Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Information Technology
2023 | LIU-IDA/LITH-EX-A--23/059--SE

Optimizing E-commerce Logistics: A Multi-Metric Approach to the Bin Packing Problem

Optimering av e-handelslogistik: Ett flermetriskt tillvägagångssätt till lådpackningsproblemet

Anton Magnusson
Vilhelm Melkstam

Supervisor : Abdelazim Hussien
Examiner : Zebo Peng

External supervisor : Hasan Mert Mercan

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

The optimization of package selection in logistics, particularly within the realm of e-commerce, offers numerous potential advantages, such as a reduction in environmental impact and decreased costs. This thesis addresses the problem of allocating items to the minimum number of packages, known as the bin packing problem, by proposing various heuristics. We develop and assess heuristics for assigning products to groups, while heuristics for accommodating these groups within packages are derived from previous research. These heuristics are evaluated within a commercial context, taking into account factors such as delivery cost, environmental impact, and their applicability in real-time systems. Our findings indicate that optimal solutions for smaller orders can be ascertained within a reasonable timeframe, while even rudimentary heuristics yield satisfactory results. It was determined that a key attribute of an effective solution was lowering the number of packages used, as this correlates with reduced shipping costs and environmental impact.

Acknowledgments

We would like to express our sincere gratitude to our examiner, Zebo Peng[1], and our supervisor, Abdelazim Hussien[2], for their unwavering support and invaluable feedback throughout the development of this thesis.

Additionally, we are grateful to our external supervisor, Hasan Mert Mercan, for his expert guidance and technical insights. We would also like to extend our appreciation to the team at Skrym, particularly Jakob Nordfeldt and Max Danielsson, for generously sharing their expertise and technical assistance. We are thankful to Skrym for providing us with the data and computational resources necessary to conduct our experiments.

Finally, we would like to express our immense gratitude to our opponents, Clara Gallon and Rebecca Södereng, for their constructive feedback and insightful suggestions. We are also grateful to Carl Fransson, Lucas Laukka and Maya Henriksson for their valuable assistance with reviewing the thesis.

[1] https://liu.se/en/employee/zebpe83
[2] https://liu.se/en/employee/abdhu99

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

3D-BPP three-dimensional bin packing problem. 2, 3, 5, 6, 28, 66

BPP bin packing problem. iii, x, 2–6, 45, 63, 67

C&P cutting and packing. 3, 4

DOE design of experiments. 24, 26

FEFCO European Federation of Corrugated Board Manufacturers. vii, 13, 14, 34

GCP Google Cloud Platform. 31, 60

GLS guided local search. 5

HBEFA The Handbook Emission Factors for Road Transport. 16, 17

ICCT International Council on Clean Transportation. 17

LCU load capacity utilization. 15–18, 35, 36, 63

ME material emissions. 34, 38, 46

NTM Network for Transport Measures. viii, 14–17, 35, 62, 63

ODP open-dimension problem. 3, 6, 67

PG price group. 12, 13, 32, 33

SBPP single bin packing problem. 2, 5, 6, 30, 37, 43, 45, 51, 56, 63, 64, 67, 68

SC shipping cost. 38, 46

SCB Statistics Sweden. 18, 36

SE shipping emissions. 18, 36, 38, 46

SKU stock keeping unit. 7, 8, 27

vCPU virtual CPU. 31

1 Introduction

This chapter serves as an introduction to the thesis. Firstly, this chapter includes the motivation and the aim of the thesis. Subsequently, building on the motivation and aim, the research questions are presented, followed by the approach used, and any relevant delimitations.

1.1 Motivation

The utilization of e-commerce has increased significantly in multiple countries in recent years. Between 2010 and 2022, the portion of members of the European Union who have ordered goods or services online in the last 12 months has increased from 53 % to 75 % [1]. The global business-to-consumer e-commerce sales are also growing rapidly, from 1336 billion USD in 2014 to a forecasted 6388 billion USD in 2024 [2].

While resulting in new business opportunities for online retailers, it also introduces new areas for them to compete in and problems that need to be solved. One such area to improve, which is also requested by stakeholders, is the environmental impact of shipping and packaging material usage [3]. For the competition, another dimension is the shipping cost, which affects the business strategy employed [4]. The way orders are packaged has an impact on both the environment and the shipping costs, meaning business value can be created by improving the methods to pack orders.

However, these aspects are rarely taken into consideration in the real world. At large, packaging and the selection of packages are still done manually, which can result in human errors and sub-optimal packaging [5]. These non-optimal packages can result in higher costs, larger environmental impacts, and missed business opportunities. Due to problems with meeting new requirements with the rise of e-commerce, many companies and warehouses are transitioning to more automated solutions [6].

Already, e-commerce purchases do generally have a lower environmental impact than purchases made in traditional retail, at least in some contexts [7]. Nevertheless, there are still improvements to be made to further decrease the negative impact, including in the packaging used for the orders [8]. Packaging is often a large part of greenhouse gas emissions and often accounts for more than 50 % of energy used for e-commerce orders [3, 9].

To meet environmental targets, regulators are also pushing for more sustainable and improved packaging. One example is the European Union, which has proposed legislation that

includes an article to decrease the amount of empty space in e-commerce order packaging to 40 % [10].

One essential part of improving business performance and mitigating environmental impacts through better packaging is to algorithmically find the optimal package, or set of packages, to fit the order into. To solve this problem, heuristics for the three-dimensional bin packing problem (3D-BPP) can be applied [11]. 3D-BPP is a classical problem in which the goal is to fit a set of items into a minimum number of bins. There are many variations to the problem, depending on the context in which is studied [12].

One approach to 3D-BPP is dividing it into two sub-problems, one for partitioning the order into groups of products and one for packing each of these groups into the best available package, where the latter is called the single bin packing problem (SBPP). For the specific use case of e-commerce, the problem is more general than the pure 3D-BPP as there might exist more than one package size and the number of bins is not the only metric relevant for evaluation. This makes the problem more complex and harder to solve, rendering a lot of earlier work on the 3D-BPP not directly applicable. The complexity of the problem combined with the real-time requirements of the e-commerce industry, make it a good case for developing heuristics. Due to heuristics giving non-exact results, the development renders the need for a thorough evaluation of their performance and applicability.

1.2 Aim

This thesis aims to create and assess order partitioning heuristics, which will be used to solve a part of 3D-BPP. The developed heuristics will be evaluated based on their impact on shipping costs and the environment.

Furthermore, this study aims to explore the practical value of these order partitioning heuristics in a real-time shipping logistics system, taking into account relevant time and data constraints.

1.3 Research questions

The following research questions assist in achieving the aim of the thesis. Research question RQ1 will evaluate the effectiveness of order partitioning heuristics from a theoretical perspective, to ensure that this approach is worth pursuing. In addition, research question RQ2 explores the real-world applicability of these heuristics while considering time and data constraints.

RQ1. How do different order partitioning heuristics affect shipping costs, material emissions and transport emissions?

RQ2. How can a heuristic for order partitioning be implemented so that it functions in a real-time system, given time and data constraints applicable to e-commerce logistics?

1.4 Approach

The approach used is to solve the problem by dividing it into two sub-problems, order partitioning, and single bin packing. The order partitioning sub-problem is solved using heuristics developed in this thesis, while the bin packing sub-problem is solved using a heuristic for the SBPP already developed by other researchers.

To evaluate the developed heuristics, the shipping costs of the proposed package selection are first analyzed. The costs will be based on the prices of a real-world shipping provider. Similarly, the environmental impact of the proposed package selection is analyzed by estimating the emissions from the material used and the transportation of the package.

1.5 External company

This thesis is conducted in collaboration with Skrym AB[3]. Skrym is a Stockholm-based start-up that provides optimization solutions for e-commerce retailers and other logistics companies, including optimization to the bin packing problem (BPP). Apart from technical, business expertise and access to a supervisor, Skrym also provides data from one of their customers, an e-commerce retailer, to be used for evaluation of the developed heuristics.

1.6 Delimitations

This thesis focuses on order partitioning as a part of bin packing with the specific goal of reducing shipping costs and emissions. Other important factors also play a role in the selection of packages, such as availability, shipping time, and delivery location. These are not considered in this thesis, as only the shipping cost and emissions are included.

The shipping costs and emissions are also calculated in a European and specifically Swedish context. These factors might differ in other geographical areas, as well as over time, due to changes in the shipping industry and the environmental impact of shipping.

Furthermore, only data from one e-commerce retailer is used to evaluate the developed heuristics. The characteristics of orders, products, and packages differ between retailers, so the results might not be generalizable to other companies.

Lastly, the developed heuristics are not compared to any current packing practices done by the company from which the data is from. This means that the thesis can not draw any conclusions as to the extent of potential improvements the developed heuristics can provide.

1.7 Thesis outline

To achieve the aim in section 1.2, the thesis relies on multiple key concepts. These key concepts are 3D-BPP and the heuristics for it, the shipping cost, material emissions, and transport emissions. These will be the main content of each of the chapters but other topics will also be presented and discussed.

First, in chapter 2, some theory behind cutting and packing (C&P) problems, the problem and relevant earlier work are presented. Furthermore, a similar problem, the open-dimension problem (ODP) is briefly discussed as well as some theory and mathematics behind order partitioning.

Subsequently in chapter 3, further relevant theories are presented. The main concepts discussed include the theory concerning shipping costs, material emissions, and transport emissions. Additionally, this chapter also incorporates additional theories related to the utilized methodology, heuristic analysis, real-time systems, and other minor concepts.

In chapter 4, the methodology used in this thesis is presented. This includes descriptions of the heuristics developed, the data used, and the evaluation of the heuristics. Again, especially interesting are the calculations of shipping costs and emissions, which are presented here.

The results of the thesis are presented in chapter 5. The heuristics are compared and the different metrics are grouped into sections. A comparison to the optimal solution is also included.

Finally, in chapter 6, the results, the method used, and the work in a wider context are discussed. This is followed by the conclusion of the thesis, which can be found in chapter 7.

[3] https://www.skrym.com

2 Cutting and packing problems

This chapter presents the theory and science behind the family of problems called C&P problems. Firstly, the problem family is introduced together with relevant nomenclature, followed by two to this thesis relevant problems, namely the bin packing and open dimension problem. Finally, some theories on the partitioning of orders are presented.

2.1 Cutting and packing problems

Multiple problems and problem variations belong to the family of C&P problems, all with different constraints and goals. To help with differentiating and naming the problems, a standardized language has been developed. It allows for more consistency and originates from research and development of typologies. Dyckhoff [13] constructed one such typology in 1990 and was the standard for a long time. Later, in 2007, Wäscher et al. [12] developed a new typology based on the work of Dyckhoff. What the two developed typologies share is the definition of C&P problems. They both state that all C&P problems involve two sets of elements, *large objects* and *small items*, and have the aim to fit the small items within the large objects. This should be done without the small items overlapping and the small items should fit within the large objects [13, 12]. In the context of an order, the small items are the products and the large objects are the packages.

To break down C&P problems, Wäscher et al. [12] developed five characteristics to use for defining problems: *Dimensionality*, *Kind of assignment*, *Shape of small items*, *Assortment of small items*, and *Assortment of large objects*. These criteria allow for the classification and naming of different C&P problems, resulting in standardized terminology. In the real world, the above problems often have additional constraints and characteristics. Examples include weight limits, orientation constraints, or stacking constraints. Incorporating one or multiple constraints leads to further variations of the problems, which might necessitate new heuristics [14].

2.2 Bin packing problem

A common C&P problem is BPP, where the goal is to distribute a collection of small items into a minimal number of larger objects of a fixed size. Using the typology from Wäscher et al. [12], this problem would have *input minimization* as *kind of assignment*, and a *strongly*

heterogeneous assortment of small items with *large objects of fixed sizes*. The other two criteria, namely the shape of the small items and the assortment of large objects, can vary between different variations of BPP [12]. The term residual BPP refers to a subset of problems where there is a strongly heterogeneous set of large objects, meaning that the large objects can differ in size and other characteristics [12]. This is the case with packing orders into packages, which is the main problem discussed in this thesis. Furthermore, BPP is NP-hard, meaning that no exact solution can be found in polynomial time [11].

There are many suggested solutions and heuristics for BPP, too many to cover in this thesis. Instead, this section will give some background on a few of them that are deemed relevant or influential.

2.2.1 Advanced heuristics for the bin packing problem

One of the seminal contributions to the research of 3D-BPP is the *ONEBIN* algorithm introduced by Martello et al. [11]. This work provides the first known exact solution to 3D-BPP by leveraging prior research on 1D and 2D bin packing. The authors employ a two-step strategy, utilizing a branch and bound algorithm for item-to-bin assignment, followed by the *ONEBIN* algorithm for SBPP. It should be noted, however, that the solution is limited in its scope as it does not accommodate rotations of items and requires identical bins. Apart from the exact solution the authors also introduce two heuristics. Although they do not provide optimal results, they exhibit faster computational times compared to the slower branch and bound approach. The algorithm was later found to only produce an optimal solution if new items could only be placed to the right, above, or in front of earlier placed items. This is equivalent to the way a robot would pack the items. However, there might be other places to put an item, such that the bin is utilized better, such as in an empty space surrounded by earlier placed items [15]. To solve this problem, Martello et al. [16] extended their original work to also solve these cases by improving the SBPP algorithm used.

Another well-regarded heuristic is the guided local search (GLS) algorithm, developed by Faroe et al. [17]. The algorithm depends on what they call features, which are attributes that describe the solution which can be used to evaluate it. By penalizing certain features, the algorithm can be re-run to find new local minima. It continues to do so for as long as it is allowed in the hope of continuously improving the solution. Similar to the *ONEBIN* algorithm, this heuristic is also limited by not allowing rotations of items. The heuristic shows good results relative to the *ONEBIN* algorithm developed by Martello et al. [11], especially given the much shorter run times [17].

Another heuristic is the *Greedy Adaptive Search Procedure* algorithm, presented by Perboli et al. [18]. It works similarly to the GLS algorithm as it iterates until an optimal solution is found or until a given time limit. The starting solution is found using any simpler, greedy heuristic. This first solution outputs an order for the smaller items to be placed. In the loop, the algorithm moves this list around to try to even out mistakes and bad orders, while simultaneously including a mechanism to go back to earlier iterations if no good solution is found. This heuristic shows great performance compared to earlier algorithms, while still giving good results [18].

Lastly, Crainic et al. [19] presents a heuristic that finds potential corner points inside the bins where it's possible to place items. This idea is similar to the *ONEBIN* algorithm by Martello et al. [16], which also uses corner points to find potential item placements. However, Crainic et al. [19] create another definition of the corner points to achieve better results than the heuristic version of the *ONEBIN* algorithm.

