Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Sabine Seljeseth

Dividing Consignments

An Exploration of Different Methods

Master's thesis in Computer Science
Supervisor: Magnus Lie Hetland
June 2023

NTNU
Norwegian University of
Science and Technology

Sabine Seljeseth

Dividing Consignments

An Exploration of Different Methods

Master's thesis in Computer Science
Supervisor: Magnus Lie Hetland
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

In this thesis, different methods of dividing consignments are investigated. The motivation is to provide the Norwegian logistics and robotics company, Solwr, with more knowledge on what gives suitable results for their automated packing robot Grab. As of writing this thesis, they are searching for a suitable approach which takes Grabs performance into account.

The different approaches investigated were how a baseline heuristic used to solve the bin packing problem, Next Fit, performs compared to methods with the goal of minimizing the number of pallets needed to assign all the items to an order. Specifically, a MIP and the heuristic simulated annealing. Expanding the cost function for simulated annealing was tested to see if a multi-objective cost function would be more suitable than the single-objective cost function. The possibility of using machine learning to incorporate a model to predict the quality of an order according to Grab and use it as a cost function in simulated annealing was investigated.

The results indicate that the method of dividing a consignment which was the most promising to explore further was using a machine learning model in the cost function. This was the only method showing an increase in the average stability compared to the baseline method.

Sammendrag

I denne masteren blir ulike metoder for å dele opp større, sammensatte ordre til mindre ordre som har plass på en enkelt pall undersøkt. Motivasjonen bak detter er å gi det norske logistikk- og robotikkselskapet Solwr mer kunnskap om hva som gir gode resultater for deres automatiserte pakkingsrobot, Grab. Per dags dato søker de etter en egnet fremgang for å dele ordre som tar hensyn til Grab, og dens svakheter.

De ulike metode for å dele en større ordre som ble undersøkt, var hvordan en baseline heuristikk brukt til å løse pakkings problemet, kalt Next Fit, gjør det i forhold til metoder med mål om å minimere antall paller som trengs for å tildele alle varene til en bestilling. Spesifikt ble en Mixed-Integer Programming (MIP)-metode og heuristikken simulated annealing brukt. Utvidelse av kostnadsfunksjonen for simulated annealing ble testet for å se om en multiobjektiv kostnadsfunksjon ville være mer egnet enn en enkeltobjektiv kostnadsfunksjon. Muligheten for å bruke maskinlæring for å inkludere en modell for å forutsi hvilken kvalitet Grab hadde gitt ordren, og bruke det som en kostnadsfunksjon i simulated annealing, ble undersøkt.

Resultatene indikerer at metoden som virket mest lovene av de metodene som er testet og som kan være aktuell å utforske nærmere, var å bruke en maskinlæringsmodell i kostnadsfunksjonen. Dette var den eneste metoden som viste en økning i gjennomsnittlig stabilitet sammenlignet med grunnleggende metoden.

Acknowledgements

This endeavor would not have been possible without my advisor Magnus Lie Hetland and his guidance throughout the work of my thesis and the research project leading up to it. I would like to thank him for both the academic support and the encouraging conversations when I was doubting my efforts.

This thesis would not have been possible if not for the efforts from the team at Solwr. I would especially like to thank Sondre Rodahl for his support and always being there to answer my questions. I would also like to extend a thank you to Signe Riemer-Sørensen from SINTEF and Håkon Hukkelås from NTNU for valuable insight and recommendations regarding my thesis.

Lastly, I would like to thank my friends and family for always believing in me.

Contents

Abstract **i**

Sammendrag **iii**

Acknowledgements **v**

Table of Contents **viii**

1 Introduction **1**
1.1 Structure 4
1.2 Related Work 4

2 Background **7**
2.1 Bin Packing and Pallet Loading Problem 7
2.2 Grab 11
2.3 Simulated Annealing 13
2.3.1 Iterative Improvement 13
2.3.2 An Analogy 14
2.3.3 Metropolis Criteria 14
2.3.4 Requirments for implementation 15
2.4 Regression 16
2.4.1 Decision Trees 16
2.4.2 KNN 20
2.4.3 Evaluation of a Model 20

3 Implementations **23**
3.1 The Data 23

3.2 Constraints . . . 27
3.3 Next-Fit Heuristic . . . 28
3.4 Mixed Integer Program . . . 29
3.5 Simulated Annealing . . . 31
3.5.1 Configurations . . . 32
3.5.2 Move set . . . 35
3.5.3 Cost Function . . . 36
3.6 Creating A Suitable Data Set . . . 39
3.6.1 First Iteration . . . 39
3.6.2 Second Iteration . . . 40
3.7 Choosing the Model . . . 41

4 Results **43**
4.1 Comparing Baseline Heuristic to MIP . . . 43
4.2 Comparing Baseline Heuristic to Simulated Annealing . . . 45
4.3 Comparing Simulated Annealing with Single-Objective Cost Function to Multi-Objective Cost Function . . . 46
4.4 Adding Machine Learning to Simulated Annealing . . . 46
4.5 Example of Splitting a Consignment . . . 54
4.5.1 Baseline . . . 55
4.5.2 MIP . . . 57
4.5.3 Simulated Annealing . . . 59

5 Discussion **67**
5.1 Future Work . . . 73

6 Conclusion **75**

References **77**

Appendices **81**

A MIP **83**

B Next-Fit Implementation **87**

C Simulated Annealing **89**
C.1 Move set . . . 89
C.2 Cost Function . . . 90
C.3 Configuration . . . 92

Chapter 1

Introduction

The boom in technology since the 2000s have sparked a new wave of the industrial revolution, also known as *The fourth industrial revolution* or *Industry 4.0*. With this, the subfield *Logistics 4.0* emerged, with the aim to provide support for the processes in industry 4.0. In this subfield of logistics, technologies such as robotics and automation are included. This is to improve safety levels, reduction of errors, necessary labor and costs, among other things. Radivojević and Milosavljević stated that it is not a matter of choice, but a matter of time before every company uses these technologies of the future [1].

One company reaping the benefits of using the technologies of logistics 4.0 is Amazon. Amazon is one of the largest e-commerce companies in the world and are currently deploying robots in their warehouses. They estimated that adding robots to their warehouses reduced operational cost by around 20%. In addition, it also allowed them to maximize inventory space, since the robots did not need as much space as a human would to navigate through the aisles [2].

Not all companies are looking to make their warehouses fully automated, which is why *hybrid order picking system* exists. This system is categorized as an environment where human workers and picking robots are working together in a shared space. Research conducted on this picking system shows that it can provide a reduction in both costs and human energy expenditure, while still keeping the average throughput time near a manual system [3].

A norwegian company working in the field of combining logistics and robotics

is Solwr. Solwr states that their goal is to creating scalable, sustainable and integrable solutions [4]. One of these solutions is the fully automated robot *Grab*, which creates a new way for distribution centers to fulfill orders. *Grab* can be integrated in distribution centers without any major structural changes and can aid workers with tasks that are heavy and repetitive. At the time of writing this thesis, the workflow in a distribution center employing Grab can be described as follows;

> Customers order a collection of items from the distribution center. These can be regular orders or orders of a higher urgency. The orders are then pooled together based on the customer they are going to, this is called a consignment. These consignments are then split into orders or pick routes, where one order is meant to fit on one pallet. In today's solution, the Warehouse management system (WMS) have their own heuristics for splitting a consignment into orders. Some of these orders are then sent to Grab, where it will start from the top of the order list and work its way sequentially down to the last item. The finished order is then placed with the other orders going to the same store, ready to be loaded onto a transportation truck.

The current solution for dividing consignments does not factor in that Grab might be the one packing the order. Grab cannot pack everything in the warehouse like a human worker, which means that orders must be filtered before being sent to Grab. Solwr therefore is interested in a new way of dividing consignment, where the method is tailored to suit Grab's strengths and weaknesses. Solwr found that filtering orders and giving Grab the ones they believed would play to its strengths increased the weight and volume Grab was able to pack in a day, see table 1.1. The hope is that having a generalized method for dividing a consignment which suits Grab will be more efficient, which in turn will give an increase in the value added for a distribution center choosing to implement Grab.

	Orders picked in a workday Priority vs random						
	Weight (kg)	**Volume (m^3)**	**# Dimension diversity / Stability**	**# D-packs**	**# pick orders**	**# aisles / distance**	**D-pack per hour**
Prioritized	5062	8.94	202	473	13	23	63
Random	3011	6.22	440	333	17	52	44
Change (%)	68%	44%	-54%	42%	-25%	-56%	43%

Table 1.1: Results from Solwr's research on prioritizing orders vs choosing them randomly

During the initial discussions with Solwr, the focus was on solving their multiple container loading cost problem. The problem involved the loading of multiple boxes onto pallets, with each loaded pallet having a cost. The objective was to minimize the total cost for all the pallets going to a store. While Solwr had the algorithm of how to pack an order, they faced the challenge of determining an appropriate cost function for splitting the consignment into orders which Grab would pack. They expressed that adding a pallet to the solution should result in a higher cost but were unsure about the specific cost function to use.

Solwr provided valuable insights related to the factors influencing the loaded pallets' costs. These factors included the total weight picked per minute picking time, stack capacity, and stability. However, it was decided to exclude the picking time from the problem formulation. As a result, the penalty for driving distance was removed.

In terms of the cost function, Solwr suggested several possibilities and made certain assumptions. The suggested cost function options included item dimension heterogeneity, item height heterogeneity, weight distribution, or a combination of these factors. These suggestions were based on what Solwr thought might produce pallets which were stable and compact. Additionally, Solwr assumed that the items were ordered by weight in the finished order and that the picking time was proportional to the number of items.

This thesis is a continuation of a research project written for the subject "TDT4501-Computer Science, Specialization Project" in the second to last semester of my masters at NTNU. In the project, I started to investigate how dividing a consignment using different approaches would affect the performance of Grab. The findings from the project have been improved upon and expanded in this thesis. The overall goal of this thesis is to gather information on what might be the best way forward if Solwr wants to divide a consignment in a way that would improve Grab's performance compared to today's solution. To figure this out, the following research questions have been defined.

Research Question 1: Given the baseline heuristic Next-Fit for dividing a consignment, how does a MIP with the goal of minimizing the number of pallets preform compared to the baseline?

Research Question 2: Given the baseline heuristic Next-Fit for dividing a consignment, how does a heuristic method like simulated annealing with the

cost function of minimizing the number of pallets preform compared to the baseline?

Research Question 3: What changes can be observed if the cost function for simulated annealing is expanded to include the goal of minimizing the item height heterogeneity and weight distribution, compared to only using the goal of minimizing the pallets?

Research Question 4: Can Grab's stability calculation be emulated using a machine learning model to create more stable pallets using simulated annealing? And what effects does it have? To test the models and choose the best suited, a data set needs to be defined. To answer this research question, the following must be investigated:

- **Research Question 4.1:** What is a suitable data set to represent the relationship between an order and its stability?
- **Research Question 4.2:** Given the data set in RQ 4.1 what is a suitable model for implementing a stability prediction in the simulated annealing cost function?

1.1 Structure

This thesis is divided into six chapters. Chapter 1, the current chapter, introduces the problem motivating this thesis and the research questions which will be explored. Chapter 2 presents the relevant theory for the implementations which will be presented in chapter 3. The results from running the implementations are gathered in chapter 4. These are then discussed in chapter 5 along with ideas of future work, before concluding in chapter 6.

1.2 Related Work

The pallet loading problem has been vastly studied throughout the years given the relevance to the logistics industry. Where E. E. Bischoff and M. S. W. Ratcliff were pioneers in the field in the late nineties [5], [6], [7]. In more modern times the problem of using machine learning in order to solve the pallet loading problem has been studied by Batin Latif Aylak et al. [8] but does not seem to be a huge focus in the field. The problem of dividing order without trying

to create a packing pattern first seemed to be little research on. I was unable to find relevant papers discussing it. The idea of using simulated annealing in order to solve the bin packing problem has been explored by R. L. Rao AND S. S. Iyengar and Kämpke, T. [9] [10].

Chapter 2

Background

In this chapter the relevant theory for understanding the implementations and their results is presented. Starting off with discussing the bin packing problem and some relevant heuristics to solve it. How the pallet loading problem can be seen as a variant of it is also discussed in section 2.1. The robot introduced in the introduction, Grab, is discussed in section 2.2. The theory behind the heuristic which will be compared to the baseline method is presented in section 2.3. Lastly the theme of regression, along with the theory behind the relevant models that are explored later in the thesis, is discussed in section 2.4.

2.1 Bin Packing and Pallet Loading Problem

The one-dimensional bin packing problem is well known in the field of combinatorial optimization problems. The classical bin packing problem is defined by an infinite supply of bins with a capacity C and a list of items $L = a_1, ..., a_n$. Each item is given a value s_i which indicates the size of the item a_i, $s_i \equiv s(a_i)$. The value given to s_i has to satisfy the criteria $0 < s_i \leq C$, $1 \leq i \leq n$. The goal is to pack all the elements in the list into the minimal number of bins, so that the total size of all items packed in each bin does not exceed the capacity C.

The bins can either be opened or closed, where open bins are bins available for items to be assigned to and closed ones are off limits. If a bin contains items, the

current content of the bin is denoted as shown in equation 2.1. It is assumed that bins are opened in the order they are indexed, starting from the lowest index.

$$C(B_j) = \sum_{a_i \in B_j} s_i \tag{2.1}$$

The problem is strongly NP-hard, which has created many heuristics approaches since the problem first was introduced in the thirties [11]. They different approaches can be categories into on-line and off-line. In the on-line algorithms, the algorithm will pack the items sequentially as they appear in the list, without knowing what comes next. An off-line algorithm has the option for preprocessing, reordering and grouping before starting, since it has all the items visible from the start. For the scope of this thesis, only some of the most common on-line algorithms will be explored, namely First Fit and Next Fit.

Starting off with a problem instance, borrowed from Michael T. Goodrich of The Donald Bren School of Information and Computer Sciences [12], where the bins have a capacity of $C = 1$ and the following list of items $L = 0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.1, 0.6$. The optimal number of bins for this problem instance is 4, see figure 2.2.

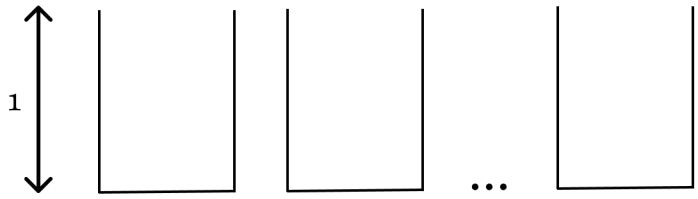


Figure 2.1: The starting point for the problem instance, with the unallocated list L

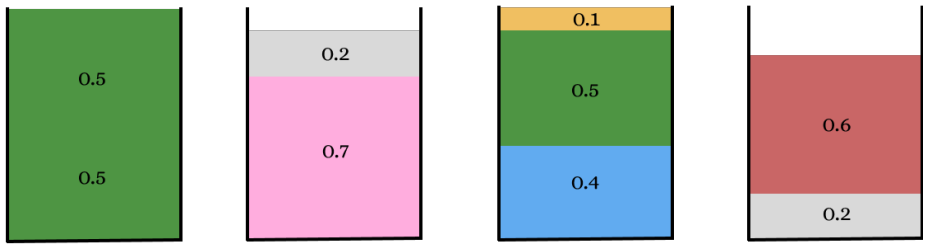


Figure 2.2: The optimal allocation of the items in the problem instance

The Next Fit algorithm will try to fit the current item into the bin that currently is opened, if the bin still has capacity left it will place the item there. If not, it will open a new bin and place it there. Using this method will give us 6 bins to solve the problem instance. How it places the items is illustrated in figure 2.3.

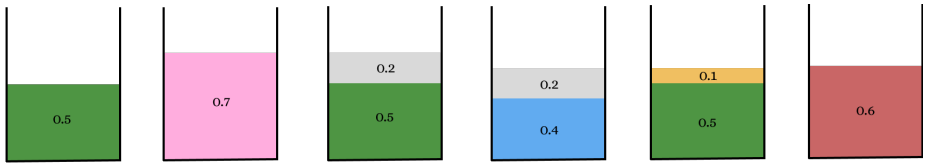


Figure 2.3: The final distribution of items among the bins after running the Next Fit algorithm

The Next Fit algorithm guarantees that it never uses more than 2 times the optimal number of bins needed to pack the items. The following theorem and proof was borrowed from Subhash Suri, a professor from University of California, Santa Barbara [13].

