 78,8 | 39,2 |
| Baf Minas | 88,6 | 77,9 | 43,1 |



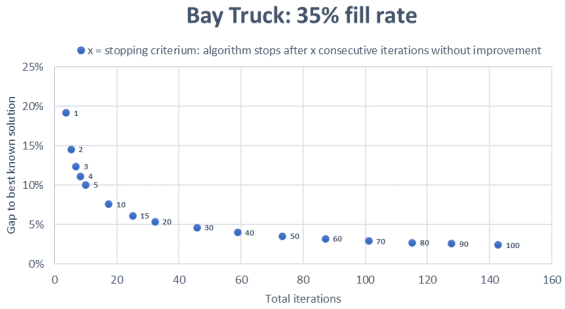

(a)



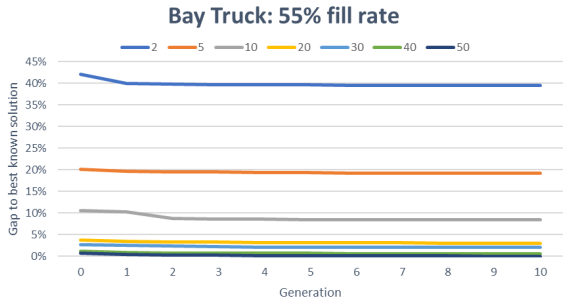

(b)



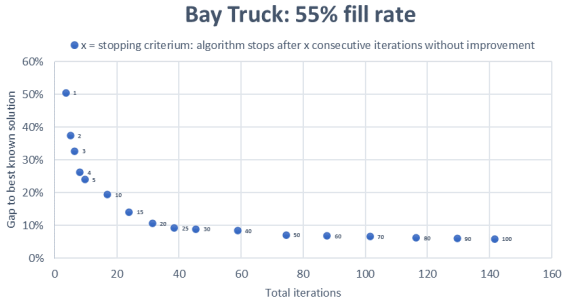

(c)




(d)




(e)




(f)

Figure 23: On the left, the coloured lines represent population sizes showing the convergence rate of Genetic Algorithm 1. On the right, multiple values for the stopping criterion are analysed.

Table 6: Results of the ranking system for tuning the parameters for Genetic Algorithm 2.

(a) Parameter values for the 16 configurations

| config. | population size | number of generations | layer-creating iterations ($\lambda$) | parents per iteration |
|---|---|---|---|---|
| 1 | 20 | 50 | 20 | 1 |
| 2 | 20 | 250 | 20 | 1 |
| 3 | 100 | 50 | 20 | 1 |
| 4 | 100 | 250 | 20 | 1 |
| 5 | 20 | 50 | 20 | 5 |
| 6 | 20 | 250 | 20 | 5 |
| 7 | 100 | 50 | 20 | 5 |
| 8 | 100 | 250 | 20 | 5 |
| 9 | 20 | 50 | 100 | 1 |
| 10 | 20 | 250 | 100 | 1 |
| 11 | 100 | 50 | 100 | 1 |
| 12 | 100 | 250 | 100 | 1 |
| 13 | 20 | 50 | 100 | 5 |
| 14 | 20 | 250 | 100 | 5 |
| 15 | 100 | 50 | 100 | 5 |
| 16 | 100 | 250 | 100 | 5 |

(b) Bay Truck 15% fill rate

| | round 1 | | round 2 | | round 3 | | round 4 | |
|---|---|---|---|---|---|---|---|---|
| config. | score | cum. | score | cum. | score | cum. | score | cum. |
| 1 | 32 | 32 | 17 | 37 | 8 | 24 | 3 | 10 |
| 2 | 57 | 57 | 28 | 62 | 8 | 38 | 4 | 23 |
| 3 | 54 | 54 | 36 | 62 | 10 | 41 | - | - |
| 4 | 73 | 73 | - | - | - | - | - | - |
| 5 | 48 | 48 | 39 | 65 | - | - | - | - |
| 6 | 70 | 70 | - | - | - | - | - | - |
| 7 | 60 | 60 | 34 | 63 | - | - | - | - |
| 8 | 74 | 74 | - | - | - | - | - | - |
| 9 | 69 | 69 | 29 | 67 | - | - | - | - |
| 10 | 42 | 42 | 30 | 54 | 17 | 45 | - | - |
| 11 | 66 | 66 | 30 | 68 | - | - | - | - |
| 12 | 80 | 80 | - | - | - | - | - | - |
| 13 | 76 | 76 | - | - | - | - | - | - |
| 14 | 85 | 85 | - | - | - | - | - | - |
| 15 | 84 | 84 | - | - | - | - | - | - |
| 16 | 101 | 101 | - | - | - | - | - | - |