All these heuristics, while being influential in the field, have two limitations that make them unsuitable for the problem of packing orders. The first issue is that neither allows for rotations of the items. While Faroe et al. [17] state that their GLS algorithm can be generalized to allow for rotations, they do not provide any such algorithm. It would also incur a performance penalty, as more possible placements would need to be evaluated. The second issue is

that the heuristics assume that all the large bins are of the same size. This is not the case when packing orders, as the retailer will have a variety of box sizes available. However, this is not a limitation if these are used as an SBPP algorithm, as that problem only has a single bin to pack the items into.

2.2.2 Simpler heuristics for the bin packing problem

Among algorithms for BPP, the best-fit algorithm stands out as a relatively simple yet effective approach. In this method, each item is assigned to the bin with the smallest remaining space that can accommodate it. One such implementation is developed by Dube and Kanavathy [20].

To determine if an item fits in a bin, Dube and Kanavathy [20] employs a sub-algorithm based on pivoting. The objective of this sub-algorithm is to identify a pivot point that allows one of the corners of the new item to be packed into the bin. The approach is based on a brute-force strategy, where each possible point and rotation is tested, and the first valid solution is selected. The best-fit algorithm is implemented in two different Golang[4] packages. First, the author behind the paper released `bom-d-van/binpacking`[5]. Later, the code was modified to support floating point numbers to improve the granularity, and released as `gedex/bp3d`[6]. Both implementations support solving 3D-BPP with multiple bins with a worst-case time complexity of $O(n^3)$ and a best-case time complexity of $O(n^2)$ [20].

2.3 Open dimension problem

A similar problem to BPP is ODP. The difference between the two is that while the bins in the traditional BPP are fixed in size, ODP allows the bins to be variable in at least one dimension. Further, the items must be placed within the constraints of the fixed dimensions while the size of the variable dimensions should be minimized [12]. ODP could be generalized to work with more than one package. However, Wäscher et al. [12] notes that to their knowledge ODP with more than one package is rarely discussed in the literature. Some e-commerce retailers use machines that support adjusting the height of packages to product dimensions [21]. This makes ODP relevant for discussion.

Scheithauer [22] suggests an approximate algorithm for the traditional 3D-BPP that correlates with ODP definition detailed in section 2.3. More specifically, they formulate the problem as a 3D-BPP where the length and width of the bins are fixed, but the height is variable [22]. Their algorithm is based on the Next Fit Decreasing Width algorithm for 2D bin packing but packs the items in layers. This algorithm packs the items left-justified until the space left to the right is less than the width of the next item. Items are packed in a starting layer until the next item can't be accommodated, and a new layer is started. The algorithm is repeated until all items are packed. The collective height of all the layers is the height of the package. Scheithauer [22] further clarifies that the 2D-bin packing algorithm can be changed for different algorithms if desired.

Miyazawa and Wakabayashi [23] also suggested an appropriate algorithm for the traditional 3D-BPP corresponding with section 2.3, formulated in the same way as Scheithauer [22], namely fixed length and width but unbound height. The algorithm developed by Miyazawa and Wakabayashi [23] is based on multiple different algorithms. These are complex and out of the scope of this thesis, but the approach can be summarized as dividing the items into subgroups and then applying the algorithms to the appropriate subgroups or a combination of subgroups.

[4] `https://go.dev/`
[5] `https://github.com/bom-d-van/binpacking`
[6] `https://github.com/gedex/bp3d`

2.4 Order partitioning

A lot of earlier work on order partitioning has focused on the fulfillment of orders in multiple warehouses, most often due to inventory constraints. As an example, Josephine Xu et al. [24] presents a heuristic to do real-time improvements to the assignment of warehouses for an order using a local neighborhood search algorithm. They show that implementing this simple heuristic can decrease the number of shipments required, and thus the cost of order fulfillment.

A similar approach to the same problem is taken by Jasin and Sinha [25], where they develop a heuristic based on a deterministic linear program to optimize for the smallest shipping cost. They find that their heuristic performs very close to the optimal solution in multiple cases. However, the mathematical model is a bit simplified as shipping is assumed to be linear to the volume of the package and distance of the shipment, which is not always true.

Whereas the mentioned research focuses on the fulfillment of orders in multiple warehouses, this thesis focuses on partitioning orders for improved packaging in the same warehouse.

2.4.1 Number of partitions for an order

Partitioning an order into multiple groups can be related to the more studied concept of the partitioning of sets. A partition of a set is a grouping of its elements into non-empty subsets, in such a way that every element is included in exactly one subset [26, 27].

For example, given a set $\{a, b, c\}$, which can be seen as an order with three distinct products, there are five possible partitions, which are as follows

1. $\{\{a, b, c\}\}$
2. $\{\{a, b\}, \{c\}\}$
3. $\{\{a, c\}, \{b\}\}$
4. $\{\{b, c\}, \{a\}\}$
5. $\{\{a\}, \{b\}, \{c\}\}$.

However, orders differ from pure mathematical sets in the sense that they can contain multiple products of the same type. For instance, the order $\{a_1, a_2, b\}$, which still has three products but where two products are of the same type, can be partitioned into the following four unique partitions:

1. $\{\{a_1, a_2, b\}\}$
2. $\{\{a_1, a_2\}, \{b\}\}$
3. $\{\{a_1, b\}, \{a_2\}\}$
4. $\{\{a_1\}, \{a_2\}, \{b\}\}$

The one partition that is not included here, but in the partitions of pure sets is $\{\{a_2, b\}, \{a_1\}\}$. That is because a_1 and a_2 are two products of the same type, meaning that the partitions $\{\{a_1, b\}, \{a_2\}\}$ and $\{\{a_2, b\}, \{a_1\}\}$ are seen as equal, and thus only one of them is included. The product type is controlled by the stock keeping unit (SKU) which is a unique identifier for a product, and is assigned to each product in a warehouse [28]. Any difference between products, such as color, results in different SKUs [28]. This means that two products might have the same dimensions and weight, but still are assigned different SKUs. As a result, there might still be partitions that are considered equal for the packing problem but still are included.

The consequence of potential duplicates is that the problem of partitioning orders is more complex than partitioning sets. Thus, a function for the number of possible partitions will need multiple variables and is difficult to construct. Instead, this section presents the upper and lower bound for this function, along with an exact function to generate the sets implemented in pseudocode presented in appendix A.

Upper bound

The upper bound for the possible number of partitions is given by the case when all the products in the order have separate SKUs. When this is the case, there are no "duplicate" sets we need to disregard. As all items are unique, this is the same as the number of partitions of a set.

To understand the number of partitions of a set, the Stirling number of the second kind needs to be introduced. Stirling numbers of the second kind, in this thesis simply referred to as Stirling numbers, describe the number of ways to partition a set of size n into k non-empty subsets, most often written as $\left\{ {n \atop k} \right\}$ [29]. By using the definition from Graham et al. [29], Stirling numbers can be defined recursively as

$$\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k \cdot \left\{ {n-1 \atop k} \right\} \tag{2.1}$$

with the base cases $\left\{ {n \atop k} \right\} = 1$ for $n = k$ and $\left\{ {n \atop k} \right\} = 0$ for $k > n$. Using this recursive formula, the first values in the Stirling number series can be seen in table 2.1.

Table 2.1: Stirling numbers of the second kind for $0 \leq n, k \leq 9$

$\boldsymbol{n}$	$\left\{ {n \atop 0} \right\}$	$\left\{ {n \atop 1} \right\}$	$\left\{ {n \atop 2} \right\}$	$\left\{ {n \atop 3} \right\}$	$\left\{ {n \atop 4} \right\}$	$\left\{ {n \atop 5} \right\}$	$\left\{ {n \atop 6} \right\}$	$\left\{ {n \atop 7} \right\}$	$\left\{ {n \atop 8} \right\}$	$\left\{ {n \atop 9} \right\}$
0	1									
1	0	1								
2	0	1	1							
3	0	1	3	1						
4	0	1	7	6	1					
5	0	1	15	25	10	1				
6	0	1	31	90	65	15	1			
7	0	1	63	301	350	140	21	1		
8	0	1	127	966	1701	1050	266	28	1	
9	0	1	255	3025	7770	6951	2646	462	36	1

The Stirling numbers can be used to define the number of partitions of a set of size n, known as the Bell number B_n. As an example, the number of ways to partition a set of size 3 is equal to the number of ways to first partition it into one subset, plus the number of ways to partition the set into two subsets, plus the number of ways to partition the set into three subsets. This means that the bell number can be defined as the sum of Stirling numbers as follows

$$B_n = \sum_{k=1}^{n} \left\{ {n \atop k} \right\}. \tag{2.2}$$

This number sequence was described by Bell [26], and the notation was introduced by Becker and Riordan [27]. It is also included in The On-Line Encyclopedia of Integer Sequences as sequence number A000110 [30]. Apart from the recursive definition in equation 2.2, the Bell numbers can also be defined recursively as

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k. \tag{2.3}$$

where $\binom{n}{k}$ is a binomial coefficient. The first values in the Bell number series can be seen in table 2.2.

Lower bound

While the upper bound is achieved when all products are unique, the lower bound is achieved when all products are identical. This corresponds to finding all possible partitions of an integer n, where n is the number of products in the order. A partition of a number n is a group of positive integers that together sum to n [31]. One element in this set matches one subset of products of that size. For example, one possible partition of 5 is $1 + 2 + 2$, which would correspond to a partitioning of one group with one product and two groups with two products each.

The number of partitions of a number n is given by the partition function $p(n)$. Although there is no simple formula for the partition function, generating functions do exist [31]. While a normal function gives the exact value, the value sought can be found in the coefficients of the polynomial the generating function creates [32]. The partition numbers have a sequence in The On-Line Encyclopedia of Integer Sequences with sequence number A000041 [33]. The first values in this sequence can be seen in table 2.2.

Table 2.2: Lower and upper bound of the number of partitions for orders of size n

n	1	2	3	4	5	6	7	8	9	10
Lower bound	1	2	3	5	7	11	15	22	30	42
Upper bound	1	2	5	15	52	203	877	4140	21 147	115 975

Code to calculate order partitions

An algorithm can be employed to systematically generate all possible partitions of the products within the order, as demonstrated in algorithm A.1. Subsequently, these partitions can be filtered to eliminate any duplicate partitions that may arise from containing identical products. The specific Python implementation of this algorithm can be found in algorithm A.1. It is noteworthy that the length of the resulting list of partitions will fall within the range defined by the lower and upper bounds that have been previously described in table 2.2.

3 Analysing packing heuristics

In this chapter, all the background and theories used to analyze the heuristics and their results are presented. This includes how to estimate shipping costs and emissions, theory on algorithmic analysis, general methods for analysis together with other concepts used in the thesis, such as real-time systems and some useful equations.

3.1 Impact of packaging

An important question is whether or not online shopping is more environmentally friendly than shopping in physical stores. In a literature review by Buldeo Rai et al. [7], they conclude e-commerce purchases generally have a lower carbon footprint than purchases made in stores. However, the impact varies a lot depending on the context, especially the car dependency of the area. The authors also note that the calculations do not account for new consumer patterns that might arise due to online shipping replacing physical store shopping.

Buldeo Rai et al. [7] also conclude that packaging has a quite big impact on the overall carbon footprint for both store and online purchases. This is also supported by Pålsson et al. [9] who show that packaging constitutes a large part of the energy consumption of an order, especially for e-commerce. The fraction does differ between product categories but is often above 50 % for e-commerce orders. This means that improved packaging can make online shopping less energy-intensive and therefore more environmentally friendly.

The extensive usage of cardboard as a material for packaging in e-commerce also has negative environmental effects due to its relatively high greenhouse gas emissions. Therefore, the maximal utilization of the packages is important to mitigate this effect [34].

Furthermore, Pålsson et al. [8] specifically includes fill rate as a variable to analyze the environmental impact of packaging in a supply chain. Apart from the environmental benefits, the fill rate also has a significant positive impact on the economic performance of the chain.

3.1.1 EU packaging regulation

In June 2020, the European Union published an initiative to reduce packaging waste [10]. The proposal outlines new EU requirements that focus on reducing packaging waste and making packaging easier to recycle [10].

For the scope of this thesis, the most relevant part is the article *Obligation related to excessive packaging* (*Article 21*) where they propose secondary packaging fill rate requirements. *Obligation related to excessive packaging* specifies that economic operators who supply products to a final distributor or an end user in grouped packaging, transport packaging, or e-commerce packaging must ensure that the fill rate is a minimum of 60 % [10]. They further specify that filling materials such as paper cuttings, air cushions, and bubble wraps are considered empty space. However, the article *Packaging minimization* (*Article 9*) states that the weight, volume, and packaging should be minimized when it's considered reasonable to do so, i.e., taking the packaging's safety and functionality into account [10]. Meaning that economic operators using sales packaging as e-commerce packaging are exempted from this obligation if they comply with the requirements in *Packaging minimization*.

3.2 Shipping costs

The assessment of shipping costs is a multifaceted issue that involves various factors such as the choice of delivery strategies and the geographic location of the recipient. To tackle this complexity, this thesis focuses on the cost of delivery for retailers using standard delivery strategies offered by PostNord, a prominent logistics provider in Sweden. To provide a comprehensive analysis, the heuristics developed in this thesis are designed to be evaluated based on their real-life impact, and to not overcomplicate this we limit the scope to packages sent within Sweden.

In this thesis, we concentrate on three services provided by PostNord, namely *MyPack Collect*, *MyPack Home Small*, and *MyPack Home*, all of which are used as data sources for our analysis. These services differ in terms of the delivery location, with *MyPack Collect* delivering the package to a pick-up point and *MyPack Home* delivering the package directly to the customer's home [35]. The pricing for these services is based on the volumetric weight of the package, which is calculated by assuming a weight of 280 kg per cubic meter. The approach of using volumetric weight allows carriers to account for the cost of transporting lighter packages with larger dimensions. It should be noted that the prices mentioned in this thesis exclude taxes.

3.2.1 MyPack Collect

MyPack Collect is a service for delivering packages to designated pick-up points, with the pricing structure dependent on both the actual weight and volumetric weight of the package [35]. To ensure the service's optimal use, several constraints have been established, including a maximum weight limit of 20 kg and a maximum length of 150 cm [35]. Additionally, the total length and circumference of the package cannot exceed 300 cm. Customers should also take note of an extra fee of 160 SEK per package if the package dimensions exceed 120 cm in length and/or 70 cm in width [35].

In the table 3.1 the pricing for *MyPack Collect* is presented. The pricing structure is determined by weight or volumetric weight, whichever is greater. For packages with volumetric weights exceeding 50 kg, the price is 11.90 SEK/kg.