Theorem 1. *If M is the number of bins in the optimal solution, then Next Fit never uses more than $2M$ bins. There exist sequences that force Next Fit to use $2M - 2$ bins.*

Proof. Consider any two adjacent bins. The sum of items in these two bins must be 1; otherwise, Next Fit would have put all the items of the second bin into the first. Thus, total occupied space in $B1 + B2$ is > 1. The same holds

for $B3 + B4$ etc. Thus, at most half the space is wasted, and so Next Fit uses at most $2M$ bins. □

Moving on to the First Fit algorithm, which scans the bins in order and places the current item in the first bin with available capacity. It only opens a new bin when an item does not fit in the previously opened bins. Using this algorithm gives us 5 bins. Figure 2.4 shows how the algorithm distributes the items between the 5 bins. The First Fit algorithm guarantees that the solution is no worse than 17/10 of the optimal solution, [14]. The proof for this is complicated and has been omitted in the papers viewed discussing it. It has therefore been omitted here.

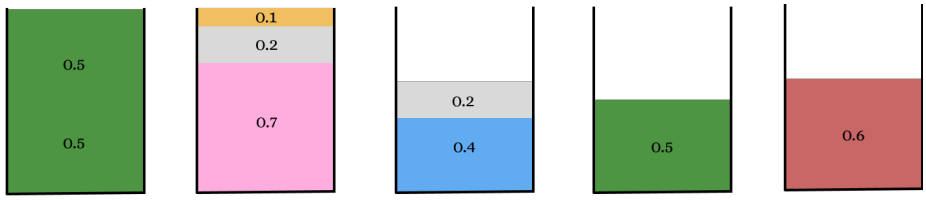


Figure 2.4: The final distribution of items among the bins after running the First Fit algorithm

Bin packing can be modified to suit multiple real-life problems. In this thesis, the problem of packing items onto pallets can be modeled as a bin packing problem. In this problem instance, the value of the items indicates their volume, and a bin has a capacity of C volume. The bin represents the pallet which the items will be placed on. The capacity of the bin is based on the dimensions of the pallet and its max allowed height. To calculate the how much of the capacity of a bin that is used, one just sums up the volume of all items in the bin, or more accurately, on the pallet. This is known as the pallet loading problem in the literature.

Building upon the concept of packing items into bins or pallets, the process of transporting goods involves additional stages, such as stacking pallets into containers. These stages can be viewed as the process of packing three-dimensional items onto larger, rectangular objects, commonly referred to as containers. The objective is to optimize a given objective function, which may vary depending on the specific use-case [15]. As previously noted, in this thesis, we focus on

packing items onto pallets, where the items exhibit varying dimensions, resulting in a strongly heterogeneous data set. This problem instance is known as the distributor's pallet loading problem (DPLP), which is a generalization of the well-known bin packing problem (BPP) [16]. Like the bin packing problem, the DPLP is strongly NP-hard. For a solution to the DPLP to be suitable, it must adhere to the hard constraints set. These can vary depending on the problem instance, but for this thesis they can be divided into pallet-related, item-related, and load-related constraints. The pallet related constraints concern the limitations enforced by the pallet type used, such as the weight limit and height limit. The item related constraints can tackle the limitation of what items can be stacked on top of each other or orientation limitations. Lastly, load related constraints tackle constraints related to the items stacked on the pallet, such as the total stability of a fully stacked pallet [15].

2.2 Grab

Grab was engineered to aid distributors in the new age of logistics. Grab is an automated guided vehicle, which allows it to drive around the warehouse without the need of assistance. It is equipped with a robotic arm with a vacuum gripper allowing it to pick up items weighing up to 30kg. It uses 3D cameras as well as sensors to be able to place items on the pallet. An illustration of Grab is shown in figure 2.5.

The goal of Grab is to allow for distribution centers to either have the robot work alongside the human workers, or replace the need for human workers entirely, depending on the demand from the distribution centers.

Orders sent to Grab are stacked onto the pallet in the same order they are listed in, which makes the ordering of the order important. For this thesis we will assume that the input is optimized so that the order is sorted by weight. We also assume that the picking time is proportional to number of items, which means no penalty for driving distance. The items would be ordered after where they were in the distribution center. This would add the complexity of route planning to find the fastest route, while also making the order to pick a good fit.

To stack orders Grab is equipped with a heuristic for solving the pallet loading problem. This heuristic is based in a beam search algorithm, which takes in a list of items, where each item includes their height, width, depth, and weight.

The output is then a mapping of where each item will be placed on the pallet. The algorithm is treated as a black box algorithm and will not be discussed further in this thesis.

Compacity Compacity is Solwr's measure on how efficiently the stack is utilizing the available space on the pallet. This is calculated as the ratio of the total volume occupied by the boxes in the stack to the total volume of the space available on the pallet. This is given in percentages, where 100% is fully utilizing the available space. Solwr considers pallets with a compacity of 80% to be ideal, since these pallets have shown to hold up the best when trying to pack in real life.

Stability Each stack is given a stability measure, which tells us how well it will withstand an acceleration in the worst direction. Solwr has a cutoff at 1.73, where orders yielding a lower stability are deemed as non-stable since Grab cannot stack them. The orders that are deemed unstable are not given a stability number of a compacity percentage, Grab's algorithm discards it.

Figure 2.5: An illustration of Grab loading items onto pallet

2.3 Simulated Annealing

Many large-scale combinatorial optimization problems can only be solved approximately, since many of these problems are NP-hard. For this, one uses a heuristic, which is an approximation algorithm. This is when the algorithm cannot guarantee the optimal solution but can guarantee that it is completed in polynomial time. Simulated annealing falls under this category of algorithm, more specifically, it is a general optimization technique for solving combinatorial optimization problems. Simulated annealing provides a probability of escaping local minima; however this does not guarantee finding the optimum solution. Simulated annealing is a non-deterministic process, which means that each run will create a different outcome for the same problem. Simulated annealing was introduced by Kirkpatrick et al. and Cerny independently in the 1980s [17] [18].

2.3.1 Iterative Improvement

Simulated annealing builds on the idea of iterative improvement, where we try to perturb some existing suboptimal solution in the direction of a better, lower cost solution. This cost is measured by a set of parameters, which we want to minimize. This function is usually referred to as the cost or objective function. It is how we measure the quality of a solution. Iterative improvements typically start with a random initial configuration, or a heuristically constructed solution, depending on which of those might be less computationally expensive. To improve upon this solution, we attempt some small random perturbation to the solution to produce a new nearby solution. This process is repeated until no further improvements can be made, which is when the process terminates. The main problem with using iterative improvement is that the solution is easily stuck in local minima, since the solution looks good in some small neighborhood of the cost surface but is not necessarily the global optimum. Simulated annealing is like this method but has one major improvement, it allows perturbations to move uphill in a controlled fashion. This gives simulated annealing the same generality as iterative improvement, while offering a solution for the problem of getting stuck in local minima. Summarized, simulated annealing can be described as a generalization of iterative improvement in that it accepts, with non-zero but gradually decreasing probability, deterioration in the cost function.

2.3.2 An Analogy

Simulated annealing uses the analogy of statistical mechanics of annealing in solids to that of approximating a solution for a combinatorial optimization problem. In the annealing of solids, a solid is converted to a low energy state, which means it is in a highly ordered state, for example a crystal lattice. To do this, the material is annealed, which is the process of heating the solid to a high enough temperature to allow for atomic rearrangement in the material. Then cooling it slowly, allowing it to come to a temporal equilibrium at each temperature to bring it into a good crystal. If the cooling happens too rapidly, and the solid does not reach temperate equilibrium at each temperature, it can cause defects to be frozen into the solid. This can result in a metastable amorphous structure instead of the low energy crystalline lattice structure. Simulated annealing uses the same concept on the optimization problem to bring a poor unordered solution into a highly optimized desirable solution. In the literature the same process has been referred to by other names such as Monte Carlo annealing, statistical cooling, probabilistic hill climbing to name a few.

2.3.3 Metropolis Criteria

The idea, similarly, to iterative improvement, is to propose some random perturbation, much like moving one particle to a new location, then evaluating the resulting change in energy. If the energy is reduced, the perturbation is accepted as the new starting point for making the next perturbation. If the energy is increased, unlike iterative improvement, it may still be accepted. This also happens in physical annealing, but the current temperature controls if a worst perturbation is accepted. At higher temperatures, this probability is high, but gets smaller as the temperature decreases. The metropolis criteria models this with the Boltzmann distribution [19]. This gives the probability of an uphill move to be:

$$P(accept) = \begin{cases} e^{-\Delta E/T} & \text{if } E_j > E_i \\ 1 & \text{otherwise} \end{cases} \tag{2.2}$$

In practice, the probability is given by generating a uniform random number R in $[0, 1]$ and comparing it with $P(accept)$, where the one giving the higher number is chosen. By successively lowering the temperature while applying the metropolis criteria, the process of a material coming to equilibrium at each temperature can be simulated. The temperature is just a control parameter,

but in combination with a cooling schedule, which is a sequence of decreasing temperatures, it allows for the modification of the acceptance of uphill moves over the course of the solution. The starting temperature must be high enough to allow for essentially a random search of the configuration space. This means that at higher temperatures the number of allowed uphill moves is higher, and this decreases with the temperature, just as with physical annealing. At lower temperatures the system is close to freezing into the final form, so few disruptive uphill moves are allowed. This makes the system essentially just standard iterative improvement.

2.3.4 Requirments for implementation

The requirement for simulating the process of annealing is the ability to simulate how the system reaches temporal equilibrium at each fixed temperature. The following four formulations have to be defined.

1. **Configurations:** a model of what a legal configuration is. These represent the possible solutions to the optimization problem. This is the space in which we will search for the solution in.
2. **Move set:** A set of allowed moves to go from one configuration to another in order to reach a feasible solution. These should be easily computed.
3. **Cost function:** A measure of how good a configuration is. This has to return a value which can be compared to another configuration in order to determine the change in "energy".
4. **Cooling schedule:** This includes a starting temperature or a heuristic to calculate it. Rules for when the temperature should be lowered, and how much it should be lowered, and when the annealing process should be terminated.
   (a) One cooling schedule proposed by Kirkpatrick is one called geometric, [20]. This is given by the formula 2.3. The best suited cooling schedule is determined empirically for best results.

$$T_{new} = \alpha \cdot T_{old}, \quad \alpha < 0 \tag{2.3}$$

2.4 Regression

Regression is a form of supervised learning used to model continuous variables and make predictions. In the context of stability prediction, we aim to predict the stability of orders by considering multiple different characteristics that provide valuable information. This multivariate regression problem involves examining various order attributes to achieve a comprehensive and accurate prediction. Given a set of parameters $X = X_1, X_2, ..., X_n$, where X_i represents the value of the i-th parameter, and target values $Y = Y_1, Y_2, ..., Y_n$, our goal is to find an expression that reduces the difference between the predicted values $\hat{Y}$ and the target values Y to a minimum. The data set comprises n observations, where we want to find the best possible approximation of the target values based on the given parameters and their corresponding values. It is also important to keep in mind the trade-off between performance and generality, where models can become overfitted which gives good results when training but lacks the generality when introducing a new problem which might yield bad results. We also don't want the model to become too general where the performance tanks and the results are given by it is unusable, also known as underfitting. In this thesis some models are especially relevant, the theory surrounding them will be presented in this chapter.

2.4.1 Decision Trees

A decision tree, as the name suggests, is a tree that is constructed for determining the prediction of new data. A decision tree can be either a binary tree, which only has at most two children per node, or a multiway tree, which can have more than two children. In this thesis, when we are discussing a decision tree, we are specifically referring to a binary tree. The process of finding the optimal partitioning of the data is NP-complete, so the most common way to construct a decision tree is by recursively partitioning the input space into a tree structure until only leaf nodes remain [21]. An example of how a tree might look for a regression problem is shown in figure 2.6.

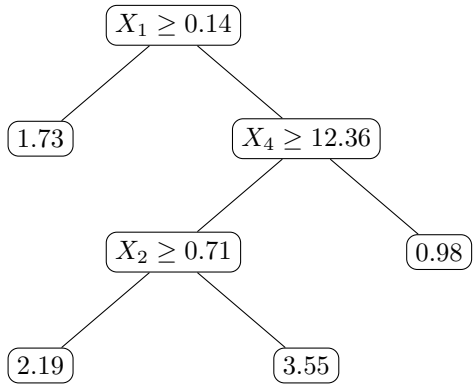


Figure 2.6: An example on how a regression decision tree might look

When passing unseen data through the tree, each internal node contains a question. In regression, this question typically compares a feature value with a value set in the question. Depending on the result of this comparison, the input passes through the tree until it reaches a leaf node. The leaf node determines the value that the prediction will receive.

One might wonder why specific questions are chosen to be turned into nodes when creating the tree. This comes down to what questions have the most information gain, or what some literature refers to as lowering the impurity. In regression, this is measured by variance, specifically the variance reduction. This is calculated for each potential split, and the one yielding the lowest variance is chosen. The higher the variance, the higher the impurity. The formula given by equation 2.4 shows how the variance is calculated and equation 2.5 shows how the variance reduction is calculated.

$$Var(X) = \frac{1}{n}\sum_{i=1}^{I}(y_i - \hat{y})^2 \tag{2.4}$$

$$Var_{reduction} = Var(parent) - \sum_{i=1}^{I} w_i \cdot Var(child_i) \tag{2.5}$$

When the tree is created, the problem of overfitting can arise since the tree structure can become complicated. One solution to this is to prune the tree.

Pruning can be done during the process of creating the tree (pre-pruning) or after having a fully grown tree (post-pruning). Zhou explain these pruning methods as follows: "Pre-pruning evaluates the improvement of the generalization ability of each split and cancels a split if the improvement is small, i.e., the node is marked as a leaf node. In contrast, post-pruning re-examines the non-leaf nodes of a fully grown decision tree, and a node is replaced with a leaf node if the replacement leads to improved generalization ability." [22]

The benefit of using decision trees is that they are easy to understand, relatively robust to outliers, and scale well to large data sets. Their drawbacks are their instability. The hierarchical nature of the tree-growing process means that small changes in the input data can have a significant impact on the structure of the tree, as errors higher up in the tree can propagate down and affect the rest of the tree. To overcome the challenge of unstable trees other methods of using decision trees have been created, these will be explored further in the sections below.

Random Forest

One method that aims to address the issue of unstable trees is the random forest. The random forest algorithm involves creating multiple decision trees and combining their predictions through averaging, see equation 2.6. M is the total number of trees in the forest and $y_m(x)$ is the prediction made by the m'th tree. By aggregating the predictions of multiple trees, the random forest approach can compensate for the potential shortcomings or errors of individual trees within the forest. To prevent the trees in the forest from becoming highly correlated, which could limit the potential variance reduction, they are built or learned based on a randomly chosen subset of input variables, as well as a randomly chosen subset of data cases. The latter is known as bootstrapping.

By incorporating randomness in feature selection and data sampling, random forests can build a collection of diverse and independent decision trees. This diversity enables the forest to capture different aspects of the underlying data patterns and make more robust predictions. Additionally, the averaging of the predictions helps to reduce the impact of individual tree errors and enhance the overall accuracy and stability of the model [21].

A benefit of this method is that despite being easy to implement as well as computationally low in cost, it still yields good performance in real life scenarios. It does however lose some of the benefit of being easy to interoperate, since a

forest can become quite large.

$$\hat{y}(x) = \sum_{m=1}^{M} \frac{1}{M} \cdot y_m(x) \tag{2.6}$$

Boosting Trees

While random forests build multiple decision trees independently and combine their predictions, boosting trees construct the trees sequentially, using the knowledge of the performance of the previous trees to improve upon the next tree. More specifically, the idea is to take weak learners, which in the case of boosting trees is a decision tree and apply them sequentially to a weighted version of data, where more weight is given based on the performance of the learners [21].