(c) Bay Truck 35% fill rate

| | round 1 | | round 2 | | round 3 | | round 4 | |
|---|---|---|---|---|---|---|---|---|
| config. | score | cum. | score | cum. | score | cum. | score | cum. |
| 1 | 71 | 71 | 20 | 56 | 6 | 32 | 12 | 15 |
| 2 | 67 | 67 | 27 | 63 | 27 | 63 | - | - |
| 3 | 72 | 72 | - | - | - | - | - | - |
| 4 | 80 | 80 | - | - | - | - | - | - |
| 5 | 67 | 67 | 45 | 80 | - | - | - | - |
| 6 | 69 | 69 | 41 | 79 | - | - | - | - |
| 7 | 57 | 57 | 40 | 72 | - | - | - | - |
| 8 | 91 | 91 | - | - | - | - | - | - |
| 9 | 70 | 70 | 23 | 56 | 13 | 40 | 18 | 25 |
| 10 | 64 | 64 | 36 | 70 | - | - | - | - |
| 11 | 92 | 92 | - | - | - | - | - | - |
| 12 | 73 | 73 | - | - | - | - | - | - |
| 13 | 68 | 68 | 28 | 63 | 14 | 42 | - | - |
| 14 | 71 | 71 | - | - | - | - | - | - |
| 15 | 79 | 79 | - | - | - | - | - | - |
| 16 | 108 | 108 | - | - | - | - | - | - |

(d) Bay Truck 75% fill rate

| | round 1 | | round 2 | | round 3 | | round 4 | |
|---|---|---|---|---|---|---|---|---|
| config. | score | cum. | score | cum. | score | cum. | score | cum. |
| 1 | 70 | 70 | 45 | 82 | - | - | - | - |
| 2 | 52 | 52 | 46 | 77 | - | - | - | - |
| 3 | 42 | 42 | 27 | 57 | 10 | 37 | 8 | 24 |
| 4 | 38 | 38 | 24 | 50 | 17 | 43 | - | - |
| 5 | 85 | 85 | - | - | - | - | - | - |
| 6 | 83 | 83 | - | - | - | - | - | - |
| 7 | 45 | 45 | 40 | 71 | - | - | - | - |
| 8 | 54 | 54 | 40 | 74 | - | - | - | - |
| 9 | 74 | 74 | - | - | - | - | - | - |
| 10 | 74 | 74 | - | - | - | - | - | - |
| 11 | 71 | 71 | 21 | 63 | 9 | 36 | 2 | 16 |
| 12 | 79 | 79 | - | - | - | - | - | - |
| 13 | 108 | 108 | - | - | - | - | - | - |
| 14 | 111 | 111 | - | - | - | - | - | - |
| 15 | 73 | 73 | - | - | - | - | - | - |
| 16 | 57 | 57 | 36 | 68 | 24 | 60 | - | - |

(e) Standard Truck 15% fill rate

| | round 1 | | round 2 | | round 3 | | round 4 | |
|---|---|---|---|---|---|---|---|---|
| config. | score | cum. | score | cum. | score | cum. | score | cum. |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | - | - |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | - | - |
| 5 | 0 | 0 | 0 | 0 | - | - | - | - |
| 6 | 0 | 0 | 0 | 0 | - | - | - | - |
| 7 | 0 | 0 | 0 | 0 | - | - | - | - |
| 8 | 0 | 0 | 0 | 0 | - | - | - | - |
| 9 | 0 | 0 | - | - | - | - | - | - |
| 10 | 0 | 0 | - | - | - | - | - | - |
| 11 | 0 | 0 | - | - | - | - | - | - |
| 12 | 0 | 0 | - | - | - | - | - | - |
| 13 | 0 | 0 | - | - | - | - | - | - |
| 14 | 0 | 0 | - | - | - | - | - | - |
| 15 | 0 | 0 | - | - | - | - | - | - |
| 16 | 0 | 0 | - | - | - | - | - | - |