3.2.2 MyPack Home Small

MyPack Home Small is a service designed for the home delivery of smaller packages. The pricing scheme for this service is based on both the weight and volumetric weight of the package, with the volumetric weight being used to calculate the price if it is higher than the actual weight [35]. To take advantage of this service, it is important to note that the package cannot exceed specific size limitations, with maximum dimensions of 34 cm in length, 24 cm in width, and 7 cm in height. Table 3.2 offers a clear overview of the price points associated with different package weights. For packages exceeding 3 kg, an extra fee of 40 SEK is added, along with the weight-based price of 43.50 SEK/kg.

Table 3.1: PostNord pricing for *MyPack Collect*

	Weight [kg]	**Price** [SEK]
Physical weight	3	152
	5	182
	10	245
	15	297
	20	351
Volumetric weight	25	400
	30	427
	35	472
	50	580
	> 50	$11.90^{\dagger}$

† [SEK/kg]

Table 3.2: PostNord pricing for *MyPack Home Small*

Physical weight [kg]	**Price** [SEK]
0.25	53.50
0.5	59.90
1	69.50
2	83.50
3	104.50
> 3	40 + 43.50/kg

3.2.3 MyPack Home

The *MyPack Home* service is a widely used service for the delivery of larger packages, and its pricing is determined based on both weight and volumetric weight, as well as a price group (PG) that is dependent on the origin and destination of the package. To present the data in a digestible format, table 3.3 is delimited to only showing PGs based on the origin postal code *694 91*. This postal code corresponds with the demographic center presented in section 3.5.1, and is used as an example due to the vast combinations of postal codes. The complete pricing information for all PGs can be found in the PostNord price lists for service agreement customers[7]. Similar to the other services, the volumetric weight of a package is considered when its volumetric weight exceeds its actual weight. Certain restrictions apply to the use of this service, including a maximum length of 175 cm and a maximum length plus circumference of 300 cm, as well as a weight limit of 35 kg per package. Additionally, packages larger than 120 cm or wider than 70 cm will incur an extra fee of 160 SEK per package.

3.3 Material emissions

One part of the total emissions of the order comes from the material used for the packaging. This includes the whole lifecycle of the material, including production, recycling, and disposal. In e-commerce, the most common package material is corrugated cardboard [9].

[7] PostNord [36]

Table 3.3: PostNord pricing for *MyPack Home*. Unit is [SEK/package].

	Weight [kg]	**PG 0**	**PG 1**	**PG 2**	**PG 3**	**PG 4**	**PG 5**
Physical weight	3	126	129	136	151	198	231
	5	133	139	146	179	231	271
	10	142	149	156	244	299	342
	15	153	159	204	298	359	409
	20	162	169	235	351	418	472
	25	298	314	364	400	482	548
	31.5	424	447	524	599	722	820
	35	507	533	565	582	629	681
Volumentric weight	50	661	696	738	757	818	885
	75	13.40†	14.10†	14.90†	15.40†	16.70†	18.00†
	100	12.90†	13.16†	14.30†	14.70†	15.90†	17.30†
	300	12.30†	12.90†	13.70†	14.10†	15.30†	16.50†
	> 300	11.60†	12.20†	13.00†	13.40†	14.50†	15.70†

† [SEK/kg]

3.3.1 Corrugated cardboard

According to the European Federation of Corrugated Board Manufacturers (FEFCO)[8] 2021 annual report, the average *grammage* (weight per area) of the cardboard in Europe is 503 g/m^2 [37].

Brogaard et al. [38] have gathered comprehensive data on the life cycle carbon footprints of various materials, among which corrugated cardboard is included. The mean carbon impact of corrugated cardboard for primary production, which refers to the creation of new material, was found to be 1.14 kg CO_2-eq/kg. The range of values for this process was found to be relatively large, with a minimum value of 0.49 and a maximum value of 2.45 kg CO_2-eq/kg. As for secondary production, which involves recycling the material, the mean carbon impact was established to be 0.82 kg CO_2-eq/kg, with a minimum and maximum range of 0.31 and 1.26 kg CO_2-eq/kg, respectively. It is noteworthy that the primary production statistics consisted of 17 datasets, whereas the secondary production statistics included 10 datasets.

3.3.2 FEFCO 0201

FEFCO has developed a standard for the design of corrugated cardboard boxes, called the FEFCO code. These designs are the most commonly used in the industry and help with identifying the different types. One of the simpler designs is FEFCO 0201, which is a simple box with a flat bottom and a lid [39]. An example of the box can be seen in figure 3.1 and its corresponding flat design can be seen in figure 3.2.

Disregarding minor flaps and cutouts for easier assembly, the area of the box is given by

$$A = (w + h)(2l + 2w) = 2(lw + lh + wh + w^2) \tag{3.1}$$

where l, w, and h are the length, width, and height of the box, respectively. This can easily be seen from the flat design in figure 3.2. The flaps and cutouts are not included, as they are relatively small and have unspecified dimensions. Therefore, the equation 3.1 should be seen as an estimate.

[8]European Federation of Corrugated Board Manufacturers is a non-profit trade association for the corrugated cardboard industry. `https://www.fefco.org/`.

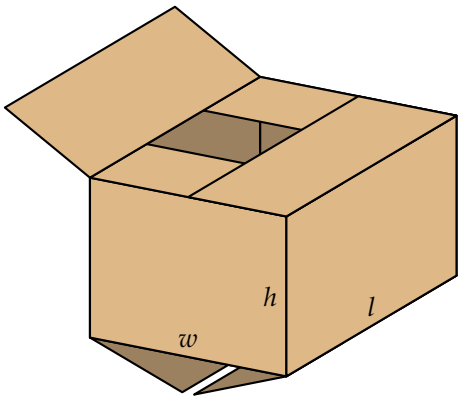


Figure 3.1: FEFCO 0201 box example

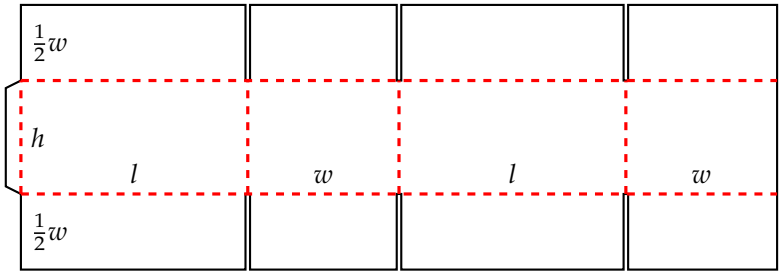


Figure 3.2: FEFCO 0201 box design

3.4 Shipping emissions

Another part of the environmental impact of the e-commerce order comes from shipping it to its final destination. One of the methods for calculating the emissions from the shipping of a package is the method provided by the Network for Transport Measures (NTM)[9]. Below, NTM's method for calculating the emissions from shipping a package and relevant terminology is presented.

3.4.1 The Network for Transport Measures (NTM)

NTM is a non-profit organization based in Sweden with the objective of providing a set of principles and methods for the calculation of the environmental impact of different transport modes, including road cargo [40]. They consist of more than 100 members with interests in the transport sector, such as government agencies, transporters, transport-heavy companies, and universities. Both Skrym and Linköping University are members[10].

NTM publicized a disclaimer stating that the data provided by them is not supported by any research or statistical analysis. Instead, it is based on approximations from the members of the NTM workgroup "Goods and Logistics" who primarily used volumetric weight for analyzing cargo with low density in Swedish domestic conditions. They further add that the data regarding Swedish truck transportation is based on approximations from Boulter and McCrae [41] [42].

[9] https://www.transportmeasures.org/
[10] https://www.transportmeasures.org/members/

3.4.2 Load capacity utilization

The load capacity utilization (LCU) is a very important factor when calculating the environmental impact of a shipment using NTM's method. This is defined as the ratio between the capacity of the vehicle and the cargo loaded on the vehicle. This can be measured in several different units, such as the percentage of pallet space used, the percentage of volume used, or the percentage of weight used. It can also be measured in the percentage of volumetric weight used, a term which is introduced in section 3.2 which is often used in the industry [43].

3.4.3 Transportation systems

When discussing transportation from a storage facility to the end customer different systems can be used. NTM defines these three different systems as shared, dedicated (frequent), and dedicated (single) [44]. The dedicated systems transport packages directly from the company warehouse to the customer, resulting in shorter transit times and lower transport distances but a lower LCU [44]. The shared system is complex but also more applicable to the e-commerce industry. Therefore, this thesis will focus on the shared system and not discuss the other systems any further.

Shared (Integrated) transport systems

Shared integrated transport systems involve the use of several different vehicles and logistic centers. Each shipment is integrated with other shipments and sent through a complex network of logistic centers. The route for each shipment is therefore almost always longer, a sacrifice made to achieve higher LCU [42]. The complexity makes it difficult to track the exact route for each shipment ahead of time. NTM [42] states that the most accurate route data will be obtained from the integrator. And in the cases where that is not possible, they argue that the route data can be estimated using the following two steps:

1. Find the road distance between the two largest city centers close to the source and destination of the shipment. At these city centers, we place imaginary logistic centers as that is their most likely position. Then assume that the shipment is sent using a semi-trailer truck for that distance.

2. Find the road distance between the city centers and the start and end points of the shipment. Then assume that a medium-sized vehicle is to transport the shipment from the imaginary logistic centers to and from the actual start and end points.

To find out LCU of the different vehicles, NTM [42] suggests using real data for the specific vehicle as the capacity utilization varies greatly depending on different regions, seasons, and companies. If no data is available, NTM [42] suggests using the following values:

- 70 % of the loading capacity for vehicles employed between terminals e.g. the vehicles used for step 1.

- 50 % of the loading capacity of the vehicles used to/from the integrating terminals e.g. the vehicles used for step 2.

Concluding that shared integrated transport systems are complex and difficult to model as many different vehicles and logistic centers are involved in the same shipment. This results in higher transit times and longer transport distances, but high utilization of capacity and resources [44].

3.4.4 Vehicle type

The vehicle type used for transporting the package affects the emissions from the transportation. Not only do the vehicles differ in terms of size, capacity, and fuel consumption, but also in LCU. For better estimations, the values should be dependent on the actual vehicle type used. However, NTM presents a set of default values for 14 different types of vehicles. The vehicle nomenclature has a corresponding The Handbook Emission Factors for Road Transport (HBEFA)[11] nomenclature, where fuel consumption data is available [45]. The other values that are of interest for the calculations are the load capacity and LCU, which can be seen in table 3.4.

Table 3.4: NTM vehicle types. Data from NTM [45]

NTM Nomenclature	**Load capacity**	**Load capacity utilization**		
		Weight	Volume	DimW
Light commercial vehicle - Pick-up	0.60 t	20 %	25 %	30 %
Light commercial vehicle - Van	1.50 t	20 %	25 %	30 %
Rigid Truck $\leq$ 7.5 t	5 t	40 %	40 %	50 %
Rigid Truck 7.5 t – 12 t	6 t	40 %	40 %	50 %
Rigid Truck 12 t – 14 t	9 t	40 %	40 %	50 %
Rigid Truck 14 t – 20 t	12 t	40 %	40 %	50 %
Rigid Truck 20 t – 26 t	15 t	40 %	40 %	50 %
Truck with Trailer 14 t – 20 t	12 t	40 %	40 %	50 %
Truck with Trailer 20 t – 28 t	16 t	40 %	40 %	50 %
Truck with Trailer 28 t – 34 t	22 t	50 %	50 %	60 %
Truck with Trailer 34 t – 40 t	26 t	50 %	50 %	60 %
Truck with Trailer 40 t – 50 t	33 t	50 %	50 %	60 %
Truck with Trailer 50 t – 60 t	40 t	50 %	50 %	60 %

3.4.5 Road type

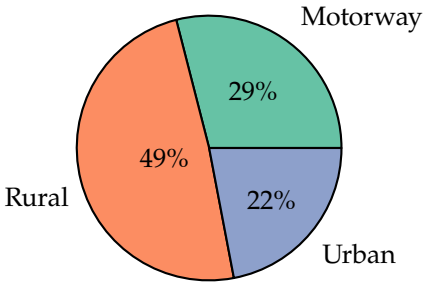
The road type affects fuel consumption. It affects the speed and the number of stops. There are many different road types in Sweden, and NTM divides them into motorway, urban, and rural. They further state that the road type for a specific transport route can be determined using data from a geographical information system. If that data is not available then the road type can be estimated using the national averages [43]. They provide a national average for Sweden which can be seen in figure 3.3. It's worth noting that the road type can be different for different sections of a transport route. This means that it's very dependent on the length of the transport and the route that is taken. NTM further specifies topology as a driving factor in fuel consumption but gives no way of using it in the calculations other than saying that they use data from HBEFA [43].

3.4.6 LCU fuel consumption

According to NTM, fuel consumption is heavily dependent on the weight of the vehicle, including its cargo. As the weight of the cargo increases so does the rolling resistance and the force required to put the mass in motion. In turn, this means that the fuel consumption is

[11] The Handbook Emission Factors for Road Transport is an organization with support from multiple countries and the European Union that provide emissions factors for multiple vehicle categories. `https://www.hbefa.net/`.

Figure 3.3: Swedish national average road type



dependent on LCU. NTM provides a formula to describe the relationship between LCU and fuel consumption.

$$FC_{LCU} = FC_{empty} + (FC_{full} - FC_{empty}) \cdot LCU \tag{3.2}$$

where FC_{LCU} is the fuel consumption for a given LCU, FC_{empty} is the fuel consumption for an empty vehicle, FC_{full} is the fuel consumption for full LCU. The fuel data is based on reports in HBEFA and then processed by NTM [43].

3.4.7 Vehicle emissions

Given the default LCU values, fuel consumption, fuel characteristics data, and road type usage, the emission from the transportation for each vehicle type can be calculated. For a subset of the vehicle types, NTM presents these emissions for Swedish vehicles. The data can be found in table 3.5 and is based on data from HBEFA and International Council on Clean Transportation (ICCT)[12] [45].

Table 3.5: Baseline vehicle emissions in Sweden. Data from NTM [46]

NTM Nomenclature	**LCU**	**Emissions**
	%	kg CO_2-eq/km
Light commercial vehicle - Van	20	0.163
Rigid Truck $\leq$ 7.5 t	40	0.227
Rigid Truck 7.5 t – 12.0 t	40	0.353
Truck with Trailer 20.00 t – 28.00 t	40	0.491
Truck with Trailer 34.00 t – 40.00 t	50	0.608
Truck with Trailer 50.00 t – 60.00 t	50	0.828

3.4.8 Assign vehicle emissions to individual shipments

As multiple shipments are sent in the same vehicle when it comes to integrated transportation, the total emissions of the vehicle also need to be shared between the shipments. NTM suggests that the environmental impact of the shipment is divided amongst the orders in the shipment based on how much of the vehicle's capacity the shipment uses. The capacity utilization of the shipment can be calculated using different metrics. Similarly to LCU, common measures are weight, volume, and volumetric weight. To accommodate both heavy and large shipments, the greatest of the physical and volumetric weights can be used for shipping emissions. This

[12] https://theicct.org/

means that very light but large shipments still get assigned the emissions for the space it takes, while heavy but small shipments still get assigned the emissions from their weight [42]. This results in equation 3.3, where the total emissions of a shipment (SE) is calculated as

$$SE = \sum_{r \in R} c_r \cdot \frac{w}{LC_r \cdot LCU_r} \tag{3.3}$$

where R is the set of all legs of the route, c_r is the vehicle emissions for the vehicle that is used on leg r, w is the maximum of the physical and volumetric weight of the shipment, LC_r is the load capacity of the vehicle on leg r, and LCU_r is LCU of the vehicle on leg r.