At the start all the weighted coefficients are given the same value, and after an iteration the weights are adjusted to give data points with higher loss greater weight and decrease data points where the loss was lower. This forces the next models to put greater emphasis on the weaker data points. This sequential nature allows boosting algorithms to iteratively learn from mistakes and improve the overall performance of the ensemble. After all the trees are trained, they are combined to form a committee using coefficients that give different weight to different trees [23].

$$y(x) = \sum_{m=1}^{M} \alpha_m \cdot y_m(x) \tag{2.7}$$

Here, $y_m(x)$ represents the prediction made by the m'th boosting tree, α_m is the weight assigned to the m'th tree's prediction, and M is the total number of boosting trees in the ensemble. The final prediction $y(x)$ is obtained by summing up the weighted predictions of all the trees.

Boosted trees have the same problem as random forest, where the number of trees compared to only one makes the models hard to interoperate. Given that the focus is to achieve a high accuracy, this doesn't necessarily have to be an issue.

2.4.2 KNN

The KNN, which is short for K-nearest neighbor, is a technique for making predictions based on the values of the k-nearest neighbors. When used in regression, a prediction is typically made by taking a weighted average of the values of the K-nearest points. To determine the distance of a point, most methods use Euclidean distance, which calculates the distance of two points p and q in an n-dimensional space using equation 2.8. In the equation, p_i and q_i represent the coordinates of the points p and q in the i-th dimension.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \tag{2.8}$$

When training the model, it stores the feature vectors and corresponding target values of the training data set, this is the points used when making new predictions. In this method, the choice of K is important since it can impact the performance of the model. A small value of K will make the prediction sensitive to noise or outliers in the data (overfits), while a large value of K may lead to over smoothing and loss of local details (underfits) [21].

2.4.3 Evaluation of a Model

The goal when utilizing machine learning is to retrieve a generalized function $\hat{y}$ fitting the problem domain. When dealing with a prediction problem, where we have a data set $D = (x_1, y_1), (x_2, y_2)...(x_n, y_n)$ where y_i is the actual value for x, we need to compare $\hat{y}(x_i)$ to the value given in y_i in order to measure performance. The goal is that the model will do well on data it has not seen while training, in other words the difference between y_i and the prediction for y_i is small, meaning the loss is low. This value can be calculated in various ways, depending on what fits our data the best. For a regression problem, the most common loss functions are called Root mean square Error and Mean squared error, see equation 2.9 and 2.10 [22].

Root mean square Error

$$RMSE(y) = \sqrt{\sum_{i=1}^{N} \frac{|\hat{y}(x_i) - y_i|}{N}} \tag{2.9}$$

Mean squared error

$$MSE(y) = \sum_{i=1}^{N} \frac{\hat{y}(x_i) - y_i^2}{N} \tag{2.10}$$

Chapter 3

Implementations

In this chapter, the different implementation needed to answer the research question in the introduction will be presented. Starting off, the data provided by Solwr will be presented in section 3.1. The data is a crucial part of the thesis, since it lays the foundation for accurately representing the real-world orders which Grab can be expected to encounter. The constraints used in the implementations are presented in section 3.2. The implementation of the baseline approach for dividing a consignment, the Next Fit algorithm, will be presented in section 3.3. This will be used to compare performance of alternative approaches. These alternative approaches will also be presented in this chapter, which are a MIP in section 3.4 and the heuristic simulated annealing in section 3.5, both with the goal of minimizing the total pallets needed to pack a consignment. Simulated annealing and its cost function will be expanded to conduct the experiments needed to answer research question 3 and 4. Some of the information presented in this chapter is derived from the research project leading up to this thesis, it is necessary to present them considering the new discoveries and modifications made since its conclusion.

3.1 The Data

At the beginning of the research project, Solwr provided 100 MB worth of orders, collected over four separate months. This amounted to a total number of 814 525 items going to 628 different stores. Given the run time of Grab's algorithm,

it was unrealistic to use all the data, when conducting the experiments which are relevant for the first three research questions presented. The data was then reduced to include only a month's worth of orders. This resulted in 7.3 MB worth of workable data, which is a considerable reduction. This contained a total of 65 200 items which were going to 326 different stores, making up 865 different orders. In this context, when discussing an order, I am referring to a collection of items a store has ordered, where they have not yet been divided further to reflect the term I use when talking about orders in other parts of the thesis. Since I defined a consignment as a collection of items going to one store, I found the term inaccurate to use here, since multiple orders might be going to the same store.

During the research project, the orders were not filtered furthered, but the findings after the conclusion of the project showed that some of the orders consisted of few enough items so that they would fit onto one pallet regardless of what approach was used. These orders did not contribute to what this thesis sets out to investigate and created noise in the results, making it harder to see the effects of the different approaches tested on the data. They were therefore removed when the thesis started up again after the research project. Instead, the data was filtered so only orders which exceeded the max volume of five pallets would be included. This furthered reduces the data to include 53 646 items which were distributed over 68 stores. The reasoning for using five pallets, was to ensure larger enough consignments in the hope that it would highlight the differences between the approaches.

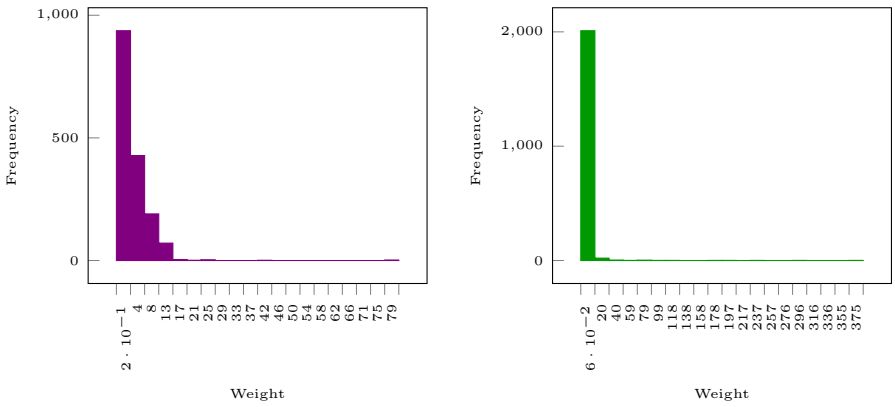


Figure 3.1: Distribution of weight of boxes in the filtered data used for the experiments and the original data

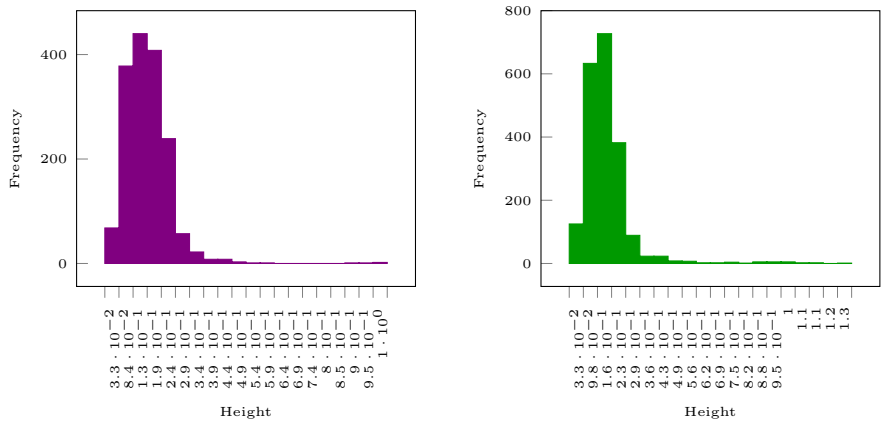


Figure 3.2: Distribution of height of boxes in the filtered data used for the experiments and the original data

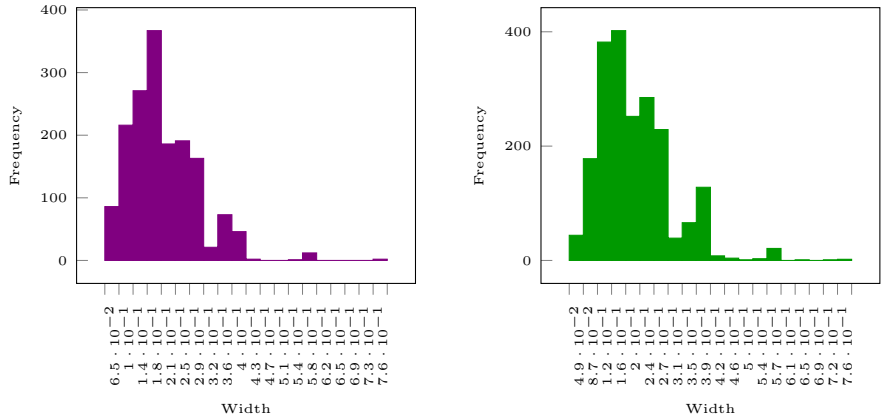


Figure 3.3: Distribution of width of boxes in the filtered data used for the experiments and the original data

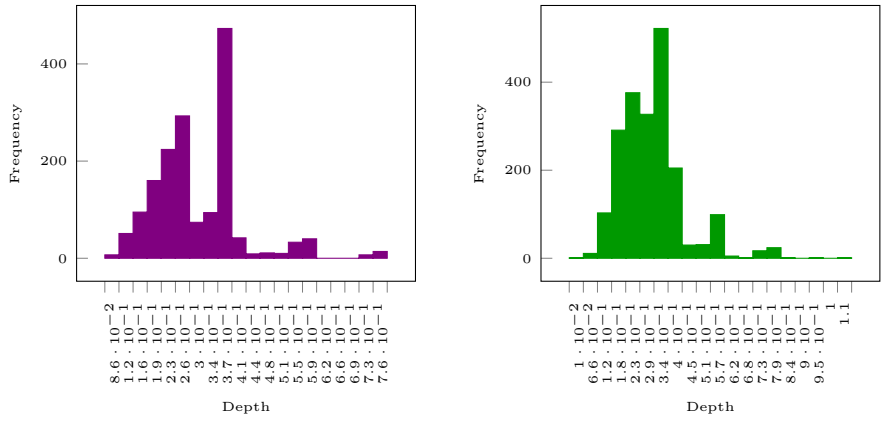


Figure 3.4: Distribution of depth of boxes in the filtered data used for the experiments and the original data

In order to illustrate how well the new data pool, represent the data in its entirety, the dimensions and weight of the items have been plotted. From the figures 3.1, 3.2, 3.3 and 3.4, we can see how the item's dimensions and weight

are distributed. All the plots have the same bin size of 20, which was the size that was able to capture some of the nuances while still being easy to read. From figure 3.1, which shows the distribution of the weight of items in the data, that most items fall between 0.1 kg to 20 kg, where more outliers are present in the entirety of the data. This is caused by the filtering of the orders, since orders which fit onto less than five pallets can consist of heavier items. Solwr has registered a half pallet of beer as a single item, which as the name suggests is half a pallet worth of beer. This might be registered under a single order, which causes it to be filtered out. When it comes to the different dimensions, it appears that the data selected represents the overall data well. Where the biggest differences in the different dimension's histograms are the frequency of the different dimensions, which is natural since all the data contains more boxes.

3.2 Constraints

As mentioned in section 2.1 when working with a pallet loading problem, you define different constraints for the problem at hand. To investigate the problem presented by Solwr the following constraints were defined.

Pallet-Related Constraints The Euro Pallet, produced by the European Pallet Association, hereby referred to as EPA, is used as the standard pallet through the experiments conducted in this thesis. This decision was made given that the pallet is the most widely used exchange pallet in the world [24]. It also is the same pallet dimensions used by Solwr in some of the code I have reviewed. Solwr expressed that they wanted the possibility to change the pallet type when needed, so the code developed for the experiments makes it easy to exchange the pallet dimensions if desirable.

The EPA supports a loading volume in the x- and y-axis of $1200mm \times 800mm$, when determining the limit in the y-axis, or the height, the constraint is depended on the max weight and the dimensions of the container which will be used to transport the pallet. The EPA pallet has a max workload of 1500 kg, but in discussions with Solwr the weight limit was to be ignored, since we wanted to encourage heavier pallets. The max height ended up being set 1100 mm after discussions with Solwr. This gave the total volume allowed on the pallet to be $1200mm \times 800mm \times 1100mm$

Item-Related Constraints The boxes can only be rotated horizontally since Grab only can access them from the top with its suction arm and rotate them around the z-axis. Given that in this thesis the packing pattern will not be generated, this is not relevant to the methods developed, but is included to give context for the reader.

It is also assumed that the boxes are placed on the pallet from heaviest to lightest. In a distribution center, Grab will grab the boxes in the order they are placed in the warehouse, but we have already established that the order of the items are optimal when passing it to Grab, i.e. sorted by weight when discussing the assumptions of this thesis.

Load-Related Constraints When it comes to what makes a stacked pallet acceptable for shipping, Grab provides measures to determine this. As mentioned in section 2.2, Grab requires the pallets to have a stability of 1.73 or higher when creating a packing pattern. If Grab fails to find a packing pattern for the order that satisfies this, it is disregarded and deemed unstable. Grab's algorithm also provides a measure of how compactly an order is stacked. In discussions with Solwr they stated that a compacity of 80% or more is desirable. However, if a pattern falls below this, Grab does not disregard it, as it does with the stability. The load also must adhere to the constraints of the pallet. This means the total load volume must fit inside the max volume presented in the pallet-related constraints. To implement a machine learning model, which is discussed in detail later in this chapter, a new loaded related constraint was added. This made the max number of items allowed on a pallet be 60. This is only relevant when employing the machine learning model to partition the consignments.

3.3 Next-Fit Heuristic

To investigate the effect an approach of partitioning a consignment had, a baseline needed to be established. The development of it started with what I, in discussions with my supervisor, thought would produce a workable but not optimal division. The idea was to start off with shuffling all the items going to a store and then dividing it sequentially. This shuffling process aimed to create a more diverse arrangement of items on the pallets. The rationale behind shuffling the items was to mitigate the potential issue of some pallets having mostly, if not all, the same box type, resulting in a significant gap in heterogeneity between

pallets. By shuffling the items, the algorithm intended to encourage pallets to have a similar degree of heterogeneity. However, this shuffling step was later dropped in the final version of the algorithm due to its non-deterministic nature, which increased the time required for debugging and testing. Although it is possible to achieve reproducibility by providing a random seed and still shuffling the items, this aspect was not prioritized as the focus of this thesis did not revolve around the specific effects of shuffling.

The heuristic begins with one pallet and iterates over the list of the items, assigning as many items as possible to the pallet until the volume capacity is met. Once the capacity is reached, the pallet is closed and included in the result, which will be passed to Grab's algorithm. This process continues until all items in the consignment are allocated to pallets. The implementation used in can be viewed in appendix B.1.

As discussed in chapter 2.1, the problem of minimizing the number of pallets required to pack a consignment can be viewed as an abstraction of the well-known bin packing problem, where the items are considered based on volume rather than weight. The heuristic explained above might look familiar, since it is the same as the algorithm presented in section 2.1, which is commonly known as the 'Next-Fit' algorithm in the field of bin packing.

3.4 Mixed Integer Program

The first method which is going to be compared to the baseline method presented in the previous chapter is a mixed integer program (MIP) using the Gurobi solver. Gurobi was chosen based on recommendations made by a fellow student specializing in optimization. The solver is not freely accessible, it was made available through the university. To solve the problem using Gurobi, a mathematical model was formulated.

Starting off with the relevant sets and indices needed to be defined to set up the solver. The items are categorized by type, which is denoted by i. The index of a pallet is denoted by j, see table 3.1.

To know how many items of each type we can fit on a pallet, we need to know the dimensions of the items, which is the volume of the item. The item of type i's volume is denoted by D_i. In this approach, the weight is also a parameter we consider, so this is stored as well, denoted by W_i. We also create a parameter to denote the quantity available of an item of type i, n_i. As previously mentioned,

pallets are of the same type, where each pallet a max volume allowed denoted by K_j^D and a weight limit denoted by K_j^W. All the parameters are summarized in table 3.2.

The final partition is given by the matrix x, where x_{ij} notes how many items of type i are going on pallet j. We also get a list y where the indices of the pallets used are given the value 1, if not they are given 0, see table 3.3.

The optimization goal of this model is to minimize the number of pallets needed to pack a consignment. This is expressed in equation 1. The model implements the constraints discussed previously, starting off with ensuring that all the items in the consignment are assigned to a pallet, 2. The model also must adhere to the constraint that no pallet can be loaded with a load exceeding its capacity, both in terms of volume and weight, see equation 3 and 4. To avoid extensive computation and improve computational efficiency, a symmetry breaking constraint was implemented in the form of equation 5. This constraint ensures that the pallets are opened sequentially, introducing a preference for solutions where the used pallets are assigned lower indices compared to the unused pallets. By doing so, it breaks the symmetry between multiple solutions that have the same number of bins used but differ in the order or index of the pallets used. Without the symmetry breaking constraints, the model would explore all possible arrangements of the pallets, including symmetrically equivalent solutions. These symmetric solutions only differ in the ordering or index of the pallets used, leading to redundant computations and an exponential increase in the search space. The equation 6 enforces the model to use integer values from x_{ij} and binary values for y_j. The implementation used is shown in appendix A.1.