(f) Standard Truck 35% fill rate

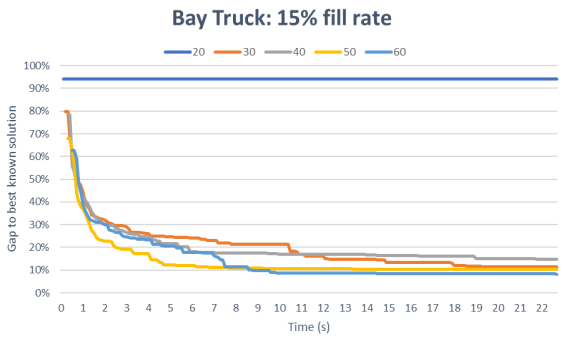
| | round 1 | | round 2 | | round 3 | | round 4 | |
|---|---|---|---|---|---|---|---|---|
| config. | score | cum. | score | cum. | score | cum. | score | cum. |
| 1 | 0 | 0 | 5 | 5 | 0 | 2 | 0 | 0 |
| 2 | 0 | 0 | 5 | 5 | 0 | 2 | 0 | 0 |
| 3 | 0 | 0 | 2 | 2 | 0 | 2 | - | - |
| 4 | 10 | 10 | - | - | - | - | - | - |
| 5 | 10 | 10 | - | - | - | - | - | - |
| 6 | 10 | 10 | - | - | - | - | - | - |
| 7 | 0 | 0 | 3 | 3 | 0 | 3 | - | - |
| 8 | 0 | 0 | 5 | 5 | - | - | - | - |
| 9 | 10 | 10 | - | - | - | - | - | - |
| 10 | 10 | 10 | - | - | - | - | - | - |
| 11 | 0 | 0 | 9 | 9 | - | - | - | - |
| 12 | 0 | 0 | 10 | 10 | - | - | - | - |
| 13 | 10 | 10 | - | - | - | - | - | - |
| 14 | 0 | 0 | 12 | 12 | - | - | - | - |
| 15 | 0 | 0 | - | - | - | - | - | - |
| 16 | 0 | 0 | - | - | - | - | - | - |

(g) Standard Truck 55% fill rate

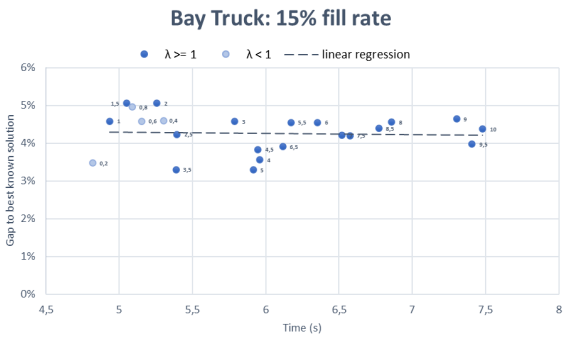
| | round 1 | | round 2 | | round 3 | | round 4 | |
|---|---|---|---|---|---|---|---|---|
| config. | score | cum. | score | cum. | score | cum. | score | cum. |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | - | - |
| 4 | 12 | 12 | - | - | - | - | - | - |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | - | - |
| 6 | 0 | 0 | 0 | 0 | - | - | - | - |
| 7 | 6 | 6 | 0 | 5 | - | - | - | - |
| 8 | 6 | 6 | 0 | 5 | - | - | - | - |
| 9 | 18 | 18 | - | - | - | - | - | - |
| 10 | 6 | 6 | 0 | 5 | - | - | - | - |
| 11 | 17 | 17 | - | - | - | - | - | - |
| 12 | 18 | 18 | - | - | - | - | - | - |
| 13 | 6 | 6 | - | - | - | - | - | - |
| 14 | 25 | 25 | - | - | - | - | - | - |
| 15 | 13 | 13 | - | - | - | - | - | - |
| 16 | 13 | 13 | - | - | - | - | - | - |