3.5 Population distribution of Sweden

This section will serve as a basis for shipping cost and shipping emissions estimations as the population distribution of Sweden is useful for simulating order destinations and deliveries. However the population distribution is not uniform so to get a more granular view, the demographic center of Sweden and the Statistics Sweden (SCB) grid division is introduced below.

3.5.1 Demographic center of Sweden

The demographic center refers to the geographic location of a country that is determined by the distribution of its population. Specifically, it is determined by calculating the average location of all citizens within a country. This computation can be visualized as a map, whereby a small weight is placed on each citizen's location, and the point at which the map is balanced is the demographic center. It is noteworthy that the demographic center also represents the point to which the population on average has the shortest geodetic distance [47].

In the case of Sweden, the demographic center is located within the municipality of *Hallsberg* in *Örebro County*. This result was obtained by SCB [47], who utilized population data from the year 2006 to calculate this value.

3.5.2 Population statistics of grid cells

The government agency SCB has divided Sweden into a grid of 114 800 cells, each with a size of 1 km × 1 km. For each of these cells, the registered population is also available, resulting in a quite detailed population distribution of Sweden. This grid also follows the INSPIRE directive from the European Union [48].

This data is available online and is released in a GeoPackage[13] format, which is a standard format used for storing geospatial data. It is stored as an SQLite[14] database, meaning it is simple to directly use in any programming language without needing any advanced programs [49].

However, the coordinate system is not the standard *WGS84*, but instead, the Swedish grid system *SWEREF 99 TM* [48]. To work with this format, there are tools available such as the Python library *sweref99*[15].

3.6 Algorithmic analysis

There are multiple ways to analyze algorithms and their performance. One popular approach is to use asymptotic analysis for both time and space complexity [50].

[13] `https://www.geopackage.org/`
[14] `https://sqlite.org/`
[15] `https://pypi.org/project/sweref99/`

3.6.1 Time and space complexity

The most prevalent method for measuring the time complexity of algorithms is the *big-O notation*, also known as *asymptotic notation*. This analysis examines how the running time scales as the input size grows. More specifically, it only considers the largest term of the running time expression, seeing as this is the dominating factor [50] for large inputs. We use the definition from Sipser [50] in definition 3.1.

Definition 3.1 (Big-O notation)**.** Given the functions $f, g : \mathbb{N} \mapsto \mathbb{R}^+$, we say that

$$f(n) = O(g(n)) \tag{3.4}$$

if $c, n_0 \in \mathbb{Z}^+$ exists such that

$$f(n) \leq cg(n) \tag{3.5}$$

for for every $n \geq n_0$ where $n \in \mathbb{Z}$.

Big-O notation can also be used for analyzing how much memory space is required to run the function, given the size of the input. Just like time, Big-O for space analysis is useful for understanding what happens when input sizes grow large [50].

3.6.2 P and NP

A central piece to understanding P and NP is the concept of *polynomial time algorithms*. Using definition 3.1, these are algorithms whose time complexity is $O(p(n))$ for some polynomial function p and input size n. All other algorithms are instead called *exponential time algorithms*. As exponential functions dominate polynomial functions, polynomial time algorithms scale better than exponential as n grows large [51].

Using this notation, P is defined as the set of problems solvable by a deterministic Turing machine in polynomial time, while NP is the set of problems solvable by a non-deterministic Turing machine in polynomial time. The distinction between the two lies in the utilization of deterministic or non-deterministic Turing machines. Simplified, deterministic Turning machines sequentially perform calculations, one operation at a time. In contrast, non-deterministic Turing requires some guess of a solution which it can then check similarly to its deterministic counterpart. So when referencing polynomial time solution using a non-deterministic Turing machine, it's the time complexity of the latter deterministic checking stage that is referenced. This means that all problems in NP can be verified in polynomial time [51].

We can further introduce two classes that help describe problems in terms of their complexity. The first one is NP-hard problems. If every problem in NP can be transformed into a problem in polynomial time, we say that problem is NP-hard. NP-complete problems on the other hand are problems that are NP-hard, but still live up to the definition of a NP problem. That means that NP-hard problems are not necessarily NP problems, but are at least as hard as the NP problems [51].

All these complexity classes can be seen in figure 3.4. Note that the figure shows the case for if $\mathsf{P} \neq \mathsf{NP}$, something that is not known to be true, and it might be that $\mathsf{P} = \mathsf{NP}$ [51].

3.7 Real-time systems

Using the definition from Burns and Wellings [52], a real-time system is a system for which response time is of importance. Most computer systems have some real-time requirements since there exists an expectation of timely responses when input is given. Furthermore, to differentiate between systems with regards to how important a response in a specified period is, the terminology *hard* and *soft* real-time systems can be used. A hard real-time system is

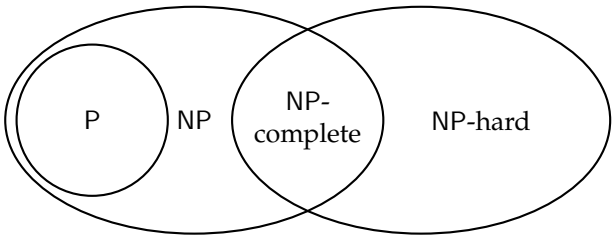


Figure 3.4: Complexity classes

one where any defined expected response times are crucial, while soft indicates that while not satisfactory, infrequent longer responses are tolerated [52].

In addition, another useful categorization is *reactive* versus *time-aware* systems. Time-aware systems work in absolute time, meaning that the time of execution is of interest. On the other hand, reactive systems are expected to work with relative time instead, meaning the time between input and output is important [52].

3.8 Useful definitions, equations, and formulas

This section introduces the key definitions, equations, and formulas that are essential for understanding the theoretical underpinnings of the research presented in this thesis. These formal descriptions provide a more precise and consistent language to describe the concepts and methods used.

3.8.1 Fill rate

The definition of fill rate used in this thesis is based on work by Svanes et al. [53]. They define the fill rate as the percentage of the total volume filled with products or packaging materials. Furthermore, the fill rate is defined on multiple hierarchical levels, such as the relation between product to primary packaging or the relation between primary packaging to secondary packaging, etc [54]. This definition is also used by other papers discussed in this thesis, such as Pålsson et al. [8].

For the scope of this thesis, the relation used is the one between the products and the package in which they are shipped. Any filling material used to protect the orders during shipping is considered empty space and does therefore not contribute to the fill rate. This is the same as used by the European Union in their proposal on legislation for packaging and packaging waste, further described in section 3.1.1. Below are more formal definitions of fill rate which are used in this thesis.

Definition 3.2 (Fill rate)**.** Let p be a package, and Q_p be a set of products placed in p. The fill rate $\Phi(P)$ is defined as the ratio of the volume of the products to the volume of the package, as follows:

$$\Phi(p) = \frac{\sum_{q \in Q_p} V(q)}{V(p)} \tag{3.6}$$

where $V(p)$ is the volume of the package p and $V(q)$ is the volume of the product q.

However, when considering a set of packages, the fill rate can be defined in two different ways. The first is the average fill rate, which is defined in definition 3.3.

Definition 3.3 (Average fill rate)**.** Let P be a set of packages. The average fill rate $\Phi_{avg}(P)$ for P is defined as the average of the fill rates of all packages in P, as follows:

$$\Phi_{avg}(P) = \frac{1}{|P|} \sum_{p \in P} \Phi(p) \tag{3.7}$$

The other way to define the fill rate for a set of packages is the cumulative fill rate, as given by definition 3.4.

Definition 3.4 (Cumulative fill rate)**.** Let P be a set of packages. The cumulative fill rate $\Phi_{cum}(P)$ for P is defined as the ratio of the total volume of the packages to the total volume of the products in the packages, as follows:

$$\Phi_{cum}(P) = \frac{\sum_{p \in P} V(p)}{\sum_{p \in P} \sum_{q \in Q_p} V(q)} \tag{3.8}$$

where Q_p is the set of products in package p, $V(p)$ is the volume of the package p and $V(q)$ is the volume of the product q, as defined in definition 3.2.

The difference between the average and cumulative fill rates lies in the way they account for the amount of air in the packages. The cumulative fill rate considers the actual amount of air in all packages, while the average fill rate only considers the mean package fill rate. Therefore, larger packages with the same fill rate as smaller packages will have a larger impact on the cumulative fill rate, as they contain more air.

It is worth noting that since the products placed in a package cannot be larger than the package itself, the fill rate will always be in the range of $[0, 1]$. As a result, the average and cumulative fill rates will also always be in $[0, 1]$.

Different dimensions

The fill rate is defined for three dimensions since it is the volume that is of interest. However, a fill rate can also be viewed as the utilization of the length, width, and height. This can help understand how much of the package is used for the products and how much is used for air.

For example, to comply with the EU regulation mentioned in section 3.1.1, the fill rate needs to be 60 % on average. To get the space utilization in one dimension, the cube root can be used, giving a required utilization of $\sqrt[3]{60\,\%} \approx 84.34\,\%$. However, for height adjustable packaging, which results in a height utilization of 100 %, there are only two dimensions that will have worse utilization. This means that the length and width need a specific utilization of $\sqrt{60\,\%} \approx 77.45\,\%$ to comply with the 60 % EU goal. Algorithms that are suitable for height-adjustable packaging are further discussed in section 2.3.

Simply translating a fill rate in three dimensions to a specific utilization in one dimension is just an ideal case. Products are not placed in a perfect cuboid that uses length, width, and height the same. Instead, the three-dimensional placement of the packages is more complex, but the idea of converting can help with the visualization.

3.8.2 Haversine formula

The Haversine formula is a mathematical function that can be used to calculate the distance between two points on a sphere. Let the coordinates of two points on a sphere be defined by their latitudes, φ_1 and φ_2, and longitudes, λ_1 and λ_2, respectively. Let d be the distance between these two points, and let r be the radius of the sphere. The central angle θ between the two points can be expressed as $\theta = d/r$. The Haversine formula provides a relationship between the coordinates and the central angle, as given by:

$$\operatorname{hav}(\theta) = \operatorname{hav}\left(\frac{d}{r}\right) = \operatorname{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2 - \lambda_1) \tag{3.9}$$

where $\operatorname{hav}(\theta)$ denotes the Haversine function. Using the identity $\operatorname{hav}(\theta) = \sin^2(\theta/2)$, the inverse of the Haversine function can be expressed as $\operatorname{archav}(\theta) = 2\arcsin(\sqrt{\theta})$. Therefore, the distance d between the two points can be calculated using the following expression:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{3.10}$$

This formula can be used to compute the distance between any two points on the surface of a sphere, such as the Earth. In that case, the radius is a constant of $r = 6371\,\text{km}$. However, the earth is not a perfect sphere, meaning that equation 3.10 will not give the exact value for d but a close approximation [55].

3.9 Method for analyzing heuristics

Analysis of heuristics is often done in a rather ad-hoc manner, with little to no structure. This is problematic since it makes it difficult to ensure replicability and unbiased analysis of the results. Montgomery [56] first proposes a guideline for the design and analysis of experiments. This guideline was later adapted by Barr et al. [57] to better suit designing experiments for a computer environment and analyzing heuristics. Having a guideline for designing experiments and analyzing heuristics helps researchers maintain a scientific structure and avoid pitfalls. The main steps in the process of analyzing heuristics are listed below.

- **Defining goals** - What is the purpose of the experiment?
- **Choosing performance measures and factors** - How is quality measured and what factors affect the measures?
- **Designing the experiment and execution** - How is the experiment designed and executed?
- **Data analysis and conclusions** - How is the data analyzed?
- **Reporting results** - How is the computing experiments reported?

This section will go through each of these steps in more detail and acts as a basis for the methodology used in this thesis.

3.9.1 Defining goals

Barr et al. [57] define the goals of heuristics, research reports, and experiments as follows: Heuristics should be fast, accurate, robust, simple, highly impactful, generalizable, and innovative. Research reports should be revealing and theoretical. And the goal of computer experiments is usually to determine the performance and characteristics of a heuristic [57].

They also provide different options for setting the goals of a heuristic; comparing it to others or self-comparison. If other heuristics are solving the same problem, then one suggested way of measuring the performance of a heuristic is to compare it to the best competitor. If no other heuristics are available then it can be evaluated by setting a greedy solution as a baseline. The authors also suggest that to better understand the behavior of a heuristic it can be effective to create different variants with isolated changes to different factors. The investigator can then run simulations to see how the changes affect the performance of the heuristic.

3.9.2 Choosing performance measures and factors

Barr et al. [57] state that heuristics are dependent on two different categories of variables: *Performance measures* and *affecting factors* where performance measures are the result and affecting factors are the variables that affect the performance measures such as problem, algorithm, and environmental factors. They also emphasize the importance of choosing the right performance measures and that performance measures usually focus on solution quality, computational effort, and robustness.

Performance measures

According to Barr et al. [57], the quality of a solution can be measured in many ways but the speed at which it converges to the optimal solution and how close it comes to the optimal solution or if no optimal solution exists, how close it comes to a close upper-bound is usually good measures. Plotting the quality of the solution over time is suggested as an insightful way to visualize the performance of a heuristic [57].

The computational effort is also mentioned by Barr et al. [57] as a valuable performance measure and can be measured in terms of time, memory, or the number of iterations. When measuring the computational effort they make a point of not only measuring the total effort but also the effort for each phase of the algorithm. The authors also suggest extending the graph of the solution quality over time to include the computational effort as it can provide insight into how effort and solution quality correlate. Robustness is also mentioned as a performance measure and is the measure of how well a heuristic performs on different problem instances [57]. Barr et al. [57] suggest measuring robustness with the metrics of the variance and failure rate. Further specifying that it's important to test how different factors affect the robustness of a heuristic. They also make a note of not fine-tuning the parameters of a heuristic to improve the results of specific problem instances if the specified problem cannot be detected and changes made automatically. Barr et al. [57] emphasize the importance of choosing the right way to calculate the performance measures and that it can be difficult to do across multiple computer systems.