Sets And Indices

i:	Item Type
j:	Pallet

Table 3.1: Sets and Indices

Parameters

D_i :	Item Dimension
W_i :	Item Weight
n_i :	Nr of items of type i
K_j^D :	Volume capacity if container j
K_j^W :	Weight limit of container j

Table 3.2: Parameters

Variables

x_{ij} = number of items of type i on pallet j

$$y_j = \begin{cases} 1, \text{if pallet j is used} \\ 0, \text{otherwise} \end{cases}$$

Table 3.3: Variables

$$\text{minimize} \quad \sum_{j \in M} y_j \tag{1}$$

$$\text{subject to} \quad \sum_{j \in M} x_{ij} = n_i, \qquad i \in N, \tag{2}$$

$$\sum_{i \in N} D_i x_{ij} \leq K_j^D y_i \qquad j \in M, \tag{3}$$

$$\sum_{i \in N} W_i X_{ij} \leq K_j^W y_i \qquad j \in M, \tag{4}$$

$$y_j \geq y_{j+1} \qquad j \in M/\{|M|\}, \tag{5}$$

$$x_{ij} \in \mathbb{Z}^+, y_j \in \{0, 1\} \tag{6}$$

3.5 Simulated Annealing

To be able to use simulated annealing, the four requirements presented in section 2.3 needs to be defined for the problem definition we are investigating in this thesis. To summarize, we need to know what configuration to use in order to tell what a valid state of a solution of the problem is. The move set, which

expresses the moves the system is allowed to take to reach a new state, needs to be defined. The cost function, or optimization goal, which is what we want to optimize, as well as the cooling schedule, which defines how fast the system will be cooled down, needs to be defined. In this section, these requirements will be presented. The code used to define the different requirements for this thesis is placed in the appendix, see appendix C.

The requirements presented lay the foundation for the simulated annealing used in this thesis, where some modification needed to be made to conduct the needed experiment to answer some of the research questions. Namely changing up the cost function.

3.5.1 Configurations

The problem formulation varies depending on the cost function we want to use, but the configuration stays mostly the same throughout. To do so, we need to represent what is a valid solution for simulated annealing. The configuration related to the problem statement is a collection of what I have chosen to call packed pallets. Despite the name, a packed pallet is categorized as a collection of items that are assigned to fit onto a single pallet, rather than the packing pattern of how the boxes will be stacked on the pallet. The packing pattern is generated by Grab's algorithm. A packed pallet contains as mentioned the information on what items Grab will be stacking on one pallet. The information about the total volume of the items being packed onto the pallet, as well as the weight, is also stored in a single packed pallet. In order to visualize this figure 3.5 was created. On the left side you can see the pallet type and the collection of items, each with a given volume and weight, which we want to pack onto the pallet. This configuration is then fed through Grab's algorithm and Grab returns the packing pattern of the configuration as seen to the right of the arrow in the figure.

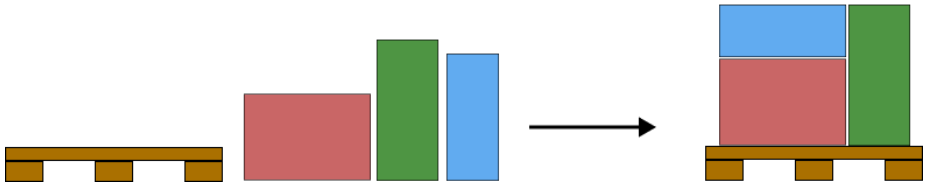


Figure 3.5: A single pallet in the configuration

Since we are dealing with a consignment going to a store, we need multiple of these packed pallets to represent a solution to the problem as a whole. As previously established, the pallet type is consistent throughout the problem, which means that all the packed pallets created will be packed onto the same pallet type. So in order for a solution to be a valid configuration, it has to have assigned all the items to a pallet, where each pallet confides to the constraints presented in section 3.2.

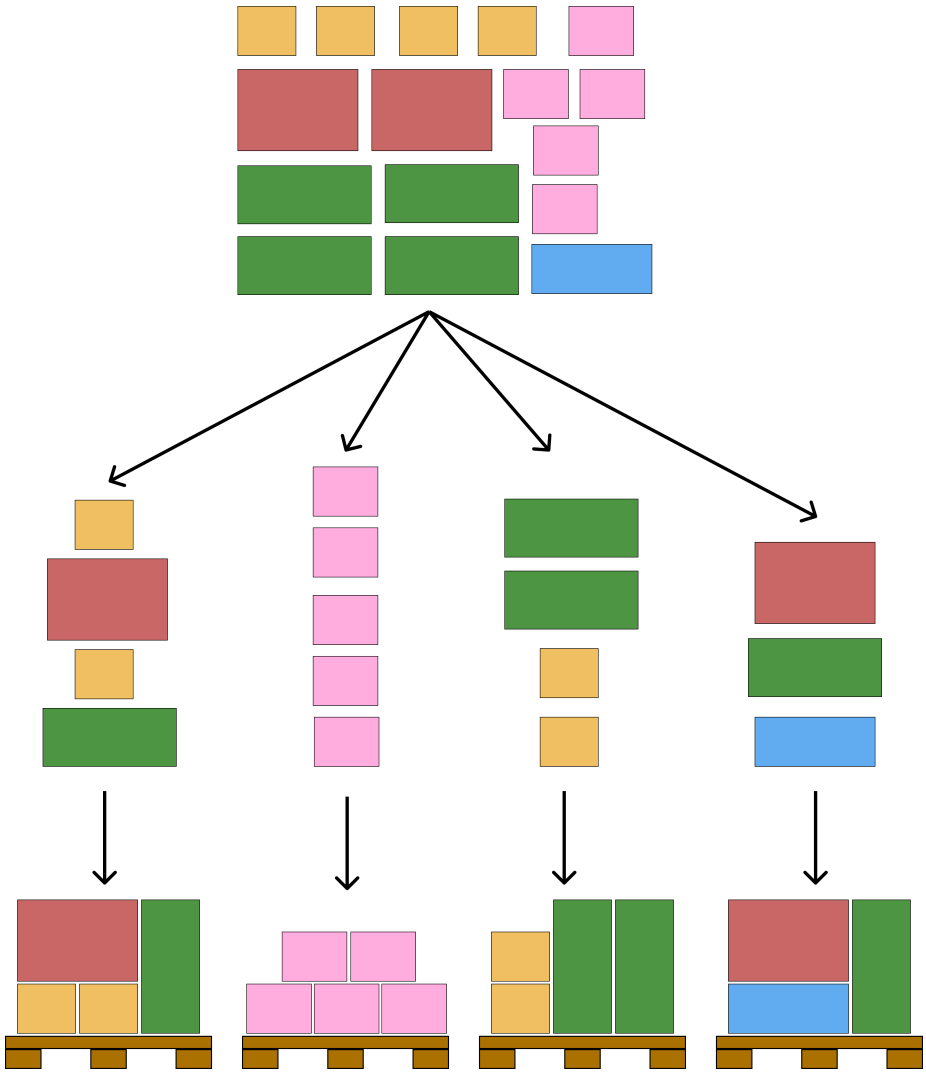


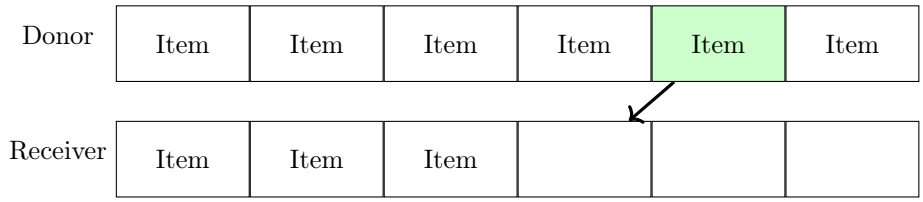
Figure 3.6: Illustration of how a configuration might look

In figure 3.6 we can see the whole process from taking a consignment and splitting it and assigning the items to multiple pallets. These are then fed to Grab

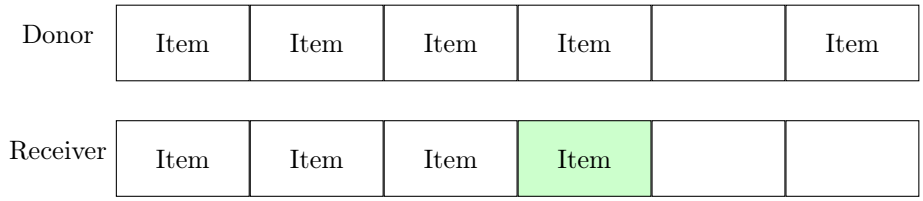
which returns the packing pattern along with the stability and compacity for the pallets if they proved to be stable enough.

3.5.2 Move set

An essential part of simulated annealing is how we move from one configuration to the next. For the changes between neighboring states to be small, a move set where only one box is moved between two pallets was selected. The pallets that would perform the swap were chosen at random, with the condition of having different indices to ensure no pallet would try to give or take an item to itself. Once the pallets were selected, the first item from the list of the chosen pallet, or the donor, would be examined to determine if adding it would exceed the volume limit of the receiving pallet. If the constraints were not broken, the item would be removed from the list of the pallet it was taken from and added to the list of the receiver. If the constraints were broken, the donor would move on to the next item in the list and preform the same test. Should none of the boxes from the donor pass the test, the function for the move set would return the same configuration it started with. The process is illustrated in figure 3.7. In this example, the item laying at index 5 is chosen (Julia starts the index at 1) and tries to give it to the receiver. Since the receiver has space left the item is added, and then removed from the doner. The code used for this can be seen in appendix C.1.



(a) The donor pallet tries to place the randomly chosen item onto the receiving pallet



(b) Since the receiver had room left, the item is transferred from the donor to the receiver

Figure 3.7: A donor pallet trying to give its item to the receiver

3.5.3 Cost Function

To conduct all the experiments needed to answer all the research questions, several cost functions needed to be implemented. They all will be presented in this section, starting off with the optimization goal of minimizing the number of pallets in a configuration.

Minimizing Number of Pallets

The cost function takes in a configuration and gives it a score based on the number of pallets in it, see equation 3.2. This score is then used when comparing two neighboring states, where the acceptance probability is depending on what state that reduces the energy of the system. If a state has a lower number of pallets, the probability of accepting the new state is 100%. If not, the probability of accepting the new state is given by the metropolis criteria presented in section 2.3. The implementation of the calculation of the acceptance criteria and the cost function can be viewed in appendix C.2.

$$E = \text{Number of Pallets} \tag{3.2}$$

Minimizing Number of Pallets, Height Heterogeneity and the Weight Distribution

After the first experiments were conducted, it was of interest to investigate if performance could be improved by expanding the simulated annealing cost function. The cost function was expanded with the suggestions made by Solwr on what an appropriate cost function might be, to see if there were any significant changes to the stability and compacity of the output. The reasoning behind expanding simulated annealing versus the linear model was the level of complexity needed to achieve results. It was easier to formulate and experiment with simulated annealing.

To use the average heterogeneity in the cost function, the structs for packed pallet presented in the section for configuration were altered to include the value over the height heterogeneity of the items. A function that calculates the ratio of unique elements to the total number of elements in the input array, providing a measure of how diverse or homogeneous the array is, was implemented, see appendix C.7. This was then used in a function which takes the list of items on a pallet to determine the heterogeneity of each of the dimensions, see appendix C.8. Despite the heterogeneity of all the dimensions being calculated, only the height heterogeneity is used further. The reason behind this decision was that reducing the variation in height was expected to make it easier for Grab's algorithm to create even layers. These leveled layers would provide greater stability for stacking, ultimately increasing the overall stability of the pallet. This was then used to calculate the average height heterogeneity in a state, which consists of multiple packed pallets, see appendix C.3.

In addition to calculating the height heterogeneity, a measure was needed to represent the distribution of weight in the system. For this, the standard deviation of the weights for the different pallets in a state was used. The thought was that this would encourage the pallets to distribute the weight evenly, and increasing the weight of each pallet Grab would pack. These changes were included in the cost function, see equation 3.3.

$$E = \text{Number of Pallets} \cdot \overline{X}_{Heterogeneity} \cdot \sigma_{weight} \tag{3.3}$$

Maximizing the Stability

When implementing the machine learning model which predicts the stability of pallets, the cost function had to be reworked. In addition, some of the other requirements also had to be altered, to make sure the model did not crash. The process of choosing the model is discussed later in this chapter, only the changes made to incorporate the new cost function will be presented here.

As mentioned, the model will take one pallet and provided it with a stability prediction to emulate the process of running an order through Grab, without having to run Grab's algorithm. The packed pallet struct was expanded further by adding a field for the predicted stability. To assess a state, the collected stability must be evaluated, so a field for the overall stability of a solution was added to the state struct. The goal is no longer to minimize a value, but rather maximize it, since we want the collective stability to be as high as possible. To achieve this, a flag was implemented to change the logic on the calculation of the acceptance criteria. A state would now be accepted with 100% probability if there was an increase in the stability of the state, and the metropolis criteria would be used if the stability was lowered in the new state, see appendix C.2. A helper function was added to sum the individual stability for each pallet, which was used to compare each state's collective stability, see appendix C.6.

To keep the input size for the machine learning model consistent, a max limit was imposed on how many items could be on a pallet. Changes were made to incorporate this constraint in the move set function. As with the acceptance criteria, a flag was added to indicate if the cost function using the model was being employed. How the max input size was determined will be explained in the section 3.7.

Cooling schedule

As a cooling schedule, the geometric cooling schedule presented in section 2.3 was used. Two different cooling schedules were used, one for simulated annealing using the machine learning model and one for the rest. Starting with the one used by most of the methods. At the start of the experiments, I wanted to keep the cooling schedule simple. To achieve this only one iteration per temperature was done. The exact starting temperature was 50000, which was lowered each iteration by $\alpha = 0.998$. This was run through 10000 iterations or epochs. When the number of epochs were reached the search would terminate. The parameters were determined through running a number of different combinations.

For the machine learning model, the cooling schedule was altered to include more rearranging at each temperature. Instead of only doing 1 iteration/epochs it would do 5000 iterations/epochs for each temperature. This was determined by testing different numbers of epochs with a $\alpha = 0.998$ and the starting 50000, which remained the same. I also implemented a different stopping criterion to limit the time the program would use. The new criteria would stop the process if the temperature reach zero. Or if there was no change after three consecutive temperatures.

3.6 Creating A Suitable Data Set

Since the machine learning model is going to be used to predict the stability of real orders, it was important to have a data set that would reflect the orders it would be expected to pack. In section 3.1 I talked about how much of the data was removed to keep the experiments manageable, this data ended up being used for creating the final data set which was used to choose a suitable model. Having a large collection of data is a good thing when creating a data set and can be a major bottleneck when employing machine learning. The data first needed to be labeled, which was done by running it through the stacker. This was a time-consuming process, where some orders even ended up being unstable and therefore did not receive a stability number which could be used. Grab's algorithm is still under development, which means that it might still be containing bugs, which could affect the performance of the model. The version used in this thesis has since received updates, but to keep the results consistent, only the one version was used. Some order / stability pairs were removed due to unexpected stabilities. These were pallets that were given negative stability number, or numbers lower than the threshold set in Grab's code which was considered a bug. This has opened the possibility of there being some faulty pairs that might have gone undetected. In addition to the data previously presented, Solwr provided additional data with the required labeling. The orders that were run through Grab combined with the orders already labeled by Solwr gave a total of 27470 orders that could be used in the making of a data set.

3.6.1 First Iteration

At the start of creating the data set, a meeting was held with a representative from Solwr and SINTEF to brainstorm ideas of how a suitable data set might look. It was important to provide the model with enough information to be

able to make accurate prediction, but challenging to know what attributes of the pallets would provide that. The first iteration ended up using heterogeneity of the different dimensions of the boxes in the order. Since heterogeneity was a recurring attribute when discussing the cost function, it seemed like a good place to start. Along with the heterogeneity, the number of boxes was included. It was simple and easy to implement, since the functions needed to calculate this already existed. It also gave a fixed number of input parameters, allowing for orders with varying number of boxes to be included in the training data. As mentioned previously, the label used along these parameters was the stability number Grab provided.