(h) Standard Truck 75% fill rate

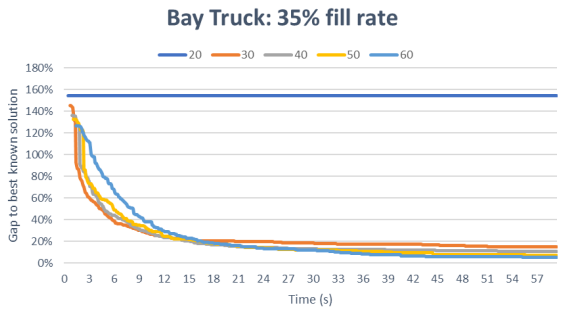
| | round 1 | | round 2 | | round 3 | | round 4 | |
|---|---|---|---|---|---|---|---|---|
| config. | score | cum. | score | cum. | score | cum. | score | cum. |
| 1 | 20 | 20 | 16 | 26 | 6 | 19 | 3 | 10 |
| 2 | 24 | 24 | 12 | 27 | 8 | 21 | 3 | 13 |
| 3 | 36 | 36 | 15 | 33 | 7 | 22 | - | - |
| 4 | 30 | 30 | 14 | 35 | - | - | - | - |
| 5 | 22 | 22 | 29 | 42 | - | - | - | - |
| 6 | 33 | 33 | 18 | 36 | - | - | - | - |
| 7 | 26 | 26 | 13 | 30 | 9 | 23 | - | - |
| 8 | 19 | 19 | 19 | 33 | - | - | - | - |
| 9 | 57 | 57 | - | - | - | - | - | - |
| 10 | 57 | 57 | - | - | - | - | - | - |
| 11 | 52 | 52 | - | - | - | - | - | - |
| 12 | 68 | 68 | - | - | - | - | - | - |
| 13 | 49 | 49 | - | - | - | - | - | - |
| 14 | 42 | 42 | - | - | - | - | - | - |
| 15 | 61 | 61 | - | - | - | - | - | - |
| 16 | 79 | 79 | - | - | - | - | - | - |



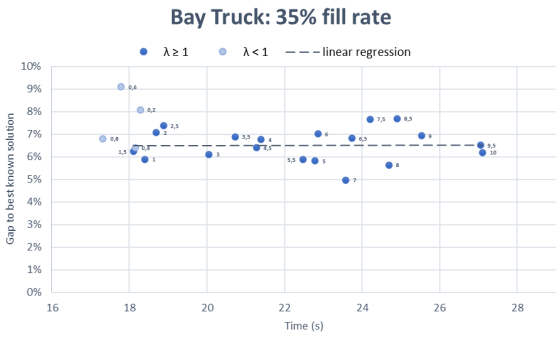

(a)



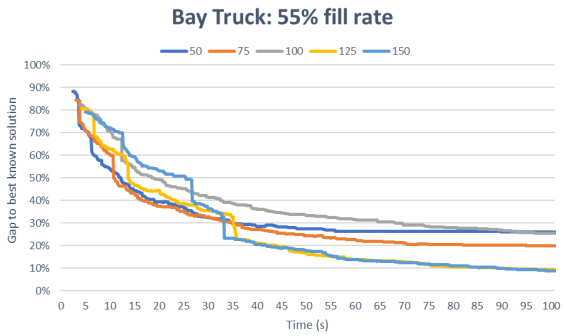

(b)



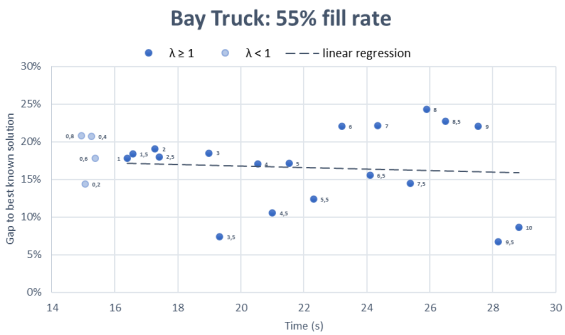

(c)




(d)

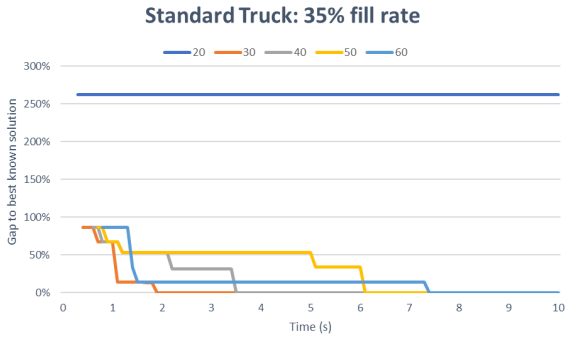



(e)