McGeoch [58] says that performance measures should be used when the research questions can not be answered directly and that the research questions hint about what performance measures are interesting. However, she also states that other factors should guide experiment planning. McGeoch [58] lists helpful advice for choosing performance measures and factors to study. She points out the importance of performance measures to have low variance within problem instances compared to the variance between different problem instances. Using examples from bin packing research, she states that according to experts in data analysis, there is a big difference between measuring the difference between measurements and the ratio between measurements. She further suggests that one can identify new performance measures by using variance reduction techniques.

Affecting factors

Barr et al. [57] state that there are three main categories of factors that affect performance measures: problem, algorithm, and environmental factors. All of which must be taken into consideration [57]. Some questions that they mention should be taken into account are: Which factors should be studied and which should be ignored? Are they fixed or are they varied? The authors give some examples for categorizing the affecting factors as follows:

- **Factors to study** - Problem size, heuristic, number of processors.
- **Factors to fix** - Problem class, stopping rule, computing environment.
- **Factors to ignored** - Distribution of problem costs, System job mix

Problem factors

Barr et al. [57] classify problem factors as the dimension, structure, and distribution of the problem. They stress the importance of including the effects of problem factors in the analysis as it's important to understand the behavior of a heuristic. Moreover, they also state the importance of testing for different parameter distributions, data presentation order, and problem sizes.

Algorithm factors

Barr et al. [57] state that many heuristics often employ different heuristic strategies and the factors that determine what heuristic strategy to use are categorized as algorithm factors.

Environment factors

According to Barr et al. [57], the environmental factors are all the factors that affect the performance of the heuristic that is not part of the algorithm or the problem. These factors include various hardware, software, system, and programmer-related conditions that can affect the performance of competing algorithms.

3.9.3 Designing the experiment and execution

Barr et al. [57] state that experimental design is the process of planning an experiment to ensure that the appropriate data will be collected. A good experimental design is unbiased, achieves the experimental goals, clearly demonstrates the performance of the tested process, uncovers performance reasons, has a justifiable rationale, generates supportable conclusions, and is reproducible [57].

Choosing design

Barr et al. [57] argue that the best way to obtain reliable and objective experimental results for heuristic methods is through the use of well-known statistical experimental design methods, such as the design of experiments (DOE) approach. Further, they explain DOE as a process for collecting and analyzing data with an established statistical model, and that it ensures that the collected data can be analyzed to reach valid and objective conclusions. DOE is based on the principles of replication, randomization, and blocking, and it is designed to minimize testing efforts and maximize information acquired [57]. DOE is routinely employed in the physical sciences, engineering, and medicine, but there is a broad range of accepted practices in the computing and mathematical sciences [57].

Selecting test problems

According to Barr et al. [57], the choice of test problems is essential in the experimental design of heuristic methods. Real-world problems are valuable for assessing the effectiveness of a given approach, while special test problems can be important for testing particular performance features and response patterns. Problem sets may be drawn from practical settings or created with problem-generation software, where the generation process should be clearly described and the newly generated problems archived for use by other researchers [57].

Executing the experiment

In the data-collection step Barr et al. [57] suggests, that the investigator must follow the experimental design and document the process details. Randomization and maintaining a uniform computing environment are important to reduce variability and the influence of unknown factors. If job execution times are affected by the system's job mix, data should be collected during lightly loaded or dedicated periods [57].

3.9.4 Reporting results

Following Barr et al. [57], the phase of analyzing and drawing conclusions within an experiment entails the conversion of collected data into meaningful information. This process involves the utilization of both statistical and nonstatistical techniques. Additionally, it is suggested that analysts should carefully consider the experimental goals and make key tradeoffs,

aiming to identify the factors that contribute to performance. Figures and graphs can be used to gain deeper insights into the data and its relationships [57]. Furthermore, the application of statistical methods is crucial for assessing the strength of relationships between different factors and performance measures. Once the analysis of the data is complete, the results are interpreted, and conclusions and recommendations are formulated. According to Barr et al. [57], this phase represents the culmination of all the preceding planning and implementation activities, ultimately determining the overall worthiness of the work. Finally, the last step involves documenting the experimental details and findings and effectively communicating them to the research and practice communities [57].

Reproducibility

In scientific research, it is important to ensure reproducibility, which necessitates meticulous documentation of experimental procedures. As indicated by Barr et al. [57], one way to achieve this is to provide a comprehensive account of the algorithms and their implementation, as well as any relevant parameters, data structures, and problem instances employed during heuristic testing. To promote openness and collaboration among researchers, non-proprietary problems should be shared with others, and the provision of code can enhance the scientific value of the research. Overall, a rigorous approach to documenting experiments is essential for scientific progress and knowledge dissemination [57].

Computing environment

When reporting on experimental results, Barr et al. [57] emphasizes the significance of documenting the test-environment factors that can influence the empirical performance of a heuristic method. These factors include the model and manufacturer of the computer, the number and types of processors, the size and configuration of memories, the interprocessor scheme and communication times, the operating system, programming languages and compilers, system load, computing resources, and machine speed. Additionally, the computing resources required by the algorithm should be detailed and expressed as a function of problem parameters [57].

Timing

Barr et al. [57] underscore the significance of accurate timing reports in heuristic method experiments. To achieve this, the authors need to document the measurement methods and distinguish between user, system, and real times. In particular, provide timings for aspects, such as overhead operations, each heuristic stage, total time, time to the best solution, and calibration routines. Barr et al. [57] recommend clearly stating at what points in the code the timings were recorded. Additionally, it's important to be clear about which operations are included and which are excluded [57].

Quality of solution

Barr et al. [57] highlight the importance of measuring the difference between a heuristic solution and an optimal solution to demonstrate the effectiveness of a proposed heuristic. The measurement can be conducted directly by comparing both solutions or indirectly by benchmarking against known performance standards such as lower and upper bounds. Additionally, Barr et al. [57] recommend providing insights into how solution quality behaves when problem instances increase in size or complexity. This serves as an essential aspect of evaluating algorithmic performance and aids researchers in developing robust heuristics that can handle real-world problems with varying complexities effectively [57].

Parameter selection

To provide a comprehensive report, Barr et al. [57] recommends that the algorithm control parameters and their corresponding values for each problem instance should be specified. Moreover, the process used in selecting these parameter values must also be clearly stated, including any differences observed in parameter values across instances. Additionally, it is essential to provide evidence that supports the robustness and generalizability of the identified parameter values while assessing the significance and interplay of each parameter. A time estimate for fine-tuning algorithms should also be provided in such reports. To ensure accuracy and objectivity in selecting appropriate parameter values, a scientific approach utilizing statistical analysis should be employed during this exercise [57].

Variability

To increase the validity and reliability of experimental results, Barr et al. [57] suggests that careful consideration should be given to DOE. This can involve running longer experiments or conducting them on dedicated or lightly loaded machines to minimize variability. Additionally, Barr et al. [57] recommend that a variety of problem classes be tested with multiple instances within each class. If pseudo-random number generators are employed in the experiment, it is important to test the code on each instance with different seed initialization for optimal validity and precision [57].

Analysis and interpretation

Barr et al. [57] emphasize the importance of conducting a comprehensive analysis and interpretation of experimental results to derive valid conclusions. This involves going beyond the mere presentation of tables displaying running times. In particular, researchers need to consider central tradeoffs such as solution quality versus computational effort, as well as other factors like time constraints versus problem size and robustness versus quality [57].

Benefit-cost analyses are also crucial for understanding the performance of heuristics in various contexts. Such analyses might involve weighing time and memory tradeoffs against each other, or identifying how individual stages contribute to overall heuristic effectiveness [57].

Furthermore, any unexpected or anomalous results that arise should be given special attention during analysis. Researchers should strive to explain these findings if possible while also leaving them open for future investigation by others in the field [57].

4 Method

In this chapter, the method for solving the problem and analyzing the solution is described in detail. The methodology adheres to the well-established guidelines formulated by Montgomery [56] and subsequently modified by Barr et al. [57]. These guidelines are described in section 3.9 and by adopting these established principles, the research conducted in this thesis aims to maintain high standards of quality and reliability throughout all stages of its implementation. This chapter will first describe the dataset, followed by the structure of the simulation tool and test environment. After this, the method of setting and evaluating goals is described in detail. Finally, all the necessary algorithms, estimation, and pseudocode for the heuristics are presented.

4.1 Dataset

The main dataset in this thesis is provided by Skrym and is based on real-world data from one of their customers that sells medicine and other health products in Sweden. In total, this dataset consists of 2 028 842 orders, 52 583 products, and 4 package sizes.

4.1.1 Orders

Each order contains a set of products, each with a unique SKU and a quantity. All orders have at least one product, and there is at least one of each of the products in the order. Table 4.1 presents meta-information about the orders.

Table 4.1: Order statistics

Metric	Min	Max	Mean	Median	Stddev
No of products	1	1000	4.62	3	4.71

4.1.2 Products

The dataset consists of 52 583 products. While products can be of any shape, they are all assumed to be cuboids. The cuboid shape is estimated by taking the cuboid closure of the actual product shape and representing it by the width, height, and length.

We define the height (h) as the smallest dimension of the product, the width (w) as the second smallest, and the length (l) as the largest dimension. The volume V is then defined as $V = h \cdot w \cdot l$. Some metadata regarding the products can be found in table 4.2.

Table 4.2: Product statistics

Metric	Unit	Min	Max	Mean	Median	Stddev
Height	mm	1	850	49.55	45	32.05
Width	mm	2	980	85.75	70	54.39
Length	mm	10	9000	158.50	143	97.06
Volume	dm^3	0.000 495	2817.75	1.32	0.40	14.83
Weight	g	1	26 000	279.44	104	785.82

Some of these dimensions are very large and do not fit in the available packages. We however decided to keep them as they are. Removing these products would force us to make decisions on which to keep and not, which could introduce bias into the dataset. Any products not fitting in any package will be logged by the testbed.

4.1.3 Packages

All four package sizes are height-adjustable, meaning that the height of the package can be cut by a machine to fit the contents. This machine then supports a certain height range and a predefined base area. These dimensions can be seen in table 4.3.

Table 4.3: Package dimensions

Package	Length	Width	Min height	Max height
	mm	mm	mm	mm
Small	225	150	20	95
Medium	315	220	55	165
Large	383	288	142	286
ExtraLarge	578	387	201	395

Most previous research on 3D-BPP has used packages with fixed dimensions. However, taking the height range of the packages into account is briefly discussed in sections 2.3 and 3.8.1. Nevertheless, to simplify the calculations and make this work more comparable to other approaches, the packages are converted to fixed dimensions. The conversion splits each of the height-adjustable packages into three packages with the same base area but with different heights. One is assigned the minimum height of the range, one the maximum height, and the last is assigned the middle point between the two. This results in a dataset of 12 packages which can be seen in table 4.4.

4.1.4 PostNord terminals

Skrym has compiled a list of PostNord's terminals used for shipping packages, which are scattered across Sweden. There are 8 terminals in total, and the complete list can be found in

Table 4.4: Fixed package dimensions

Package	Length mm	Width mm	Height mm	Volume dm^3
Small Min	225	150	20	0.68
Small Mid	225	150	57	1.92
Small Max	225	150	95	3.21
Medium Min	315	220	55	3.81
Medium Mid	315	220	110	7.62
Medium Max	315	220	165	11.43
Large Min	383	288	142	15.66
Large Mid	383	288	214	23.61
Large Max	383	288	286	31.55
ExtraLarge Min	578	387	201	44.96
ExtraLarge Mid	578	387	298	66.66
ExtraLarge Max	578	387	395	88.36

table 4.5. It should be noted that PostNord has more terminals for shipping letters and pallets, but these are the only ones that are used for shipping packages.

Table 4.5: PostNord package terminals

Name	Address	Latitude	Longitude
PostNord Segeltorp	Häradsvägen 253	59.285 °N	17.944 °E
PostNord Järfälla	Elektronikhöjden 4	59.408 °N	17.836 °E
PostNord Malmö	Vevaxelgatan 8	55.602 °N	13.100 °E
PostNord Landvetter	Kurirvägen 2	57.689 °N	12.160 °E
PostNord Jönköping	Bordsvägen 2	57.674 °N	14.159 °E
PostNord Örebro	Vältgatan	59.245 °N	15.112 °E
PostNord Östersund	Hägnvägen 15	63.165 °N	14.757 °E
PostNord Luleå	Cementvägen 7	65.615 °N	22.045 °E

4.2 Simulation tool

The testbed, or simulation tool, is divided into two main sections. Firstly, there is the *optimization* section, that's tasked with generating package proposals from orders using the heuristics. Secondly, the *analysis* section, is then tasked with gathering statistics from the generated package proposals, and calculating performance metrics such as environmental impact, and shipping cost.

4.2.1 Optimization section

The optimization section is responsible for executing the heuristics and generating the package proposals. The connections between the components in the optimization section can be seen in figure 4.1. In the optimization section, the main component is the *test runner*, which accesses

the dataset described in section 4.1 and calls the heuristic using this data, and records the result, including the performance of the heuristics.

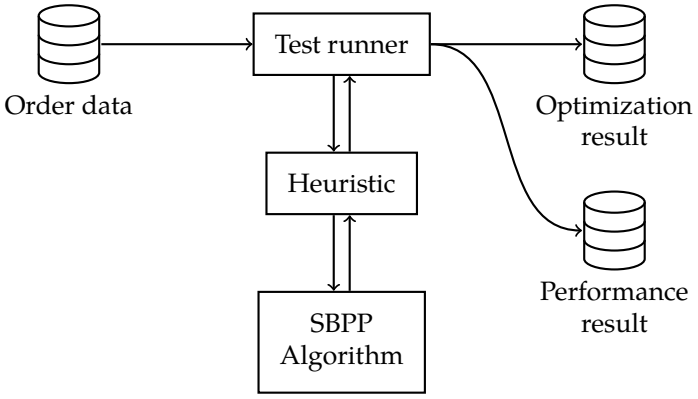


Figure 4.1: Optimization section flow chart

It's the test runner that calls the heuristics defined in section 4.12. The design is modular, meaning that as long as the heuristics follow the same interface, they can be used in the test runner. The same goes for the SBPP algorithm where any algorithm that follows the same interface can be used. The choice of the SBPP algorithm for this thesis is explained in section 4.9.

4.2.2 Analysis section

The analysis section, depicted in figure 4.2, constitutes the second phase of the simulation tool. This section calculates the cost and emissions of the package proposals, using the optimization results from the heuristics. It is important to note that the analysis components exclusively focus on the analysis requiring advanced calculations and emissions assessments, rather than encompassing the entirety of the results. There are endpoints for fetching simpler performance metrics e.g. fill rate, run time, and failure rate, but these are not included in the analysis section. Sections 4.6 to 4.8 provides an elaboration on the pivotal analysis components, namely cost calculation, material- and shipping emissions estimation.