3.6.2 Second Iteration

The data set was revised and change in the hopes that with more information, the model would preform better. The results from the first iteration of the data set are presented in chapter 4 and will be discussed more in chapter 5. The idea was to give the model as much info as possible, and to give it as much information as we have, giving it all the dimensions for the boxes along with their weight seemed like a good way to go. The problem with this is that many of the orders which built up the first data set had a varying number of boxes. For the model to use the data, it needed a fixed number of input parameters, which meant that passing all orders used in the first data set was challenging. To ensure that every order would have the same number of parameters, the outliers were removed, the remaining order with the most items was then used as the upper bound. Orders which were made up of a smaller number of boxes was padded with zeros.

To determine the outliers an analysis was preformed, here the outliers were removed using the IQR method. This showed that the max number of boxes needed would be 59, since this was the highest number of items in all orders was 59. This number was rounded up to 60. The orders with fewer boxes than the input size was padded with zeros. The last element in the data set was the label, the stability. Figure 3.8 illustrate how the data set is organized.

$$\underbrace{h_1, w_1, d_3, x_1}_{box\ 1}, \cdots, \underbrace{h_{60}, w_{60}, d_{60}, x_{60}}_{box\ 60}, stability \tag{3.4}$$

Figure 3.8: How the boxes are organized in the data set

This reduced the data set from the first iteration drastically from 27470 orders down to 1125 orders. The total number of parameters ended up being 241, since the max number of boxes was 60, and wanting to have the dimensions and weight gives us 4 parameters for a box and then adding the stability at the end.

3.7 Choosing the Model

To make the predictions, various models were tested from Julia's machine learning framework MLJ.jl. The models were chosen using MLJ's model search feature, where the data being used can be passed and several models suited are returned [25]. This returned 63 potential models, upon further testing, where some of the models did produce errors, the number was reduced to 26 models. All the models using the SciKitLearn package had problems running. As a result, all the models from that package were discarded. It was decided to not investigate what might be causing this since there still had 26 models which did work.

To determine the best fit for our problem, all models returned from MLJ's model search were tested out on the first and second iteration of the data set. Since the first data set was changed, given that the error for the models were high, the models were chosen only on their performance on the second data set. The models were trained on 80% of the data set and validated on the remaining 20%. Given the threshold for a stable pallet, it was important for the model to have a low loss. The RMSE and MAE from each model is showed in figure 4.2 and in table 4.11.

From these results, the tree models with the lowest root-mean-square error were chosen. If multiple models had the same result, the mean absolute error was then used to break the ties. This resulted in the EvoTreeGaussian model from the package "EvoTrees" [26], the KNN Regressor model from the package "NearestNeighborModels" [27] and the Random Forest Regressor from the "BetaML" package [28].

The parameters of all the three models were then further tuned to find the best preforming model of them all. This turned out to be the random forest regressor. The best combination of tunable parameters turned out to be a forest size of 30 where each tree had a max depth of 5.

Chapter 4

Results

In this chapter, the results from the experiments conducted to answer the research questions will be presented. In section 4.1 the results of running the baseline heuristic and the MIP are collected in tables to highlight the differences. The same results from the baseline heuristic are then repeated in section 4.2 to illustrate the difference between the simulated annealing using the same optimization goal as the MIP. The results of expanding the simulated annealing cost function are presented in section 4.3. In section 4.4, the results from changing out the cost function for simulated annealing to maximize stability is shown, along with the results from testing the different models and data sets. Lastly, results using all the different method on a single consignment are presented in section 4.5.

The experiments run to produce the results presented in section 4.1, 4.2, and 4.3 were run using the data presented in section 3.1. The different splitting algorithms presented in chapter 3 were all run on a MacBook Pro 2019 model with a 2.4 GHz Quad-Core Intel Core i5 processor. Grab's algorithm was run on the NTNU cluster Idun.

4.1 Comparing Baseline Heuristic to MIP

In relation to research question 1 the data presented was run through the baseline method. All the consignments were divided to create orders, where each

order is assigned to a pallet. The process was then repeated for the MIP program. The results given by Grab's algorithm for the two approaches can be viewed in 4.1, where the difference in result is illustrated with percentages. The total number of pallets needed to pack all the consignemnts, according to the splitting made by the different methods, compared to how many of the orders were deemed unstable by Grab is shown in table 4.2. The difference in computational performance is shown in table 4.3.

Method	**Stability**		**Compacity**		**Weight**		**Height**	
	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ
Next Fit	2.33	1.312	0.749	0.109	320	179	1413	311
MIP	1.91	0.247	0.718	0.066	327	105	1486	212
Change (%)	-18.0%	-81.2%	-4.13%	-39.4%	2.19%	-41.3%	5.17%	31.8%

Table 4.1: The average and standard deviasion of measurements provided by Grab using orders created by Next Fit and MIP. The change is noted in procentages difference from Next Fit.

Method Used	**Pallet Orders Created**	**Pallet Orders Stacked**	**% Acceptable Pallets Produced**
Next Fit	878	829	94,4 %
MIP	871	845	97,0 %

Table 4.2: Orders created by the Next Fit and the MIP, with how many of them Grab was able to generate a packing pattern for.

Method	**Run time**	**Nr of allocations**	**Memory used**
Next Fit	2.50 s	4.51 M	714 MiB
MIP	173 s	94.0 M	6.84 GiB

Table 4.3: Performance measures given by @*time* in Julia for the baseline Next Fit and the MIP

4.2 Comparing Baseline Heuristic to Simulated Annealing

The results presented in the tables in this section marked as Next Fit, are the same used in the previous section. To create tables that easily highlighted the difference the results were repeated. The data was run through simulated annealing, like with the baseline and MIP, to produce the orders, which were then fed to Grab's algorithm. The results are organized as the previous section where the difference in the orders produced are shown in table 4.4, the difference in the pallets needed to pack the planned orders vs the once that Grab were able to pack shown in table 4.5 and the computational differences in table 4.6.

Method	**Stability**		**Compacity**		**Weight**		**Height**	
	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ
Next Fit	2.33	1.312	0.749	0.109	320	179	1413	311
Simulated Annealing	1.86	0.193	0.704	0.053	326	76.0	1500	182
Change(%)	-20.2%	-85.3%	-6.01%	-51.4%	1.88%	-57.5%	6.16%	-41.5%

Table 4.4: The average and standard deviation of measurements provided by Grab using orders created by Next Fit and Simulated Annealing. The change is noted in percentages difference from Next Fit.

Method Used	**Pallet Orders Created**	**Pallet Orders Stacked**	**% Acceptable Pallets Produced**
Next Fit	878	829	94,4 %
Simulated Annealing	878	875	99,7 %

Table 4.5: Orders created by the Next Fit and simulated annealing, with how many of them Grab was able to generate a packing pattern for.

	Performance		
Method	**Run time**	**Nr of allocations**	**Memory used**
Next Fit	2.50 s	4.51 M	714 MiB
Simulated Annealing	3.37 s	17.5 M	1.12 GiB

Table 4.6: Performance measures given by *@time* in Julia for the baseline method Next fit and simulated annealing

4.3 Comparing Simulated Annealing with Single-Objective Cost Function to Multi-Objective Cost Function

To compare the two cost functions the results their orders created when run through Grab are gathered in table 4.7. The difference in orders created in order to pack all the consignments are shown in table 4.8, along with how many of them turned out give a stable enough result for Grab.

Cost Function	Stability		Compacity		Weight		Height	
	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ
Single Criteria	1.86	0.193	0.704	0.053	326	76.0	1500	182
Multi Criteria	1.86	0.221	0.703	0.053	326	77.8	1504	179
Change (%)	0%	14.5%	-0.142%	0%	0%	2.37%	0.267%	-1.65%

Table 4.7: Overview of the change in mean and standard deviation for stability, compacity, weight and height for simulated annealing using different objective function

Cost Function	Orders Created	Orders Stacked	% Acceptable Pallets
Single Criteria	878	875	99.7%
Multi Criteria	877	875	99.8%

Table 4.8: Overview over how many of the orders created were able to be stacked by Solwr's algorithm using the different cost functions for simulated annealing

4.4 Adding Machine Learning to Simulated Annealing

To determine a suitable data set for training and testing the different models, two different data sets were created and both were tested on the relevant models. The performance of the different models on the first data set is plotted in figure 4.1. For the second data set created, the model preformance is shown in 4.2. The exact loss given by each model on the different data sets are shown in 4.11, where one of the models ended up not being able to run on the first data set.

This is indicated by "-" in the table. Tables 4.9 and 4.10 shows some descriptive statistics for the parameters for the first and second data set respectively.

The three models with the lowest loss were tuned further to determine the most suitable model. From table 4.11 the three best preforming models were Random Forest Regressor, KNN Regressor and the Evo Tree Gaussian. For the Random Forest Regressor, the input parameters which could be tuned were the number of trees and the max depth of these trees. The different values used for those input parameters and the resulting loss is shown in table 4.12. From the table, a Random Forest Regressor using 30 trees, where the max depth is set to 5, gives the lowest loss out of the parameters tested. This gave a RMSE of 0.550 and a MSE of 0.262.

The different parameters tested on the KNN model are shown in 4.13. The best preforming model for the KNN model was the one using three neighbors with a RMSE of 0.588 and a MAE of 0.267.

The last model tuned was the Evo Tree Regressor, where the input parameters which could be changed were the number of rounds and their max depth. From the parameters tested, the best preforming was the model using 30 rounds and 3 as max depth. This gave a RMSE of 0.568 and a MAE of 0.248.

I included a table comparing the results from the Next Fit to the simulated annealing using the machine learning model in table 4.15. The difference in orders created by the two methods is shown in table 4.16. Finally a graph over each consignements acumalated predicted stability compared to the stability given by Grab is shown in 4.3.

Variable	**mean**	**min**	**median**	**max**	**std**	**mode**	**skewness**
Height heterogeneity	0.4497	0.0121	0.4444	1.0	0.0801	0.5	0.2595
Depth heterogeneity	0.4045	0.0122	0.3968	1.0	0.0788	0.4	0.7547
Width heterogeneity	0.4169	0.0122	0.4118	1.0	0.07840	0.4	0.5258
Number of boxes	51.79	1.0	51.0	259.0	10.98	51.0	4.021
Stability	1.901	1.617	1.828	22.64	0.3178	1.736	22.93

Table 4.9: Descriptive statistics for the first iteration of the dataset

Variable	**mean**	**min**	**median**	**max**	**std**	**mode**
Height	0.2028	0.033	0.204	1.05	0.0706	0.207
Depth	0.3602	0.086	0.38	0.8	0.1238	0.4
Width	0.2478	0.065	0.238	0.8	0.0901	0.3
Weight	6.186	0.180	5.07	83.0	4.151	12.7
Stability	2.025	1.730	1.853	15.68	0.6491	1.800

Table 4.10: Descriptive statistics for the second iteration of the dataset

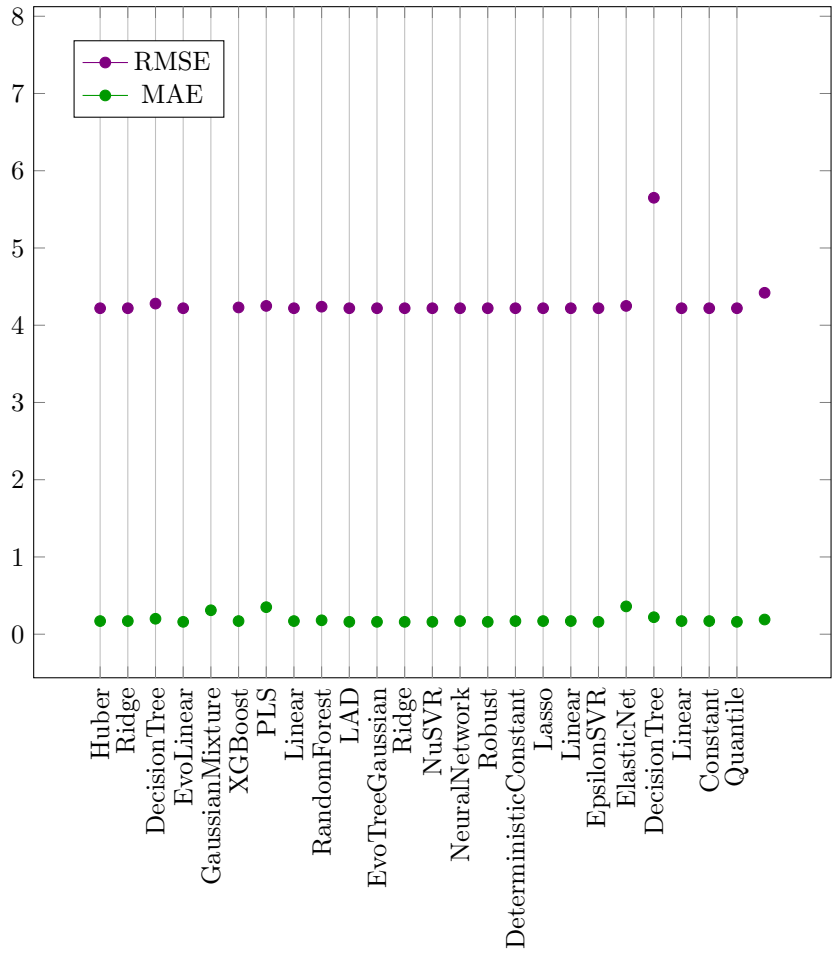


Figure 4.1: RMSE and MAE for tested models on data set 1

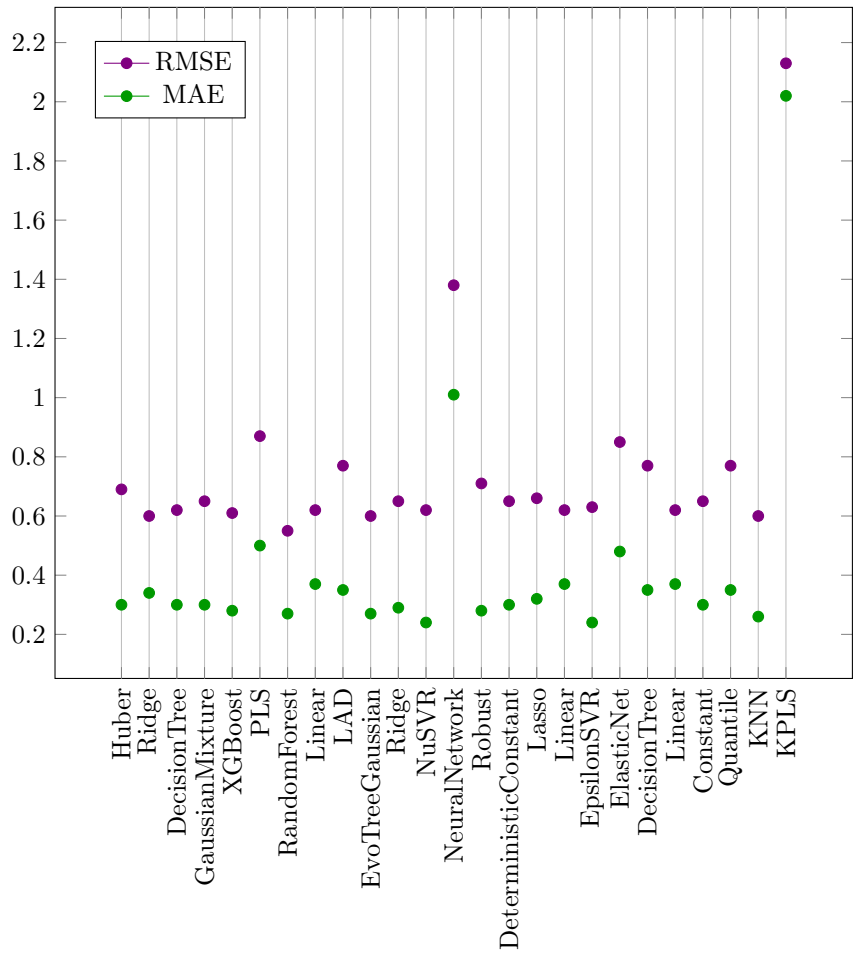


Figure 4.2: RMSE and MAE for tested models using data set 2

Models	Root-Mean-Square Error		Mean Absolute Error	
	Dataset 1	**Dataset 2**	**Dataset 1**	**Dataset 2**
Random Forest Regressor	4.24	0.55	0.18	0.27
KNN Regressor	4.42	0.60	0.19	0.26
Evo Tree Gaussian	4.22	0.60	0.16	0.27
Ridge Regressor	4.22	0.60	0.16	0.34
XGBoost Regressor	4.23	0.61	0.17	0.28
Linear Regressor	4.22	0.62	0.17	0.37
Decision Tree Regressor	5.65	0.62	0.22	0.30
NuSVR	4.22	0.62	0.16	0.24
EpsilonSVR	4.22	0.63	0.16	0.24
Ridge Regressor	4.22	0.65	0.17	0.29
Deterministic Constant Regressor	4.22	0.65	0.17	0.30
Constant Regressor	4.22	0.65	0.17	0.30
Gaussian Mixture Regressor	7.40	0.65	0.31	0.30
Lasso Regressor	4.22	0.66	0.17	0.32
Huber Regressor	4.22	0.69	0.17	0.30
Robust Regressor	4.22	0.71	0.16	0.28
Decision Tree Regressor	4.28	0.77	0.20	0.35
LAD Regressor	4.22	0.77	0.16	0.35
Quantile Regressor	4.22	0.77	0.16	0.35
Elastic Net Regressor	4.25	0.85	0.36	0.48
PLS Regressor	4.25	0.87	0.35	0.50
Neural Network Regressor	4.22	1.38	0.17	1.01
KPLS Regressor	-	2.13	-	2.02