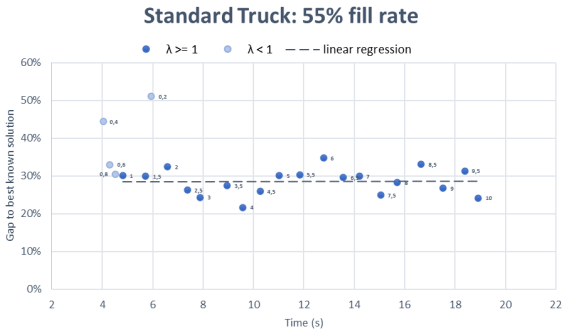



(f)

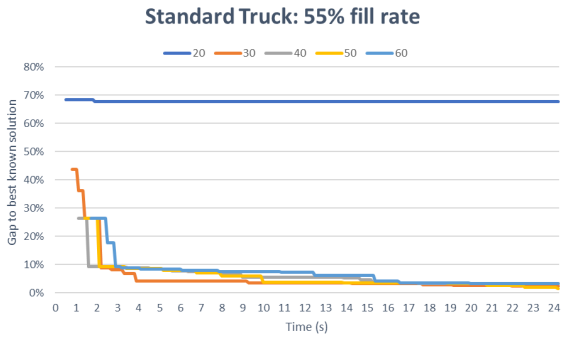
Figure 24: Results of Genetic Algorithm 2 for Bay Trucks, where the coloured lines on the left represent population sizes. The figures on the right represent the impact of the number of layer-creating iterations on the solution value.



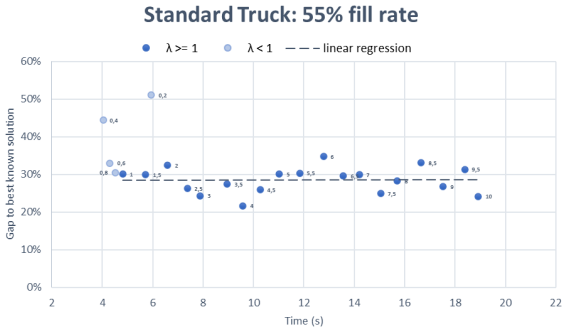

(a)



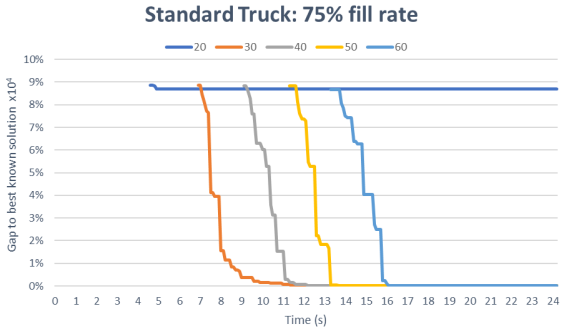

(b)



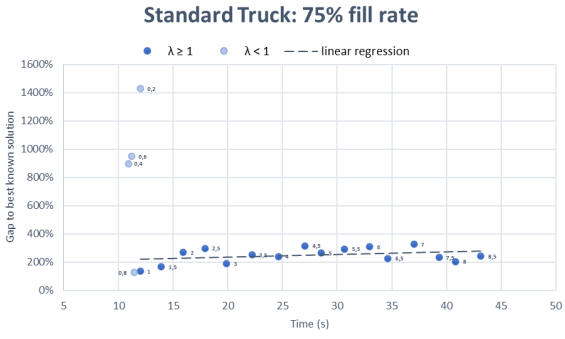

(c)




(d)

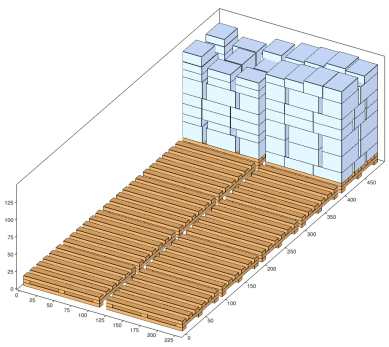



(e)

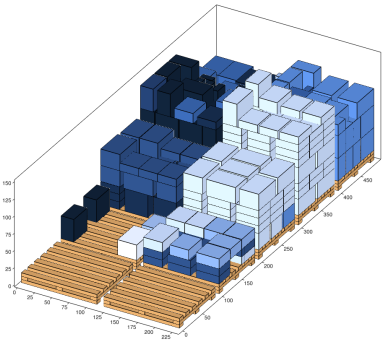



(f)

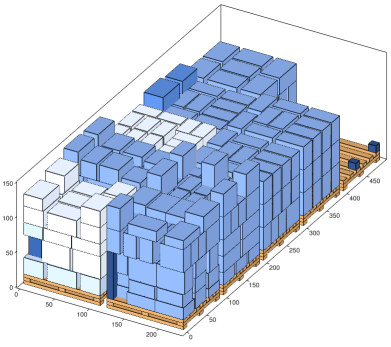
Figure 25: Results of Genetic Algorithm 2 for Standard Trucks, where the coloured lines on the left represent population sizes. The figures on the right represent the impact of the number of layer-creating iterations on the solution value.