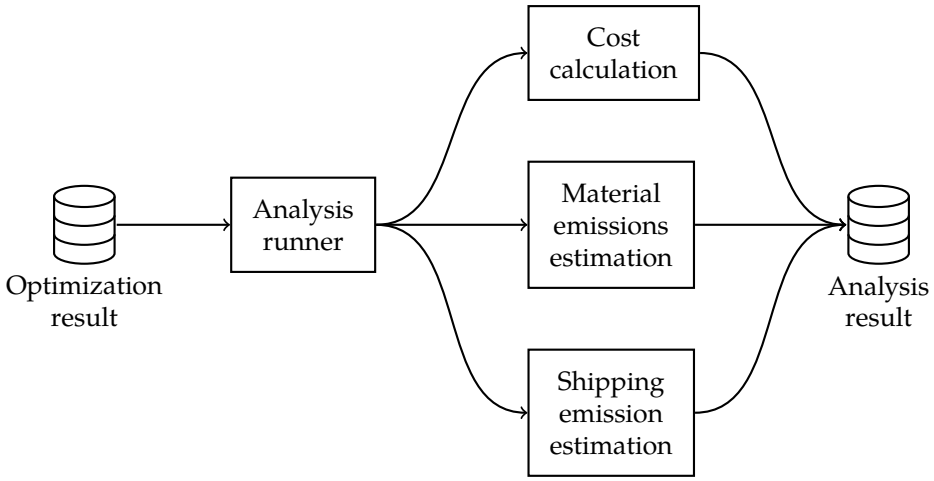


Figure 4.2: Analysis section flow chart

4.2.3 Cloud computing

Some of the calculations are time-dependent, meaning that the performance of the machine running the simulation tool can affect the results. Performance-sensitive calculations were therefore run in the cloud. As described in section 3.9.4, the idea is to achieve more reproducible and consistent results by containerizing the test runs and running them in the cloud.

The simulation tool is written in Go[16] using Encore[17]. Encore has support hosting the application on Google Cloud Platform (GCP) using their Cloud Run platform [59].

The configured capacity of the containers is 1 vCPU and 256 MB of memory on their first generation of Cloud Run. All processors used on GCP has support for simultaneous multithreading. This technique adds support for a processor to execute instructions from multiple threads at one time [60]. Each of these simultaneous threads is called a vCPU on GCP [61]. The containers are also set to only allow one concurrent request at one time to avoid any potential decreased performance due to sharing CPU resources. The GCP region is `europe-north1` which is located in Finland.

4.3 Defining goals

The primary objective of our research is to evaluate the efficacy of heuristics in order partitioning. As discussed in section 3.9.1, heuristics should meet several criteria, such as being fast, accurate, robust, simple, highly impactful, generalizable, and innovative.

To evaluate the effectiveness of the heuristics in addressing the research questions, we classify our performance metrics into two categories: impact and real-world applicability. The impact category addresses item RQ1, which explores the effectiveness of the heuristics in minimizing the shipping costs, and environmental impact. Whereas the real-world applicability category addresses item RQ2, which explores the practical utility of the heuristics by evaluating factors such as computational effort, robustness, and generalizability.

Specifically, we measure robustness by quantifying the failure rate of the heuristic, which provides insight into its ability to consistently provide solutions. The computational effort is assessed using execution time and order analysis.

Apart from computational effort and robustness, other critical factors contribute to the real-world applicability of a heuristic such as generalizability. A heuristic's ability to handle multiple problems is indicative of its generalizability. However, as the number of problems a heuristic can handle impacts its failure rate, we include generalizability as a part of the robustness metric.

4.3.1 Measures evaluation

The evaluation of heuristics is an essential aspect that requires careful consideration when addressing complex problems. As highlighted in section 3.9.1, it is imperative to have suitable measures for evaluating heuristics, along with a way to compare the measures. However, when dealing with a problem that lacks existing heuristics, it becomes necessary to create baseline heuristics that can be used as a benchmark for comparison. In this thesis, we have opted to employ simple and greedy heuristics as our baseline heuristics.

Additionally, we compare the different heuristics to optimal solutions, which are obtained from exhaustive searches on a subset of the problem instances. Heuristics involve a trade-off between quality and computational effort. This is especially true for heuristics that show incremental improvements in quality at the cost of increased computational effort. Therefore results of such heuristics must be weighed against the computational effort required to achieve them. It is worth emphasizing that the problem we discuss in this thesis currently lacks any

[16] `https://go.dev/`

[17] Encore is a development framework for writing backends and configuring infrastructure in Go. `https://encore.dev/`

existing solutions. However, by establishing appropriate measures and baseline heuristics, we can still effectively evaluate the performance of our heuristics and make informed decisions regarding their effectiveness. Comparing the heuristics to optimal solutions and baseline heuristics, as well as evaluating the computational effort is supported by the literature, as discussed in Quality of solution & Analysis and interpretation in section 3.9.4.

4.4 Selection of problem instances

The process of selecting diverse and representative test cases is of utmost importance when using heuristic methods for real-world scenarios, as discussed in section 3.9.3. The use of real-world problems is crucial for evaluating the effectiveness of a heuristic. To accomplish this, the testing methodology involves utilizing a vast dataset of real-world orders, further detailed in section 4.1.

A seed value is generated to ensure the reproducibility of the experiment results. If no seed value is provided in the input, the current time is used as the seed. The batches are sorted randomly based on the generated seed value using a slice offset. This also aligns with the discussed guidelines on reducing variability presented in section 3.9.4.

This approach enables the creation of a diverse set of test cases that encompass different scenarios while also providing an efficient means of evaluating the heuristics by running them in batches. The use of a seed value ensures reproducibility and randomization of the results, as discussed in section 3.9.3. The methodology employed enables the re-running of the same experiment with the same test cases in the same order, if necessary. Further, when comparing multiple heuristics the same parameters are used, ensuring that they are evaluated using the same test problem instances.

4.5 Placement of warehouse

In both cost calculation and shipping emissions, the geographic location of the shipment's source and destination plays a crucial role. The cost of delivery for certain services is determined by the sender and recipient locations, while the distance between these locations directly impacts emissions.

While e-commerce companies may have multiple warehouses in different locations for logistical reasons, for simplicity's sake, we opted to use a single warehouse for all orders. To determine the optimal placement of this warehouse, we relied on the demographic center of Sweden, as presented in section 3.5.1. The demographic center of Sweden is situated in the municipality of Hallsberg, south of Örebro. For the exact location of the warehouse, the zip code 694 91 and coordinates 59.07 °N and 15.11 °E were chosen. Assuming that customer distribution follows the distribution of the population of Sweden, using the demographic center facilitates the minimization of the distance between the warehouse and the customers.

4.6 Shipping cost calculation

We calculate the cost of delivery using the delivery services described in section 3.2, as well as the proposed package options generated by the heuristics. Each delivery service can be categorized into two groups: home delivery and service point delivery. Home delivery refers to packages being delivered directly to the customer's doorstep, while service point delivery refers to packages being delivered to a service point for customer pickup. Furthermore, the home delivery options consist of two services, namely *MyPack Home Small* detailed in section 3.2.2 and *MyPack Home* detailed in section 3.2.3.

The delivery service for *MyPack Home* is the only one taking into account the location of the recipient and the sender. This results in packages being assigned, and different PGs. Due to the extensive number of combinations between the two locations, the pricing for packages

is only analyzed when they are sent from postal code 694 91. This is the location for our imaginary warehouse, as discussed in section 4.5. Using this postal code as the site of an imaginary logistics center, we get the pricing table seen in table 3.1.

The dataset of orders used in this thesis does not contain any information about the destination for that order. Therefore, we cannot simply use the PGs directly to calculate the cost of delivery. Instead, a weighted average of the PGs can be used to achieve an average shipping cost for each order. The weights are determined by the number of zip codes in each PG. For this to give a correct average cost, the number of inhabitants of each zip code would have to be the same. Sadly, the population of each zip code is not known, meaning we can't get an exact weighted average. However, using the number of zip codes should give an approximation of the population distribution of the PGs. The number of zip codes connected to each PG and the weight of each PG can be seen in table 4.6.

Table 4.6: PostNord weighted average pricing for *MyPack Home* based on weight

	PG 0	**PG 1**	**PG 2**	**PG 3**	**PG 4**	**PG 5**	**Total**
Count	3133	982	239	3445	1066	1601	10 466
Percentage	29.94 %	9.38 %	2.28 %	32.92 %	10.19 %	15.30 %	100 %

By taking the weights in table 4.6 and applying them to the PGs from table 3.1, new pricing tables based on our imaginary warehouse can be created. These pricing tables can be seen in tables 4.7 and 4.8, for weight and volumetric weight respectively.

Table 4.7: PostNord pricing for *MyPack Home* based on weight

Weight	**Weighted average**
kg	SEK/package
3	158.13
5	180.09
10	223.14
15	262.60
20	300.03
25	391.57
31.5	576.97
35	574.49

Table 4.8: PostNord weighted average pricing for *MyPack Home* based on volumetric weight

Volumetric weight	**Weighted average**
kg	SEK/kg
50	747.90*
75	10.34
100	14.53
300	13.93
> 300	13.20

*Unit is SEK/package

As an order can be comprised of multiple packages, each with separate delivery services, the cost of delivery is calculated by adding up the cost of each package. To provide more detailed cost estimates for the various deliveries, we calculate all possible combinations of delivery services and packages and sum up the cost for each combination. However, it is also worth noting that, as stated in the respective detailed delivery service sections, there are limitations on the use of certain delivery services. As a result, some combinations of delivery services and packages are not possible and therefore are not considered in our cost calculation. This method allows us to calculate averages, as well as the minimum and maximum cost of delivery for each order.

4.7 Material emission estimation

To estimate the material emissions from the packaging used in the results optimized, a simple approach is used. The basic idea is that the material emissions are proportional to the surface area of the packaging. As the studied packages are assumed to be of the FEFCO 0201 standard, the surface area A is $A = 2(lw + lh + wh + w^2)$, where l, w and h are the length, width, and height of the package, respectively.

Given the surface area, the weight of the packaging is simply calculated by multiplying the surface area with the grammage, which is the weight per unit area. The grammage of corrugated cardboard, which is the most common material used for packaging, is $0.503\,\mathrm{kg/m^2}$ as found in section 3.3.1.

With the weight of the packaging, the carbon footprint can easily be determined by multiplying the weight with the carbon footprint of corrugated cardboard per kilogram of produced material. As found in section 3.3.1, for secondary production, corrugated cardboard has a mean carbon footprint of $0.82\,\mathrm{kg\,CO_2\text{-}eq/kg}$. This results in equation 4.1 for calculating the material emissions for a given order.

$$ME = 2(lw + lh + wh + w^2) \cdot 0.503 \cdot 0.82 \tag{4.1}$$

In the equation, ME stands for *material emissions* and is measured in $\mathrm{kg\,CO_2}$-eq. As the number of packages is limited, the material emissions can be pre-computed. The material emissions can be seen in table 4.9.

Table 4.9: Material emissions for the packages

Package	**Length** mm	**Width** mm	**Height** mm	**Material emissions** g CO_2-eq
Small Min	225	150	20	73
Small Mid	225	150	57	89
Small Max	225	150	95	105
Medium Min	315	220	55	169
Medium Mid	315	220	110	202
Medium Max	315	220	165	236
Large Min	383	288	142	331
Large Mid	383	288	214	386
Large Max	383	288	286	442
ExtraLarge Min	578	387	201	651
ExtraLarge Mid	578	387	298	758
ExtraLarge Max	578	387	395	865

Table 4.10: Assigned vehicles for each leg

Leg	Vehicle name	Capacity	LCU		Footprint
			Weight	DimW	
		t	%	%	kg CO_2-eq/km
1	Rigid Truck 7.5 t to 12 t	6	40	50	0.353
2	Truck with Trailer 20 t to 28 t	16	40	50	0.608
3	Rigid Truck 7.5 t to 12 t	6	40	50	0.353

4.8 Shipping emission estimation

The method used in this thesis for estimating shipping emissions comes from NTM and is presented in detail in section 3.4. Below, the choices required to estimate the shipping emissions are explained and presented. This includes the choice of vehicles and the distances for the vehicles to cover. Lastly, these values are summarized in an equation that is used in the analysis section of the simulation tool to estimate the emissions from shipping.

4.8.1 Vehicle choice

Each of the three legs of the route can be driven by different vehicles. Exactly what type of vehicle and its emissions can only be determined if the transporter supplies that information. However, as we do not have access to that information, we have to make assumptions. NTM recommends assigning a rigid truck for the first and last leg, and a semi-trailer for the middle leg.

Choosing from the set of available vehicles in table 3.4, we picked "Rigid Truck 7.5 t to 12 t" for legs 1 and 3. It has a load capacity of 6 t, a dimensional weight LCU of 50 %, a weight LCU 40 % and a carbon footprint of 0.35 kg CO_2-eq/km. For leg 2, we used the semi-truck "Truck with Trailer 20 t to 28 t". It has a capacity of 16 t, a dimensional weight LCU of 50 %, a weight LCU of 40 % and a carbon footprint of 0.61 kg CO_2-eq/km. These choices can be seen in table 4.10.

4.8.2 Leg distances

To calculate the shipping emissions using the method from NTM as described in section 3.4, the leg distances must be known. However, as our data does not come with any recipient addresses, the leg distances must be estimated.

By separating the route of the shipment into three legs, as recommended by NTM for integrated transport systems. The first leg will be the distance from the warehouse to the terminal closest to the warehouse, the second leg will be from that terminal to the terminal closest to the recipient, and the third leg will be from that terminal to the recipient. In our case, the available terminals are the PostNord terminals as described in section 4.1.4. The average distances for each leg can be seen in table 4.11.

The first leg is the easiest, as this is always the same path, namely from the warehouse to the closest terminal. As described in section 4.5, the warehouse is placed in Hallsberg at the coordinates 59.07 °N and 15.11 °E. The closest terminal is therefore "PostNord Örebro", which is located at 59.25 °N and 15.11 °E. To then calculate the distance between the warehouse and the terminal, we used the Haversine formula as described in section 3.8.2, giving us the geodetic distance between the points. This resulted in a distance for leg 1 of 19.95 km.

To calculate the distance for leg 2, we calculated the distance from the "PostNord Örebro" terminal to all terminals and took and weighted average between them, where the weights are the number of people who has that terminal as their closest terminal. Similarly to leg 1,

we used the Haversine formula to calculate the distance between the terminals. To get the number of people that has a certain terminal as their closest, we used the 1-kilometer grid data from SCB as described in section 3.5.2, which includes population data for each grid cell. Again, the Haversine formula is used to get the distances. The result of this weighted average is a distance of 238.30 km. The calculation of the weighted average distance for leg two can be described as

$$d_2 = \frac{\sum_{g \in G} p_g \cdot h(t_w, t_g)}{\sum_{g \in G} p_g} \tag{4.2}$$

where G is the set of all 1 km × 1 km grid cells, p_g is the population in grid cell g, h is the haversine formula described in section 3.8.2, t_w is the terminal closest to the warehouse and t_g is the closest terminal to grid cell g.