Table 4.11: RMSE and MAE for models using dataset 1 and 2

Random Forest Tuning			
Number of Trees	**Max Depth**	**Root-Mean-Square Error**	**Mean Absolute Error**
1	unlimited	0.641	0.327
	1	0.652	0.298
	5	0.600	0.273
	10	0.832	0.326
5	unlimited	0.661	0.300
	1	0.651	0.296
	5	0.559	0.270
	10	0.589	0.276
15	unlimited	0.589	0.277
	1	0.651	0.299
	5	0.559	0.270
	10	0.589	0.276
30	unlimited	0.570	0.274
	1	0.651	0.297
	5	0.550	0.262
	10	0.565	0.263

Table 4.12: The values tested when tuning the random forest model's parameters on data set 2 and their resulting loss values

KNN Tuning		
Number of Neighbours	**Root-Mean-Square Error**	**Mean Absolute Error**
K = 1	0.659	0.316
K = 3	0.588	0.267
K = 5	0.595	0.257
K = 10	0.592	0.251
K = 15	0.599	0.248

Table 4.13: The values tested when tuning the KNN model's parameters on data set 2 and their resulting loss values

Evo Tree Tuning			
Number of Rounds	**Max Depth**	**Root-Mean-Square Error**	**Mean Absolute Error**
30	1	0.651	0.297
	3	0.568	0.248
	5	0.608	0.267
	10	0.606	0.270
50	1	0.651	0.297
	3	0.576	0.253
	5	0.611	0.261
	10	0.596	0.274
100	1	0.651	0.297
	3	0.578	0.265
	5	0.596	0.274
	10	0.589	0.269
150	1	0.651	0.297
	3	0.577	0.243
	5	0.577	0.247
	10	0.588	0.272

Table 4.14: Root mean square error and mean absolute error for different number of rounds and max depth for the evo tree model

Method	**Stability**		**Compacity**		**Weight**		**Height**	
	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ
Next Fit	2.330	1.312	74.90%	10.90%	320.0	179.0	1413	311.0
Simulated Annealing with ML model	2.470	1.461	71.65%	8.269%	232.5	93.91	1035	347.3
Change(%)	6.008%	11.36%	-4.339%	-24.14%	-27.34%	-47.54%	-26.75%	11.67%

Table 4.15: The difference in results given in percentages between the baseline method Next fit and the Simulated Annealing using ML model as cost function

Method Used	**Pallet Orders Created**	**Pallet Orders Stacked**	**% Acceptable Pallets Produced**
Next Fit	878	829	94,4 %
Simulated Annealing with ML Model	1231	1231	100,0 %

Table 4.16: pallets needed to pack the orders vs the number of orders Grab was able to stack made by the baseline Next fit and simulated annealing using ML.

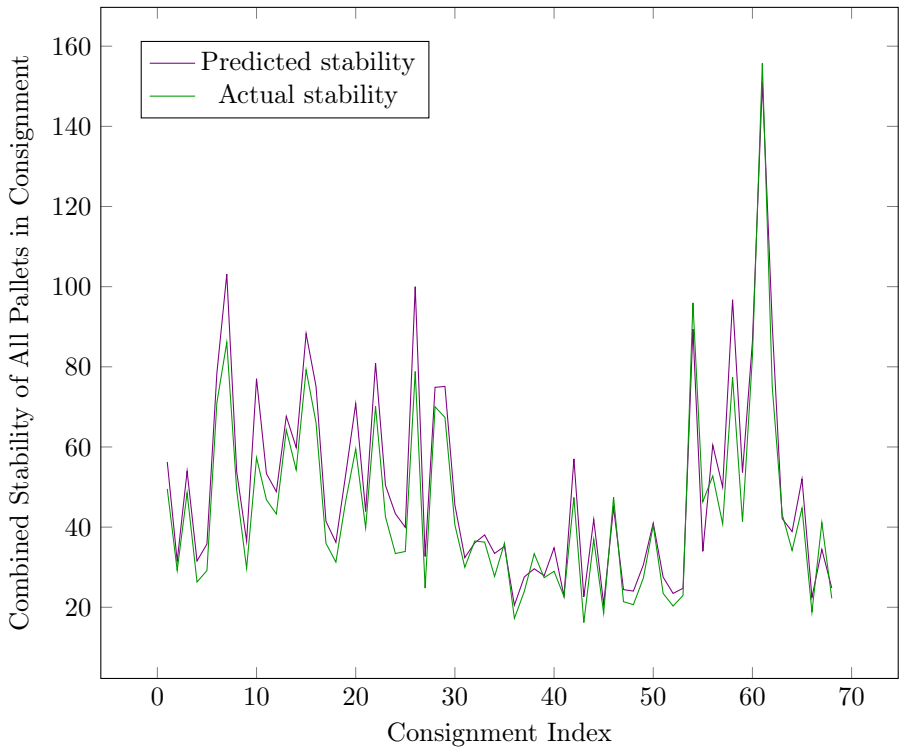


Figure 4.3: How the prediction stability made by the model compares with the actual stability given by Grab

4.5 Example of Splitting a Consignment

It can be hard to visualize the impact and differences made by the different approaches using only the tables presented in the previous sections. Therefore, I have included a closer look at how the different approaches would divide the same consignment into orders. The consignment was chosen based on the number of pallets the different methods needed to pack it. Many of the consignments created had over 10 pallets, which was harder to present in an organized way. The consignment used as an example consists of 88 items. All the items have a combined weight of 616.36 kg. To give a measure of the dimensions of the

different items in the consignment without listing them all, I have calculated the heterogeneity of the dimensions of the items using the calculations discussed in chapter 3. I have summarized the attributes of the consignment in a table, see table 4.17.

The results are grouped together by method, where each section presents the results from Grab's algorithm in a table, see table 4.18, 4.19 and 4.20, 4.21 and 4.22. Solwr provided software to visualize how Grab stacked the orders sent to it. The software allows for two kinds of plotting, one where each item is colored to be able to distinguish the boxes from each other. The other option visualizes how stable an item is where it is place in the stack. A stable placement of an item gives it a green color, and the lower the stability of the placed item, the redder it gets. This was used to visualize how each of the orders made by the different methods ended up looking, and how stable the items in the stack ended up being.

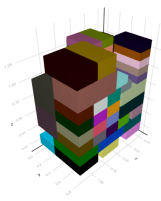
Nr of Items	Weight (Kg)	Volume (m^3)	Heterog. W	Heterog. H	Heterog. D
88	616.4	1.358	59.09%	60.23%	63.63%

Table 4.17: Overview over the attributes of the consignment used in the experiments

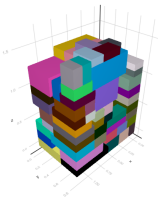
4.5.1 Baseline

Pallet Nr	Mean	1	2	3	4	5	6
Stability	2.625	1.738	1.848	1.969	1.944	1.956	6.292
Compacity	70.89%	77.41%	70.58%	68.72%	79.17%	73.27%	56.20%
Height (mm)	1296	1423	1560	1605	1392	1503	290
Weight (kg)	462.3	362.2	587.6	568.9	639.5	518.3	97.48

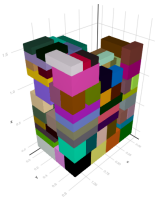
Table 4.18: Results generated by Grab's algorithm from consignment divided into orders by the baseline method



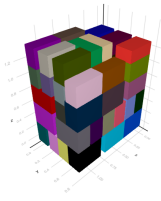
(a) Pallet 1



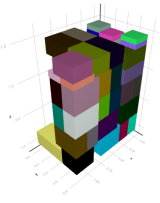
(b) Pallet 2



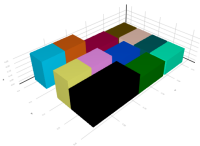
(c) Pallet 3



(d) Pallet 4

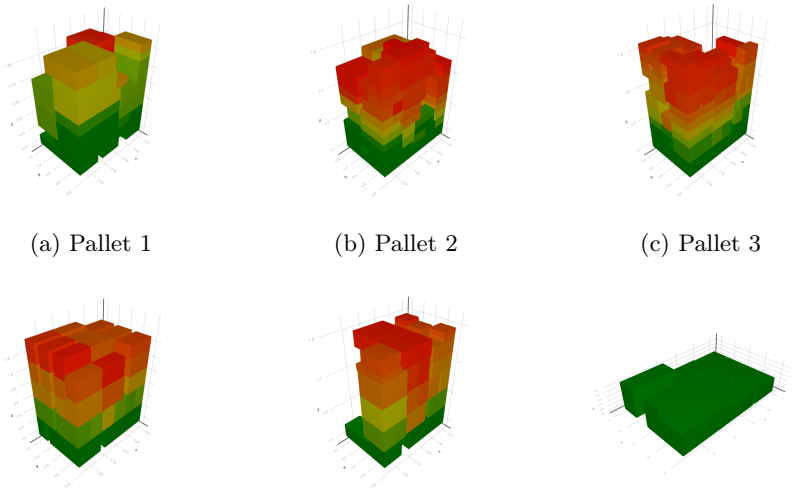


(e) Pallet 5



(f) Pallet 6

Figure 4.4: The item distribution between pallets when using the baseline method



(a) Pallet 1 (b) Pallet 2 (c) Pallet 3

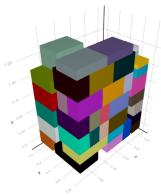
(d) Pallet 4 (e) Pallet 5 (f) Pallet 6

Figure 4.5: The item distribution between pallets when using the baseline method

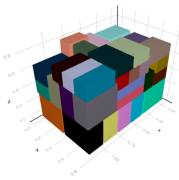
4.5.2 MIP

Pallet Nr	Mean	1	2	3	4	5	6
Stability	2.065	1.739	3.062	1.760	1.918	2.154	1.758
Compacity	75.39%	68.53%	78.70%	80.13%	73.58%	79.19%	72.20%
Height (mm)	1260	1427	838.0	1322	1253	1307	1410
Weight (kg)	462.4	580.2	370.7	543.3	404.2	412.1	463.6

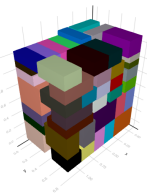
Table 4.19: Results generated by Grab's algorithm from orders made by MIP



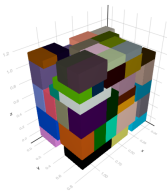
(a) Pallet 1



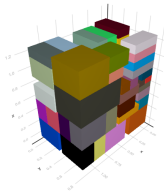
(b) Pallet 2



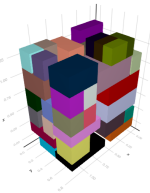
(c) Pallet 3



(d) Pallet 4

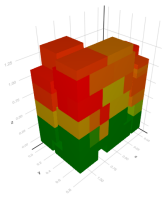


(e) Pallet 5

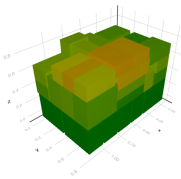


(f) Pallet 6

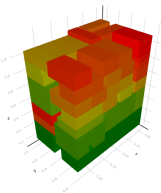
Figure 4.6: The item distribution between pallets when using MIP



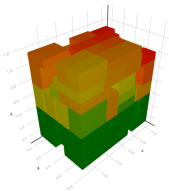
(a) Pallet 1



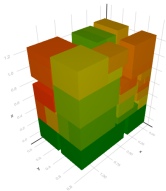
(b) Pallet 2



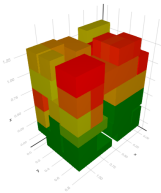
(c) Pallet 3



(d) Pallet 4



(e) Pallet 5



(f) Pallet 6

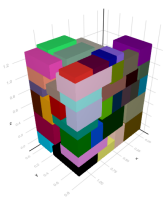
Figure 4.7: The item distribution between pallets when using MIP

4.5.3 Simulated Annealing

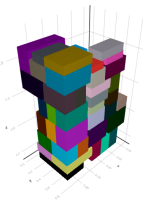
Single Objective Cost Function

Pallet Nr	Mean	1	2	3	4	5	6
Stability	1.835	1.746	1.739	1.788	1.742	1.749	2.245
Compacity	71.15%	73.75%	63.24%	72.84%	72.20%	75.49%	69.38%
Height (mm)	1334	1357	1706	1184	1288	1406	1064
Weight (kg)	462.3	501.9	551.2	417.4	477.9	519.2	306.4

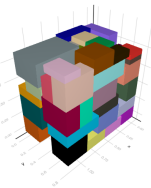
Table 4.20: Results generated by Grab's algorithm from orders made by Simulated Annealing



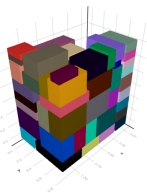
(a) Pallet 1



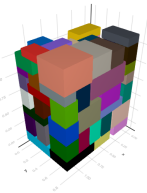
(b) Pallet 2



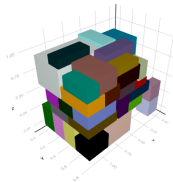
(c) Pallet 3



(d) Pallet 4

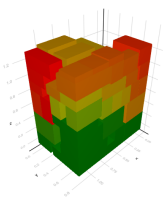


(e) Pallet 5

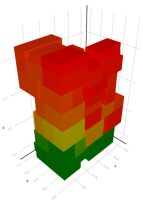


(f) Pallet 6

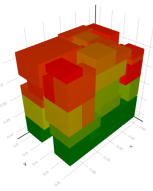
Figure 4.8: The item distribution between pallets when using simulated annealing



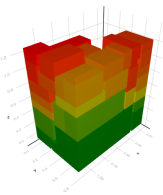
(a) Pallet 1



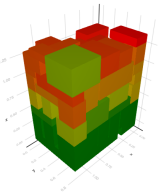
(b) Pallet 2



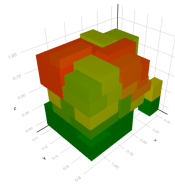
(c) Pallet 3



(d) Pallet 4



(e) Pallet 5



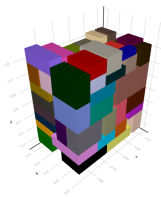
(f) Pallet 6

Figure 4.9: The item distribution between pallets when using simulated annealing

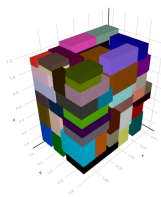
Multi Objective Cost Function

Pallet Nr	Mean	1	2	3	4	5	6
Stability	1.765	1.811	1.769	1.757	1.771	1.738	1.745
Compacity	71.45%	77.32%	70.63%	70.11%	68.06%	67.67%	74.92%
Height (mm)	1323	1371	1306	1437	1566	1015	1240
Weight (kg)	462.4	475.5	494.4	479.2	520.7	310.2	494.1