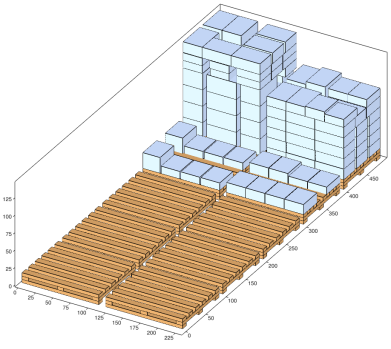
(a) Bay Truck 15% fill rate



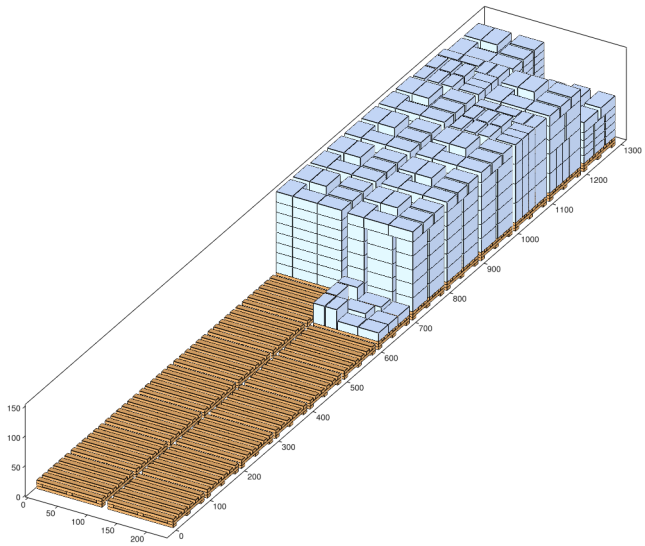
(b) Bay Truck 35% fill rate



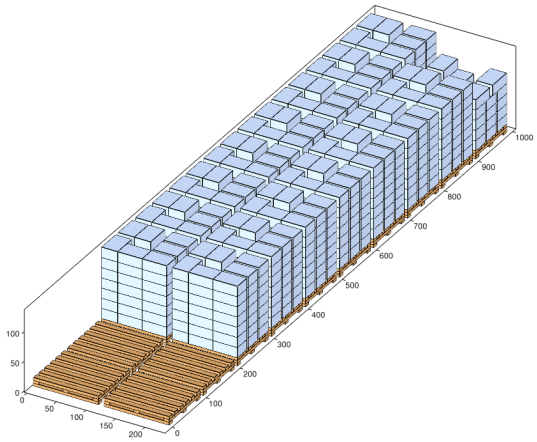
(c) Bay Truck 55% fill rate



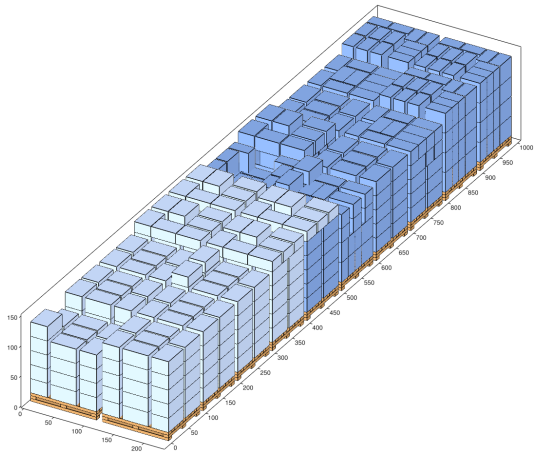
(d) Standard Truck 15% fill rate



(e) Standard Truck 35% fill rate



(f) Standard Truck 55% fill rate



(g) Standard Truck 75% fill rate

Figure 26: Visualisation of the final loading plan for Bay and Standard Trucks. Shades of blue represent customer order. Pallet measurements are shown in centimeters.