Finally, for the third leg, the idea is similar to that of leg 2, but instead of the distances being between terminals, the distance is between the closest terminal and the recipient. However, the distances are still weighted by the population in each 1 km × 1 km grid. This resulted in an average distance of 61.38 km. Similarly to leg 2 and equation 4.2, the calculation of the weighted average distance for leg three can be described as

$$d_3 = \frac{\sum_{g \in G} p_g \cdot h(t_g, g)}{\sum_{g \in G} p_g} \tag{4.3}$$

where, G is the set of all grid cells, p_g is the population in grid cell g, h is the haversine formula described in section 3.8.2 and t_g is the terminal closest to the grid g.

Table 4.11: Average leg distances

Leg	**Distance** km	**Percentage** %
1	19.95	6.24
2	238.30	74.55
3	61.38	19.20
Total	319.63	100

4.8.3 Resulting estimation equations

Given the vehicles and leg distances presented above together with the carbon dioxide emissions per kilometer for each vehicle, we can now calculate the carbon dioxide emissions for each shipment. In equation 4.4, the emissions from each leg are calculated and summed together, based on equations presented in section 3.4. With the values set, the equation becomes

$$\begin{aligned} SE = {} & 0.353 \cdot 19.95 \cdot \frac{\max(w_{phys}, w_{dim})}{6000 \cdot LCU_1} \\ & + 0.608 \cdot 238.30 \cdot \frac{\max(w_{phys}, w_{dim})}{16\,000 \cdot LCU_2} \\ & + 0.353 \cdot 61.38 \cdot \frac{\max(w_{phys}, w_{dim})}{6000 \cdot LCU_3} \end{aligned} \tag{4.4}$$

where LCU_i is LCU for the physical or dimensional weight depending on which is larger for the package. This means that the denominator and nominator always use the same capacity metric for a package.

4.9 Single bin packing problem algorithm

To develop heuristics, we need to have a method to solve SBPP. The algorithm used in the thesis is the three-dimensional best-fit algorithm with pivoting described in section 2.2.2. The algorithm supports multiple bins, but as we implement the partitioning heuristic ourselves, we only use it with a single bin as input.

The SBPP algorithm takes a list of products and a package as arguments and returns a boolean noting if the products fit in the package or not. The SBPP part of the algorithm has a time complexity of $O(n^2)$ where n is the number of products to be packed as described in section 2.2.2.

4.9.1 Find smallest package helper function

To help with finding the package with the smallest volume in which the products fit, we develop the following helper function. It assumes that the packages, P, are sorted according to their volume in ascending order, then simply iterates over the packages until it finds a package that can fit the products. This will be the package with the lowest volume in which the products can be packaged and it is returned by the function. The pseudocode is presented in algorithm 4.1.

Algorithm 4.1 Find smallest package helper function pseudocode

```
Require: P is sorted in ascending order according to volume
 1: function FindSmallestPackage(P, Q)
 2:     for p ∈ P do
 3:         fits ← sbpp(p, Q)
 4:         if fits then
 5:             return p
 6:         end if
 7:     end for
 8:
 9:     return null
10: end function
```

For each iteration, it needs to run the SBPP algorithm, which has a time complexity of $O(n^2)$, where $n = |Q|$ is the number of products. As this simply iterates linearly over all packages, the helper function has m iterations, resulting in a time complexity of $O(mn^2)$ where $m = |P|$ is the number of packages. Since the packages P are sorted, FindSmallestPackage could also be implemented using a binary search, in case it would have a time complexity of $O(\log mn^2)$.

For this thesis, however, m is always 12. Since this is a small number, not varying and to simplify all following time complexities, m will be seen as a constant and can therefore be disregarded in the time complexity of FindSmallestPackage. Therefore, the time complexity will be noted as $O(n^2)$.

4.10 Timings

Timings are an important performance measure for heuristics. It's how we measure computational effort and it has a great effect on the real-life applicability of heuristics.

The timings are recorded in line with the theory presented in section 3.9.4. For most heuristics only the total time it takes to run the heuristic is recorded. This is because most heuristics does not offer incremental improvements to the solutions. The total time is the time it takes to run the heuristic, including the time it takes to run the SBPP algorithm.

Some heuristics also record the time for iterative improvements. This means that they start to record the time when the heuristic is called and then store the time for each iteration when the solution is improved. Finally, they also record the time at the end of the heuristic to get the total run time.

4.11 Goal function

We use a simple linear goal function to evaluate different solutions. The goal function, g is defined in equation 4.5. It takes the shipping cost, SC, the material emissions, ME and the shipping emissions, SE and sums them together, with each term weighted by α_{SC}, α_{ME} and α_{SE} respectively.

$$g(SC, ME, SE) = \alpha_{SC}SC + \alpha_{ME}ME + \alpha_{SE}SE \tag{4.5}$$

As both the material and shipping emissions measure the same thing, carbon dioxide equivalents, α_{ME}, and α_{SE} should be the same value. However, we have set them as different to support increasing or decreasing the value of one type of emissions if one thinks that the default values used are too high or too low for a specific use case.

4.11.1 Balanced goal function

There are multiple ways the factors α_{SC}, α_{ME} and α_{SE} can be chosen. These can differ based on the use case and the importance of the different factors to the user. For this thesis, we create a balanced goal function where we try to make the emissions and costs balanced equally. To balance these weights we utilize the optimal solutions found using an *exhaustive search*, described in section 4.13. Normalizing the weights requires taking the average emissions when only optimizing for emissions and the average shipping cost when only optimizing for shipping costs.

More precisely, the *exhaustive search* algorithm ran for 10 000 randomly selected orders with between one and ten products. The algorithm ran twice for the same set of orders, once only optimizing for shipping costs and once only optimizing for lower emissions. Of these 10 000 orders, only 9970 were successfully packed, the rest had products that were too large for the available packages.

When optimizing for lower emissions, the average transport emission is 32.55 g CO_2-eq and the average material emissions is 100.08 g CO_2-eq, resulting in total average emissions of 132.64 g CO_2-eq. When optimizing for shipping costs, we assume that the cheapest possible combination of shipping methods is used for each order. This results in an average shipping cost of 115.41 SEK for the 9970 orders. Using the cheapest shipping methods means the cost is less likely to include penalties for large or heavy packages.

By setting $\alpha_{SC} = \frac{1}{115.41}$ and $\alpha_{ME} = \alpha_{SE} = \frac{1}{2 \cdot 132.64}$, the costs and emissions are balanced equally. This balanced goal function is defined in equation 4.6.

$$g(SC, ME, SE) = \frac{1}{115.41}SC + \frac{1}{2 \cdot 132.64}ME + \frac{1}{2 \cdot 132.64}SE \tag{4.6}$$

4.12 Heuristics

In this section, all heuristics will be explained and presented. This includes a description of how they work, their pseudocode, and a time complexity analysis of each heuristic.

All the heuristics follow a similar approach, utilizing the same input and presenting the suggested solution in the same format. The input is comprised of an ordered set of n products, each with 3 fixed dimensions, whereas the output consists of a list of packages, each containing a list of products. Some of these heuristics are relatively uncomplicated and greedy, serving

as a reference point for comparing the more intricate heuristics. A summary of their time complexities can be found in appendix B.

In all pseudocode below, P is the set of all available packages. As no stock levels or other limitations are consistent, P is always the full set of packages listed in table 4.4. Furthermore, Q is the set of all products in the order to be packaged. Another common notation used throughout the time complexity analysis done below, m is the number of packages ($m = |P|$) and n is the number of products ($n = |Q|$).

The heuristics all return a set of tuples in the form (p, Q_i) where p is the package to be used and $Q_i \subset Q, Q \neq \emptyset$ is the list of products to be packaged in the package p.

It should also be noted that the pseudocode is not meant to be a complete implementation of the heuristics. Issues such as error handling, breaking if no solution is found, and other implementation details are not included. The pseudocode is only meant to provide a general idea of how the heuristics function.

4.12.1 No split

No splits fundamental principle involves arranging items in sequential order and then placing them into the smallest possible package. However, if the order can't be contained in any package, the heuristic fails. Due to its simplicity, it serves as a benchmark for evaluating the effectiveness of the other proposed heuristics. The pseudocode can be seen in algorithm 4.2.

Algorithm 4.2 *No split* pseudocode

Require: P is sorted in ascending order according to volume
1: **function** NoSplit(P, Q)
2: $p \leftarrow$ FindSmallestPackage(P, Q)
3: **return** $\{(p, Q)\}$
4: **end function**

Since it should not do any partitioning, it simply calls the helper function FindSmallestPackage to find the smallest package that can fit the products. This means that the time complexity of the algorithm is the same as FindSmallestPackage, namely $O(n^2)$.

4.12.2 Full split

Full split can be characterized as the opposite of the no-split heuristic. In contrast to the latter, this heuristic involves dividing an order into separate packages, with each package containing a single product. The products are placed in the smallest package that can accommodate them. This approach is very simple and therefore also serves as a baseline for evaluating the other heuristics. The pseudocode can be seen in algorithm 4.3.

Algorithm 4.3 *Full split* pseudocode

Require: P is sorted in ascending order according to volume
1: **function** FullSplit(P, Q)
2: $S \leftarrow \emptyset$
3:
4: **for** $q \in Q$ **do**
5: $p \leftarrow$ FindSmallestPackage(P, $\{q\}$)
6: $S \leftarrow S \cup \{(p, \{q\})\}$
7: **end for**
8:
9: **return** S
10: **end function**

The heuristic call FindSmallestPackage for each product in Q. However, as there is always only one product, the time complexity of FindSmallestPackage in this case will only be $O(1)$, resulting in a total time complexity of $O(n)$.

4.12.3 Split by volume

This heuristic attempts to recursively partition the order into packages based on the volume of the products. It starts by attempting to fit all the products in the smallest possible package. If there is no possible solution that places them in the same package the order needs to be partitioned.

The partition is done by first sorting the products by volume. Then creating two groups, group left and group right. Iterating over the sorted products we then place the product in the group with the smallest total volume.

After the products are partitioned into two groups we then attempt to fit each group in the smallest package they fit into. If there is no package large enough we repeat the process partitioning the group into two groups and so on until we have a solution. If one group is only consisting of one product and there is no package large enough to fit it the heuristic fails. The pseudocode can be seen in algorithm 4.4.

Algorithm 4.4 *Split by volume* pseudocode

```
Require: P is sorted in ascending order according to volume
1:  function SplitByVolume(P, Q)
2:      p ← FindSmallestPackage(P, Q)
3:      if p ≠ null then
4:          return {(p, Q)}                    // Base case: Products fit into package
5:      end if
6:
7:      Q1, Q2 ← Split(Q)
8:      S ← ∅
9:      S ← S ∪ SplitByVolume(P, Q1)
10:     S ← S ∪ SplitByVolume(P, Q2)
11:     return S
12: end function
```

As can be seen, the heuristic requires a helper function to partition the products into two groups, here called Split. Any algorithm for partitioning can be used, but the one used for this thesis is described in algorithm 4.5.

First, the algorithm needs to call FindSmallestPackage to see if the products fit into a single package, this has a time complexity of $O(n^2)$. If it doesn't find any packages, it first needs to partition the set of products which in this case has a time complexity of $O(n)$. Then, it recursively calls itself on the two groups of products. Since it has divided the problem set, it will have log-like complexity, namely $O((n^2 + n) \log n) = O(n^2 \log n)$.

4.12.4 Biggie smalls

The goal of this heuristic is to group as many of the largest products in one package. The heuristic algorithm starts by sorting the products by volume in decreasing order. Then, it selects the largest product as a base product and tries to pack it into the smallest available package. If the base product fits, the algorithm adds it to a list of included products and remembers the package. Then, the algorithm tries to pack each of the remaining products with the base product, and if a combination of products fits into a package, the algorithm adds these products to the included list and remembers the package.

If there are remaining products after packing all possible products with the base product, the algorithm selects the next largest remaining product as the new base product and repeats

Algorithm 4.5 *Split by volume*: Split pseudocode

```
1:  function Split(Q)
2:      Q_1 ← ∅, v_1 ← 0
3:      Q_2 ← ∅, v_2 ← 0
4:
5:      for q ∈ Q do
6:          if v_1 ≤ v_2 then
7:              Q_1 ← Q_1 ∪ {q}
8:              v_1 ← v_1 + V(q)
9:          else
10:             Q_2 ← Q_2 ∪ {q}
11:             v_2 ← v_2 + V(q)
12:         end if
13:     end for
14:
15:     return Q_1, Q_2
16: end function
```

the above process until all products have been packed. The pseudocode for *biggie smalls* can be found in algorithm 4.6.

Algorithm 4.6 *Biggie smalls* pseudocode

```
Require: P is sorted in ascending order according to volume
1:  function BiggieSmalls(P, Q)
2:      Q ← Sort(Q)                          // Sort products by volume in decreasing order
3:
4:      S ← ∅
5:      while Q ≠ ∅ do
6:          Q' ← ∅
7:          p ← null
8:
9:          for q ∈ Q do
10:             p' ← FindSmallestPackage(P, Q' ∪ {q})
11:             if p' ≠ null then
12:                 Q' ← Q' ∪ {q}
13:                 p ← p'
14:             end if
15:         end for
16:
17:         Q ← Q \ Q'
18:         S ← S ∪ {(p, Q')}
19:     end while
20:
21:     return S
22: end function
```

First of all, to premiere packaging large products first, the products are sorted by volume in decreasing order. This can be done with any sorting algorithm, such as quicksort that has a time complexity of $O(n \log n)$. Then, the worst case is that only one product can fit in each package, in which case the while loop would only remove one product from Q each iteration. In each iteration, the algorithm would still need to test each other product left to see if any fit, using the FindSmallestPackage algorithm. This means that the time complexity of the algorithm is $O(n \log n + n^4) = O(n^4)$.