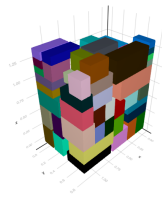
Table 4.21: Results generated by Grab's algorithm from orders made by Simulated Annealing using the multi objective cost function



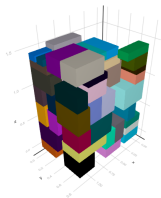
(a) Pallet 1



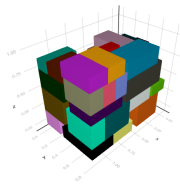
(b) Pallet 2



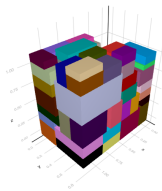
(c) Pallet 3



(d) Pallet 4

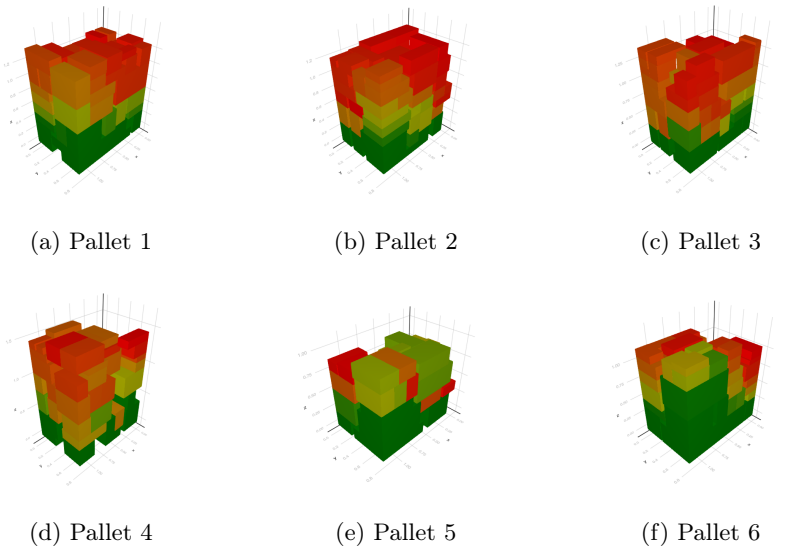


(e) Pallet 5



(f) Pallet 6

Figure 4.10: The item distribution between pallets when using simulated annealing



(a) Pallet 1 (b) Pallet 2 (c) Pallet 3

(d) Pallet 4 (e) Pallet 5 (f) Pallet 6

Figure 4.11: The item distribution between pallets when using simulated annealing

Machine Learning Model Cost Function

Since this method uses predictions I have included the predicted stabilities made by the model compared to the actual stabilities Grab gave each pallet, see table 4.23 and figure 4.12. From this, the $MAE = 0.7011$ and $RSME = 0.8372$.

Pallet Nr	Mean	1	2	3	4	5	6	7	8
Stability	2.779	1.734	1.865	1.763	5.715	1.876	1.804	1.757	5.715
Compacity	65.67%	67.36%	69.37%	72.17%	61.05%	74.91%	71.14%	65.53%	43.79%
Height (mm)	1036	1070	1392	1166	518	1217	1223	1381	321
Weight (kg)	346.7	379.73	435.5	428.4	103.7	429.9	462.6	462.8	71.38

Table 4.22: Results generated by Grab's algorithm from orders made by Simulated Annealing using the machine learning cost function

Pallet Nr	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Predicted Stability	2.773	2.465	2.307	4.565	2.333	2.896	2.516	4.959
Actual Stability	1.734	1.865	1.763	5.715	1.876	1.804	1.757	5.715
Squared Error	1.080	0.360	0.296	1.32	0.209	1.19	0.576	0.578

Table 4.23: Results generated by Grab's algorithm from orders made by Simulated Annealing using the machine learning cost function

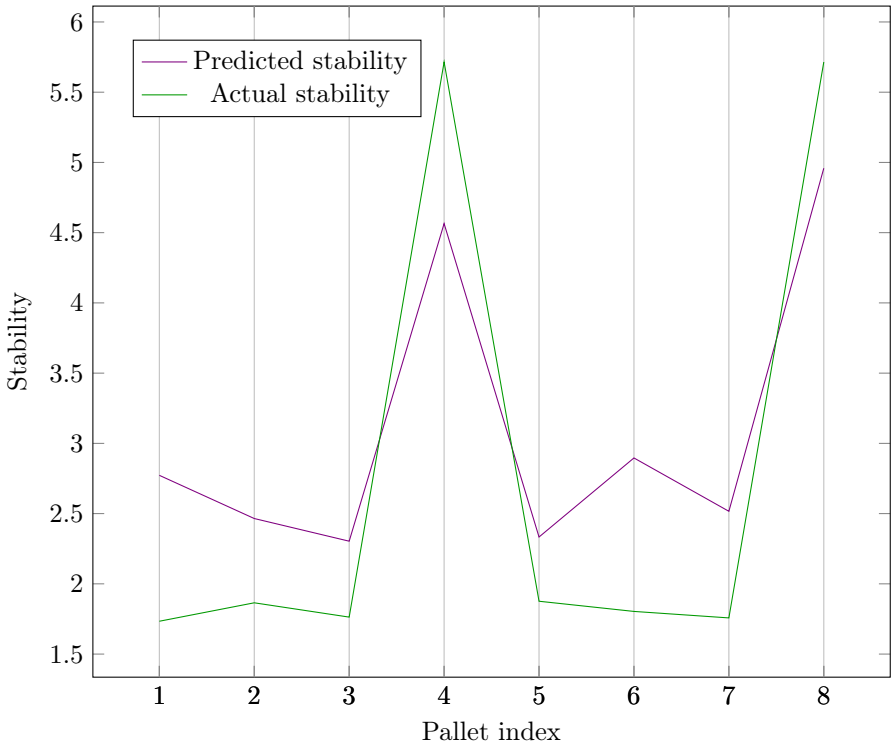
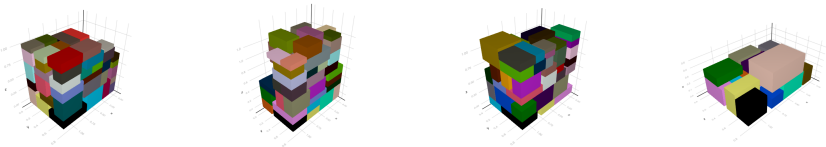
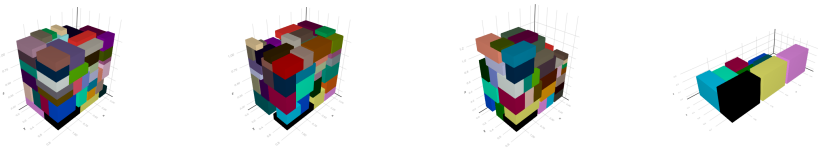


Figure 4.12: How the prediction stability made by the model compares with the actual stability given by Grab

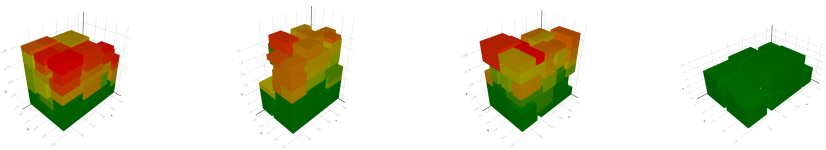


(a) Pallet 1 (b) Pallet 2 (c) Pallet 3 (d) Pallet 4

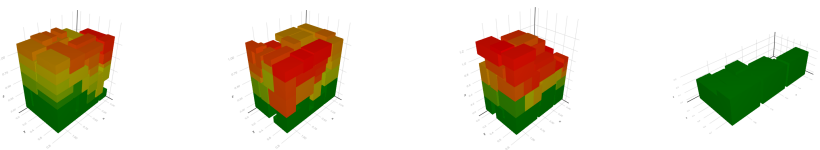


(e) Pallet 5 (f) Pallet 6 (g) Pallet 7 (h) Pallet 8

Figure 4.13: The item distribution between pallets when using simulated annealing



(a) Pallet 1 (b) Pallet 2 (c) Pallet 3 (d) Pallet 4



(e) Pallet 5 (f) Pallet 6 (g) Pallet 7 (h) Pallet 8

Figure 4.14: The item distribution between pallets when using simulated annealing

Chapter 5

Discussion

In this chapter, the implementations with their results after running them through Grab will be discussed in relation to the research questions stated at the start of this thesis. Each will be presented sequentially, starting off with research question one.

> **Research Question 1:** Given a baseline heuristic for dividing a consignment, how does a MIP with the goal of minimizing the number of pallets preform compared to the baseline?

From table 4.1 we can see that the average stability of the pallets created using the MIP decreases. This might be attributed to the increase in both height and the decrease in capacity, meaning that the average pallet is higher, but the items are less volume efficient. This means that the items higher up in the stack are placed on average on a smaller, less sound foundation of boxes, which might decrease the stability. Something that is important to note is that from table 4.2 we see that the MIP has a higher percentage of planned orders that ends up being stable enough for Grab to pack, which gives the more results which effects the average in the calculation for stability, which could be the reason for the stability being lower. Some of these consignments might have had more difficult items to stack, resulting in difficult orders for Grab to stack. Orders like these might have been eliminated from the baseline orders after running through Grab, which results in a higher average stability, since what potentially would have been an order with a lower stability was not included in

the calculation.

The MIP gives an average pallet that has a lower standard deviation for weight, stability, and height, meaning that the average pallet is more consistent and similar than those created by the baseline. Since the MIP wants to keep the number of pallets needed to a minimum, it will try to fit as many items as possible on each pallet, which might eliminate the cases where the last pallet is just a collection of the residual items in the list. This might be the case for the baseline approach, given how it divides the items. These types of pallets might be a contributing factor to the overall average of the stability being higher for the baseline. Since they just consist of residual items in the list which might fit on other pallets, but due to the nature of the Next Fit algorithm, are not able to be placed on another pallet.

Despite needing fewer pallets to divide the different consignments, it has a higher percentage of orders that turned out to be stable enough for Grab. Which is preferable if we were to decide which of these methods to use, but it is not able to pack all of them, which means it still would need reworking.

In terms of performance, the MIP requires significantly more time than the baseline. In terms of percentages, the MIP requires 6820% more time than the baseline. It also is much more computationally expensive according to table 4.2. This is not surprising since MIP's are known to be computationally expensive. If Solwr wanted to explore this further and create better results, they probably must increase complexity by expanding the optimization goal, which would in turn make it a harder problem for the MIP to solve. Therefore, I think this method is less suited for further exploration, especially given that the average pallet is less stable than the baseline ones.

To summarize, the MIP creates pallets that are more like each other compared to the baseline. However, they do receive a lower average score on both stability and capacity, which are important characteristics of a good order. It is also more computationally expensive compared to the baseline, but in turn it gives a slightly higher percentage of orders that Grab can pack.

How will simulated annealing preform compared to the baseline? This is what we will be discussing in research question 2.

> **Research Question 2:** Given a baseline heuristic for dividing a consignment, how does a heuristic method like simulated annealing with the cost function of minimizing the number of pallets preform compared to the baseline?

Starting off with the results from Grab, we see that simulated annealing gives a lower average stability and compacity compared to the baseline. The average pallet it produces is slightly heavier and higher than the baseline method, where the height might contribute to the decreasing stability. The more items stacked on top of each other, the more forces are at play when Grab is moving. Simulated annealing likely suffers the same consequences as MIP, where the average stability is worse due to each pallet having more items on them compared to the orders made by Next Fit.

From table 4.5 we see that simulated annealing requires the same amount of pallets to pack all the items, and more of the orders created by the method are deemed stable by Grab than the baseline. This means that, like the discussion around the previous research question, we have more orders which count towards the average pallet, which can be causing the average stability and compacity to be lower than that of the baseline. Simulated annealing only fails to create 0.3% of orders stable enough for Grab to pack, which is the best ratio discussed so far.

According to table 4.6 we see that simulated annealing is more computationally expensive than the baseline, but in terms of run time they are only 0.87 seconds apart. And comparing their differences to the last research question, simulated annealing is a better alternative, requiring significantly less resources.

To summarize, simulated annealing has a worse average result, but the orders generated by the method has a higher acceptance percentage by Grab. It is more computationally expensive, but not as expensive has the MIP discussed in the previous section. Given that simulated annealing is highly customizable, with some changes it might produce better division of items for Grab.

The potential of simulated annealing was further expanded with research question 3.

> **Research Question 3:** What changes can be observed if the cost function for simulated annealing is expanded to include the goal of minimizing the item height heterogeneity and weight distribution, compared to only using the goal of minimizing the pallets?

Going from a single objective cost function to a multi-objective cost function gave little improvements to the results. In fact, the average stability remained the same as the pallets using the single objective cost function. Despite including a measure for lowering the standard deviation of weight, it increased by 2.37%. I mentioned earlier that including a measure for the height heterogeneity, might

help make pallets more stable. Since, more items of similar height can be used to create an even layer. This seemed to have no effect, since the average stability got worse.

It did require one less pallet to pack all consignment but was still unable to get Grab to pack everything. When using the new cost function, every parameter was given equal weighting, this can be a factor in why the results were not drastically changed, since the values were not given the proper weighting. Further, exploring this might produce better insight into what factors can attribute to a more stable pallet.

The last research question was a bigger question, where I will start by discussing research question 4.1.

> **Research Question 4.1:** What is a suitable data set to represent the relationship between an order and its stability?

As we have seen, two different data sets have been tested to find a suitable one for determining the stability of an order. Where the first one tested, which had parameters which were thought to be descriptive enough, gave large root-mean-square error values. With the scale used for stability, a lower value was preferable. To lower the loss, the idea of expanding the data set to include all boxes and not just some of the statistics we thought might be useful, was implemented. This data set contained most of the information available. Things like product name and ID were not included, because I did not think it would have much effect. When the changes were implemented, the models gave lower RMSE values and lowered the gap between RMSE and MSE, compared to the first dataset.

One might think that using a classification dataset would be something better suited, since a pallet produced by Grab which is deemed stable is acceptable, so we only need to weed out the pallets that are potentially unstable. This was discussed and since it was desirable to get the best possible combination of items, a model which could be used to increase the "potential" quality of Grab was more useful. Since the model only knows what it sees, I wanted to include some of the unstable pallets, so they would be presented in the data set. This would make it easier for simulated annealing to identify unsuitable combinations of items and avoid them. The problem which stopped this from happening was that Grab does not provide a number for these kinds of orders. I tried to alter the code provided to make Grab's algorithm give me the results no matter how bad the stability was. However, I was unsuccessful. This could

be useful to include if exploring the use of machine learning further.

The size of the data set is small compared to what is usually used for training models. This is a limitation to the model implemented in this thesis. One way to improve is to increase the number of items the model could handle in order to increase the orders that could be included in the data set. Taking everything into account, out of the data sets tested in this thesis, the one passing in all the items as parameters to the model seems to be the best fit for our problem.

Moving on to research question 4.2, where we will discuss the choice of model, and the predictions made by it compared to the actual stability value given by Grab.

> **Research Question 4.2:** Given the data set in RQ 4.1 what is a suitable model for implementing a stability prediction in simulated annealing?

Using the results of the different models and their loss from the second data set, the Random Forest Regressor was chosen. The differences between the top models were marginal before and after tuning. In discussions with Håkon Hukkelås, a PhD candidate at NTNU, he hypothesized that the more traditional models would be a better fit than the deep learning models that are popular today. This proved to be the case, as we can see from table 4.11, where the neural network model is one of the worst preforming models. It could be interesting to implement a more customized network using Julia's Flux, but this was not further experimented with in the thesis, due to time. Two out of the three top models were based on trees, which might be due to the model's ability to combine multiple trees that are able to capture the nuances of the data. Both models using trees performed better when the number of trees were high. Indicating that the data might be hard to generalize.

To summarize, according to the results the best suited model, based on the models tested on the final data set, turned out to be the random forest regressor with 30 trees, each with a max depth of 5.

This brings us to the larger question of research question 4.

> **Research Question 4:** Can Grab's stability calculation be emulated using a machine learning model to create more stable pallets using simulated annealing?

Since simulated annealing uses the sum of all the stabilities in the consignment, this was also used when plotting the prediction and actual stabilities in figure

4.3. From the figure, we see that the prediction closely resembles the actual stability, which means that a machine learning model is capable of emulating Grab's stability values. Given that the predictions are so closely correlated with the actual stabilities show that the model has a tendency to overfit. This might be due to the model being complex, we saw that the more trees in the models, the lower the loss got. Another reason for this might be that the orders created by the previous methods explored, which deal with some of the same consignments, were used in the data set. This means that the model had seen some of the same consignments consisting of the same items in the training set. Combining the model with simulated annealing exposes the model to multiple different configurations a consignment can be split into, making it possible that some of the configurations were the same as the ones in the data set it was trained on. I know this should be avoided, but due to lack of data and wanting to test if using machine learning was suitable in simulated annealing, the orders were included. The results used in the data set from the other approaches were the results generated for the project. This means that some of the orders, which were filtered away when wanting to focus on larger orders, were included in the data set. This could be solved by creating a larger data set and replacing the orders which appear in the test scenario with new observations to avoid the model being exposed to the same consignments.

If we look at the effects this have on the simulated annealing compared to the baseline, we see that we get an increase in the overall stability as well as being able to pack all the pallets that are planned from table 4.15 and 4.16. We see that the pallets needed increases, which shows that the number of orders created by the previous methods might be unrealistically low to achieve the quality needed for Grab. The average pallet using the ML model are shorter and lighter, which might suggest that the height of the pallets created previously are too tall.

The case presented in section 4.5 provides some illustrations of the points made in this discussion. Looking at the pallets set from the baseline orders, we see that the last pallet is small compared to the others in terms of the weight and the height. This is a result of the sequential nature of the method, where the last pallet is sort of a residual pallet if the items cannot fill up the full pallets. This in turn brings the average stability up. If we were to compare table 4.18 and 4.19, we might want to choose the baseline, since the average pallet looks better. But if we want a more even distribution of the items, we might want to pick the MIP method. From the case, we see that the cost function using simulated annealing gives the best average stability, which can be attributed to

the use of more pallets. We see that two of the pallets among the orders created by this method only contain stable items on the pallet. This helps to bring the overall stability up. The graph shows how the predicted vs actual stability is similar, indicating that the model can give a good prediction of the individual pallets in the consignment.

5.1 Future Work

After concluding the experiments and assessing the results, I have several ideas which could be interesting to explore further. Starting off with generating a larger, more diverse data set for training the models. Is the model still able to predict somewhat accurately if the data set is increased significantly? It could be worth looking into what effects using the correlation between the model's predictions and the actual values could boost performance. Conducting experiments to determine what might be a suitable weighting to the different attributes in the simulated annealing cost function using a multi-objective optimization goal. Adding the stability prediction model to the multi-objective cost function could be interesting. As mentioned in the discussion, trying to implement a model using Flux to see if a neural network which is more customized can be an alternative to the models tested here. Simulated annealing is highly customizable, and some ideas I had to improve the method are further testing different cooling schedules. The cooling schedule used in this thesis would be further tested, which might improve the resulting solution. Experimenting with moving more boxes in the move function could also be interesting to investigate. In this thesis, some of the nuances were removed to make the problem smaller and more manageable. Further work could implement some of the problems that were simplified in this thesis, like adding back the aspect of ordering the items in an order after their location in the warehouse. Which would add back the issue of planning the orders with the goal of making the routes as efficient as possible.

Chapter 6

Conclusion

At the beginning of this thesis, I stated that the overall goal was to gather information on what might be the best way forward if Solwr wants to divide a consignment in a way that would improve Grab's performance compared to today's solution. To determine this, I defined and discussed several research questions.

Methods using the goal of minimizing the number of pallets compared to the heuristic Next Fit showed to give a lower average stability. But the methods did make more orders which were suitable for Grab to pack. One of the methods, simulated annealing, was easier to customize, however using a multi-objective cost function compared to a single-objective cost function did not improve the performance. Since the multi-objective cost function used the same weighting for all the different parameters, it could still provide useful information, if the weighting was investigated further.

The method showing the most promising results was simulated annealing using a machine learning model for predicting the stability as the cost function. This increased the average stability of the pallets produced. It was also the only method tested where Grab was able to pack all the orders created. The prediction plotted next to the actual stability indicate that the model is overfitting.

Given these findings, going forward I think that expanding the data set used in this thesis to include more orders and fixing the issue of overfitting, would be the

most promising way forward. Using multiple parameters in the cost function is also something that could be a way forward, where improving the weighing of the different parameters and maybe incorporating the machine learning prediction could a potential way forward for Solwr. After implementing and experimenting with simulated annealing, I can say that I find the method highly customizable, where this can be done more easily than the other method tested. So, if Solwr were to look more on the area, I would say simulated annealing has more easily implementable options than that of MIP.

Bibliography

[1] Gordana Radivojević and Luka Milosavljević. "The concept of logistics 4.0". In: *4th Logistics International Conference*. 2019, pp. 23–25.

[2] Robert Bouge. "Growth in e-commerce boosts innovation in the warehouse robot market". In: *Industrial Robot: An International Journal* 43.6 (2016), pp. 583–587.

[3] Minqi Zhang, Sven Winkelhaus, and Eric H Grosse. "Evaluation of human workload in a hybrid order picking system". In: *IFAC-PapersOnLine* 54.1 (2021), pp. 458–463.

[4] *Solwr Company*. https://www.solwr.com/company/about-us. Accessed: 2023-03-14. 2023.

[5] EE Bischoff and MSW Ratcliff. "Loading multiple pallets". In: *Journal of the Operational Research Society* 46 (1995), pp. 1322–1336.

[6] Eberhard E Bischoff, F Janetz, and MSW Ratcliff. "Loading pallets with non-identical items". In: *European journal of operational research* 84.3 (1995), pp. 681–692.

[7] A Paul Davies and Eberhard E Bischoff. "Weight distribution considerations in container loading". In: *European Journal of Operational Research* 114.3 (1999), pp. 509–527.

[8] Batin Latif Aylak et al. "Application of machine learning methods for pallet loading problem". In: *Applied Sciences* 11.18 (2021), p. 8304.

[9] RL Rao and SS Iyengar. "Bin-packing by simulated annealing". In: *Computers & Mathematics with Applications* 27.5 (1994), pp. 71–82.

[10] Thomas Kämpke. "Simulated annealing: use of a new tool in bin packing". In: *Annals of Operations Research* 16 (1988), pp. 327–332.

[11] Maxence Delorme, Manuel Iori, and Silvano Martello. "Bin packing and cutting stock problems: Mathematical models and exact algorithms". In: *European Journal of Operational Research* 255.1 (2016), pp. 1–20.

[12] Michael T. Goodrich. *Bin Packing*. Accessed: 2023-06-06. 2019. URL: https://www.ics.uci.edu/~goodrich/teach/cs165/notes/BinPacking.pdf.

[13] Subhash Suri. *Approximation Algorithms*. Accessed: 2023-05-23. Nov. 2017. URL: https://sites.cs.ucsb.edu/~suri/cs130b/BinPacking.

[14] Edward G Coffman et al. "Bin packing approximation algorithms: survey and classification". In: *Handbook of combinatorial optimization*. 2013, pp. 455–531.

[15] Andreas Bortfeldt and Gerhard Wäscher. "Constraints in container loading–A state-of-the-art review". In: *European Journal of Operational Research* 229.1 (2013), pp. 1–20.

[16] Mauro Dell'Amico and Matteo Magnani. "Solving a Real-Life Distributor's Pallet Loading Problem". In: *Mathematical and Computational Applications* 26.3 (2021), p. 53.

[17] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680.

[18] Vojtěch Černỳ. "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". In: *Journal of Optimization Theory and Applications* 45.1 (1985), pp. 41–51.

[19] Nicholas Metropolis et al. "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[20] Scott Kirkpatrick. "Optimization by simulated annealing: Quantitative studies". In: *Journal of statistical physics* 34 (1984), pp. 975–986.

[21] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[22] Zhi-Hua Zhou. *Machine learning*. Springer Nature, 2021.

[23] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. Springer, 2006.

[24] European Pallet Association. *Epal Euro Pallet (EPAL 1)*. Accessed: 2023-03-21. 2023. URL: www.epal.eu.

[25] Anthony D. Blaom et al. *MLJ: A Julia package for composable machine learning*. Version 0.14.1. Nov. 2020. DOI: 10.5281/zenodo.4178918. URL: https://doi.org/10.5281/zenodo.4178918.

[26] JuliaHub. *EvoTrees*. Version 0.14.11. 2023. URL: https://evovest.github.io/EvoTrees.jl/dev/.

[27] JuliaHub. *NearestNeighborModels*. Version 0.2.3. 2023. URL: https://juliaai.github.io/NearestNeighborModels.jl/dev/.

[28] Antonello Lobianco. "BetaML: The Beta Machine Learning Toolkit, a self-contained repository of Machine Learning algorithms in Julia". In: *Journal of Open Source Software* 6.60 (2021), p. 2849. DOI: 10.21105/joss.02849. URL: https://doi.org/10.21105/joss.02849.

Appendices

Appendix A

MIP

Listing A.1: Baseline - Implementation of MIP

```
function create_optim_model(order, pallet)

#? Sets and indicies
# i - item/box type
# j - container

N = size(order, 1)
M = 100

#? Parameters
# D_i - Dimention of item/box i
# W_i - Weight of item/box i
# n_i - Number of items/boxes of type i
#* All containers have the same size
# K_j_d - Volume capacity of container j - 180 cm x 80 cm
    x 110cm
# K_j_w - Weight capacity of container j  - 2000 kg

D::Array = order."MDMITEMVOLUM_ML"
W::Array = order."MDMITEMGROSSWEIGHT_G"
n::Array = order."ORDERLINEAMOUNT"

K_d = pallet.volume
```

```
K_w = pallet.max_weight_allowed

model = Model(Gurobi.Optimizer)

#? Variables
# x_ij - Number of items/boxes of type i in container j
# yj - Binary value, 0 when container is not used, and 1
    if container j is used

@variable(model, y[1:M], binary = true)
@variable(model, x[1:N, 1:M], integer = true, lower_bound
    = 0)

#? Objective function
# Min Z = Sum yj for all containers
@objective(model, Min, sum(y[j] for j in 1:M))

#? Constaints
# Sum xij = ni, the sum of all items of type i in
    containers j must be equal to the number of items of
    type i (we must pack all boxes)
# The sum of dimention of item type i multiplied with the
    number of items of type i must be able to fit in a
    containers dimention constrained, and the container
    must have been activated
# The same as over but for Weight

for i in 1:N
    @constraint(model, sum(x[i, j] for j in 1:M) == n[i])
end

for j in 1:M
    @constraint(model, sum(D[i] * x[i, j] for i in 1:N) <=
        (K_d * y[j]))
    @constraint(model, sum(W[i] * x[i, j] for i in 1:N) <=
        (K_w * y[j]))
end

for j in 1:M-1
    @constraint(model, y[j] >= y[j+1])
end
```

```
    optimize!(model)
    nr_of_pallets = objective_value(model)
    x = value.(x)
    return x, Int(nr_of_pallets)
end
```

Appendix B

Next-Fit Implementation

Listing B.1: Baseline - Implementation of Next-fit Heuristic

```
function split_items(pallet::Pallet, original_items::
    Array, limit)
all_pallets::Array{PackedPallet} = []
items = copy(original_items)
while length(items) > 0
    items_on_pallet, packed_volume, packed_weight =
        pack_pallet(items, pallet, limit)
    boxes = getfield.(items_on_pallet, :box)
    heterogeneity = heterogeneity_dimentions(boxes)
    packed = PackedPallet(sort_list_by_weight(
        items_on_pallet), packed_volume, packed_weight
        , heterogeneity[1])
    for item in packed.items
        filter!(e -> e != item, items)
    end
    push!(all_pallets, packed)
end
return Result(all_pallets, pallet)
end
```

Listing B.2: Pack Pallet - Helper method to Next Fit

```
    function pack_pallet(items::Array{Item}, pallet::
        Pallet, limit)

    items_on_pallet::Array{Item} = []
    packed_volume = 0.0
    total_weight = 0.0


    for item in items

        total = item.volume + packed_volume

        if limit !== nothing
            if length(items_on_pallet) == limit ||
                total >= pallet.volume
                break
            end
        end

        if total >= pallet.volume
            break
        end

        push!(items_on_pallet, item)

        packed_volume += item.volume
        total_weight += item.box.weight

    end
    return items_on_pallet, packed_volume,
        total_weight
end
```

Appendix C

Simulated Annealing

C.1 Move set

Listing C.1: generate neighbor

```
function generate_neighbour_state(state::Result, ML)
neighbour = copy(state)

if (neighbour.nr_of_pallets == 1)
    return neighbour
end

if ML
    max_items_on_pallet = 60
    take_from, give_to = generate_exchange_pair(neighbour,
        max_items_on_pallet)
    if give_to == -1
        return neighbour
    end
else
    take_from, give_to = generate_exchange_pair(neighbour,
        nothing)
end
```

```
    # Takes the first box that fits the new pallets volume

    for i in eachindex(neighbour.pallets[take_from].items)
        item = neighbour.pallets[take_from].items[i]
        if is_valid_move(item, neighbour.pallets[give_to],
            neighbour.pallet_type)

            # Add the item to the new pallet
            push!(neighbour.pallets[give_to].items, item)
            neighbour.pallets[give_to].volume += item.volume
            neighbour.pallets[give_to].weight += item.box.
                weight

            # Remove the box we moved from the original pallet
            deleteat!(neighbour.pallets[take_from].items, i)
            neighbour.pallets[take_from].weight -= item.box.
                weight
            neighbour.pallets[take_from].volume -= item.volume

            # Remove the pallet if the move leaves the pallet
                empty
            if (length(neighbour.pallets[take_from].items) ==
                0)
                splice!(neighbour.pallets, take_from)
                neighbour.nr_of_pallets = length(neighbour.
                    pallets)
            end

            neighbour.standard_deviation_weight =
                standard_deviation_of_pallets(neighbour.
                pallets)

            return neighbour
        end
    end
    return neighbour
end
```

C.2 Cost Function

Listing C.2: Function for calculating the acceptance criteria

```
function acceptance_probability(state, new_state, temperature,
    mode)

    if mode == "minimizing_pallets"

        score_old_state = basic_evaluate_state(state)
        score_new_state = basic_evaluate_state(new_state)

    elseif mode == "multi_criteria"

        score_old_state = modified_evaluate_state(state)
        score_new_state = modified_evaluate_state(new_state)

    elseif mode == "ML"
        score_old_state = ML_evaluate_state(state)
        score_new_state = ML_evaluate_state(new_state)
    end

    delta_score = score_new_state - score_old_state

    if mode == "ML"
        if delta_score > 0
            return 1.0
        else
            return exp((delta_score) / temperature)
        end
    else
        if delta_score < 0
            return 1.0
        else
            return return exp((-delta_score) / temperature)
        end
    end

end
```

Listing C.3: Minimizing Number of pallets

```
    function basic_evaluate_state(state::Result)
    return state.nr_of_pallets
end
```

Listing C.4: Minimizing Number of pallets, item height heterogeneity and σ_{weight}

```
function modified_evaluate_state(state::Result)
if state.nr_of_pallets == 1
    return 0.0
end
return state.nr_of_pallets + state.
    standard_deviation_weight + state.avg_heterogeneity
end
```

Listing C.5: Maxmizing Stability of Pallets

```
function ML_evaluate_state(state::Result)
calculate_stability_for_state(state)
return state.stability
end
```

Listing C.6: Helper function for calculating stability of the configuration and individual pallets

```
function calculate_stability_for_state(state)
stability = 0
for pallet in state.pallets
    x = format_ML_input(pallet.items)
    stab = first(MLJ.predict(mach, x))
    pallet.stability = stab
    stability += stab
end
state.stability = stability
end
```

C.3 Configuration

Listing C.7: Calculate Heterogeneity

```
function heterogeneity(array::Array)
    return length(countmap(array)) / length(array)
end
```

Listing C.8: Calculate Heterogeneity for every dimension

```
function heterogeneity_dimentions(boxes::Array)
    height = getfield.(boxes, :height)
    width = getfield.(boxes, :width)
    depth = getfield.(boxes, :depth)

    heterogeneityHeight = heterogeneity(height)
    heterogeneityWidth = heterogeneity(width)
    heterogeneityDepth = heterogeneity(depth)

    return heterogeneityHeight, heterogeneityWidth,
        heterogeneityDepth
end
```

NTNU
Norwegian University of
Science and Technology