4.12.5 Group while fits

The *group while fits* can be considered a greedy algorithm, as it aims to merge as many large products as possible into a single package. The heuristic algorithm begins by partitioning the order into the smallest possible packages that can accommodate each product. If a product can't fit into any package the heuristic fails. Subsequently, the packages are sorted in decreasing order based on their volume. The heuristic then attempts to group the products from the two largest packages into the largest of the two packages. This process is repeated iteratively until a point is reached where grouping more products in the largest package would require exceeding the package size limit. At this point, the algorithm considers the largest package to be full and proceeds to group the next two largest packages, repeating the process until all the packages have been combined or are considered full. The pseudocode for *group while fits* can be found in algorithm 4.7

Algorithm 4.7 *Group while fits* pseudocode

```
Require: P is sorted in ascending order according to volume
function GroupWhileFits(P, Q)
    Q ← Sort(Q)                                  // Sort products by volume in descending order

    S ← ∅
    while Q ≠ ∅ do
        q ← Q[0]                                 // The largest product left
        Q' ← {q}
        p ← FindSmallestPackage(P, Q')

        for q' ∈ Q \ {q} do
            fits ← SBPP(p, Q' ∪ {q'})
            if fits then
                Q' ← Q' ∪ {q'}
            end if
        end for

        S ← S ∪ {(p, Q')}
        Q ← Q \ Q'
    end while

    return S
end function
```

The bulk of *group while fits* happens in a loop iterating over the products in the order. This loop has the time complexity of $O(n)$. Inside the loop, there are two parts, first finding the smallest package for the products, and then iteratively grouping the two largest packages until all packages are full. The time complexity of finding the smallest package is $O(n^2)$, as it calls FindSmallestPackage which has a time complexity of $O(n^2)$. The time complexity of iteratively grouping the two largest packages is $O(n^3)$, as it loops over all products and calls SBPP which has a time complexity of $O(n^2)$. This gives a total time complexity of $O(n \cdot (n^2 + n^3)) = O(n^4)$.

4.12.6 Eliminate bad fill rate

The heuristic *eliminate bad fill rate* employs a local optimization strategy to generate the optimal fill rate. Initially, the heuristic divides all products from the order into individual packages that are as small as possible. Subsequently, the packages are sorted based on their fill rate, which is computed as the product volume divided by the package volume. The heuristic then

proceeds to add the product in the package with the lowest fill rate to the package with the highest fill rate iteratively. This method is very similar to the best-fit method described in section 2.2.2. This approach removes the package with the lowest fill rate. When no product can be added to the package with the highest fill rate, the heuristic shifts its focus to the package with the second-highest fill rate and continues until no more products can be added to any package with a higher fill rate than itself. The pseudocode for *eliminate bad fill rate* can be found in algorithm 4.8

Algorithm 4.8 *Eliminate bad fill rate* pseudocode

```
Require: P is sorted in ascending order according to volume
function EliminateBadFillRate(P, Q)
    R ← ∅                          // Map of product to its smallest package. Used to determine fill rate
    for q ∈ Q do
        p ← FindSmallestPackage(P, {q})
        R[q] ← p
    end for
    Q ← Sort(Q, R)                 // Sort products by fill rate in descending order

    S ← ∅
    while Q ≠ ∅ do
        q ← max(Q, R)              // The product which is currently has the best fill rate
        p ← R[q]
        Q' ← {q}

        for q' ∈ Q \ {q} do
            fits ← SBPP(p, Q' ∪ {q'})
            if fits then
                Q' ← Q' ∪ {q'}
            end if
        end for

        S ← S ∪ {(p, Q')}
        Q ← Q \ Q'
    end while

    return S
end function
```

The time complexity of the algorithm can be divided into two phases. The first phase divides the products into individual packages and sorts them by fill rate. Since we need to iterate over all the products to place them into separate packages, the time complexity of this phase is $O(n)$ multiplied by the time complexity of FindSmallestPackage. As previously mentioned FindSmallestPackage normally has a time complexity of $O(n^2)$, but since we only need to find the smallest package for $n = 1$ product, the time complexity here is $O(1)$. Resulting in the first phase having a time complexity of $O(n)$. The second phase iterates over all the products in the order, and for all those products further iterates over all the possible packages they can be added to. For all these iterations, it calls SBPP which has a time complexity of $O(n^2)$ resulting in the second phase having a time complexity of $O(n^4)$. As the second phase dominates the first, the total time complexity of the algorithm is $O(n^4)$.

4.12.7 Random brute

Random brute is a brute-force approach that generates all possible order partitions and evaluates them one by one. Given a partition, the evaluation is carried out by first finding the

smallest package for each partition subset using the FindSmallestPackage function described in section 4.9.1. When all the smallest packages are found for the partition, the total shipping cost, material emission, and shipping emissions are calculated as presented in sections 4.6 to 4.8. These three values are then used in the goal function presented in section 4.11 to get a single value, representing how good the partition is. If the partition is better than the best partition found so far, it is saved. The last saved partition is then returned as the result of the algorithm since it gives the best goal function value.

The pseudocode for the *random brute* algorithm can be found in algorithm 4.9. The GenerateNextPartition function is a generator used to create new partitions of the order. Each partition R consists of one or several subsets, where one subset represents one group of products to be packed together. This function can be designed in multiple ways. The simplest is to generate all partitions using the example given in section 2.4.1 and then return them one by one.

Algorithm 4.9 *Random brute* pseudocode

```
Require: P is sorted in ascending order according to volume
 1: function RandomBrute(P, Q)
 2:     S ← ∅
 3:
 4:     t ← Now()                                          // Current time
 5:     while (Now() − t) ≤ 1000 ms do
 6:         R ← GenerateNextPartition()
 7:         S′ ← ∅
 8:         for R′ ∈ R do
 9:             p ← FindSmallestPackage(P, R′)
10:             S′ ← S′ ∪ {(p, R′)}
11:         end for
12:
13:         if GoalFunction(S′) < GoalFunction(S) then
14:             S ← S′
15:         end if
16:     end while
17:
18:     return S
19: end function
```

The time complexity of *random brute* is $O(B_n n^3)$. For each subset of a partition, of which there are n in the worst case, the FindSmallestPackage function is called which has a time complexity of $O(n^2)$. This results in a time complexity for each partition of $O(n^3)$. As presented in section 2.4.1 there are B_n partitions in the worst case, where B_n is the Bell number. This results in a total time complexity of $O(B_n n^3)$, assuming that the time complexities of GenerateNextPartition and GoalFunction are lower than $O(n^3)$.

One limitation of this algorithm is that for orders with many products, the number of possible partitions gets very large. For example, already at $n = 10$ the number of possible partitions is $B_{10} = 115\,975$. To handle this, one can change the GenerateNextPartition to generate random partitions instead of doing an *exhaustive search* starting at some n. This can improve the performance as generating partitions can be expensive, especially if it removes duplicate partitions due to multiple products of the same type.

To also limit the runtime, a simple time limit can be included, as it is in algorithm 4.9. This means that the heuristic will return the best partition found so far when the time limit is reached. In this case, the time limit is set to 1000 ms. Of course, this means that the optimal solution might not be found. However, by trying to include a wide array of cases early in partition generation, such as including partitions of all different numbers of subsets k, $1 \leq k \leq n$, the chance of finding a good solution before the time limit is reached is increased.

4.13 Exhaustive search

Comparing the heuristics with optimal solutions requires a method of calculating the optimum. For the problem studied in this thesis, we use an *exhaustive search* algorithm where the search space is the set of all possible order partitions for an order, as described in section 2.4.1. Since we use a non-exact algorithm for SBPP, the optimal solution to the full BPP is not guaranteed to be found using an *exhaustive search* approach. However, as each partition is evaluated, the optimal partition given by the SBPP algorithm is guaranteed to be found. Also, as the heuristics use the same SBPP algorithm, the results of the *exhaustive search* are comparable with the heuristics.

The algorithm used to find the optimal solution is very similar to that of the *random brute* heuristic described in section 4.12.7. The only difference is that the performance improvements of randomly generating partitions and stopping execution after a specified time limit. Instead, all partitions are generated and evaluated. The pseudocode for the *exhaustive search* algorithm can be found in algorithm 4.10.

Algorithm 4.10 *Exhaustive search* pseudocode

Require: P is sorted in ascending order according to volume

```
1:  function ExhaustiveSearch(P, Q)
2:      S ← ∅
3:
4:      U ← GenerateAllPartitions()
5:      for R ∈ U do
6:          S' ← ∅
7:          for R' ∈ R do
8:              p ← FindSmallestPackage(P, R')
9:              S' ← S' ∪ {(p, R')}
10:         end for
11:
12:         if GoalFunction(S') < GoalFunction(S) then
13:             S ← S'
14:         end if
15:     end for
16:
17:     return S
18: end function
```

Evaluating all partitions will of course increase the execution time, especially for large orders. To combat this, the orders evaluated can be limited to only include orders with a smaller set of products. This will mean that the results are a bit skewed since not all orders are included. However, this is necessary as the number of partitions grows very fast with the number of products in the order, too fast to be able to evaluate all of them.

Any orders with only one package can also be excluded, as all heuristics will give the same results. This will make the differences between the heuristics and the optimal case less representative of the general case. However, the differences will still be in the correct direction, only amplified.

As *exhaustive serach* is very similar to and only a modification of *random brute*, the time complexity is the same, namely $O(B_n n^3)$.

4.13.1 Different goal function

What the optimal solution is depends on the goal function used. To be able to compare the heuristics for multiple aspects, three different goal functions are used. These only consider the total shipping cost, the total emission, and a balanced metric of both the total shipping

cost and the total emission, as described and defined in section 4.11. Apart from the balanced goal function, the goal functions used are defined in equations 4.7 and 4.8. In these cases, the factors don't matter, as there are no multiple values that need to be balanced.

$$g_{cost}(SC, ME, SE) = SC \tag{4.7}$$

$$g_{emission}(SC, ME, SE) = ME + SE \tag{4.8}$$

5 Results

In this chapter, the results obtained are presented. Most of the results are based on simulations of the heuristics using all the two million orders in the dataset. A few results are based on a subset of the orders for various reasons, which are specified in the corresponding section.

The results are broken down into sections covering different aspects and factors, including performance, shipping cost, and environmental impact and lastly, the heuristics are compared to the optimal solution.

5.1 Partitioning

To give an overview of how the different heuristics partitions the orders, the distribution of the number of packages proposed for the orders is presented in table 5.1. The results show that the different heuristics have very different characteristics when it comes to how many packages they propose for each order.

Table 5.1: Distribution of packages per order

Heuristic	**1** %	**2** %	**3** %	**4** %	**5** %	**≥ 6** %
Full split	20.572	16.889	13.756	11.179	8.696	28.908
No split	100.000	0.000	0.000	0.000	0.000	0.000
Split by volume	99.856	0.127	0.006	0.007	0.001	0.002
Biggie smalls	99.906	0.079	0.009	0.003	0.001	0.001
Group while fits	58.689	25.253	8.949	3.562	1.547	2.001
Eliminate bad fill rate	48.890	27.665	12.229	5.402	2.526	3.287
Random brute	96.247	3.717	0.020	0.013	0.003	<0.001

As can be seen clearly, the *full split* and *no split* heuristics are the most extreme. The *full split* heuristic will assign a package to each product, meaning that it follows the same distribution as the number of packages per order. *No split* will instead assign all orders to the same package,

and never in more than one package, even though the heuristic might fail when a split could work. The other heuristics will place themselves somewhere in between these two extremes.

Split by volume and *biggie smalls* are the most conservative of the smarter heuristics and very rarely split orders, resulting in almost all of the orders being assigned to a single package. This leaves only a very small set of orders that are assigned to more than one package. The heuristics give similar results for these orders as well, with only quite small differences in the number of packages used.

On the other hand, *group while fits* and *eliminate bad fill rate* also perform similarly to each other. Both are significantly more willing to partition, with a total of 41.31 % and 51.11 % of the orders being assigned to more than a single package respectively.

Lastly, *random brute* behaves somewhere in between the two described groups. The vast majority of orders are still partitioned into a single package, but it still produces quite a few partitions with two or more packages.

5.2 Fill rate

In figure 5.1, the cumulative and average fill rates, as described in section 3.8.1, for the different heuristics are presented. As can be seen, most heuristics give similar fill rates, with *full split* being the exception. The cumulative fill rate is around 40 % for all other heuristics, with only minor differences between them. The average fill rate for them is a bit lower, around 35 % to 40 %, and varies more.

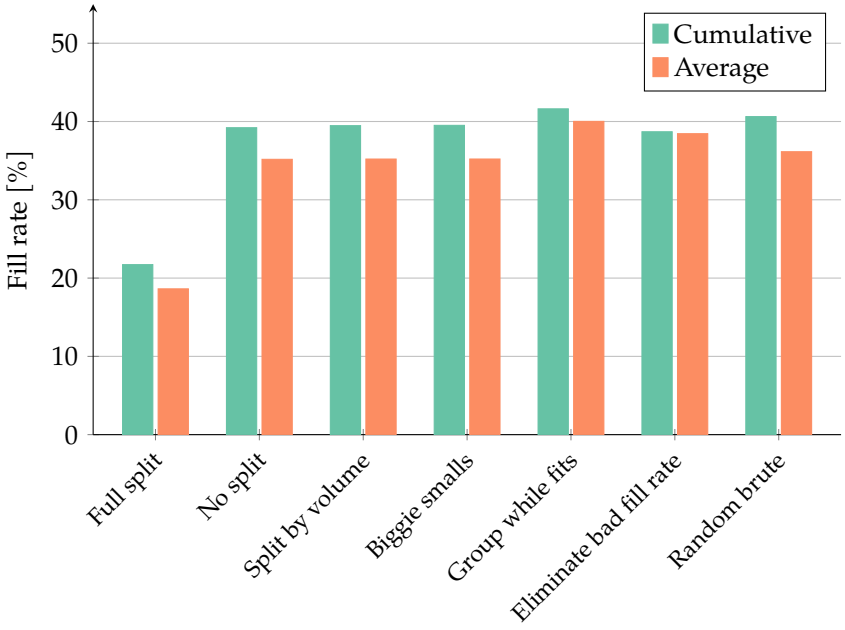


Figure 5.1: Heuristic fill rate

The discrepancy between the cumulative and average fill rate implies that large packages generally have a higher fill rate than smaller ones since they have a larger impact on the cumulative than the average fill rate, as described in section 3.8.1. Comparing using the partitioning above, the heuristics that split the orders into more packages, *group while fits* and *eliminate bad fill rate*, have a smaller span between their average and cumulative fill rate. This also fits the pattern of large and small packages and their impact on the cumulative and average fill rate, as these heuristics use more but smaller packages.

The best heuristic in regards to both cumulative and average fill rate is *group while fits*, with a cumulative fill rate of 41.64 % and an average fill rate of 40.02 %.

It should be noted that the average fill rate is the average for each package, not each order. So for orders that are partitioned into multiple packages, each package is counted separately toward the average fill rate. Technically, the same is true for the cumulative fill rate, but since the volumes are simply summed there, it does not make a difference.

5.3 Performance

To measure performance, the execution time of the heuristics is collected as a part of the simulation tool. It should be noted that the calls to the single bin packing algorithm execution time are included in this timing. In table 5.2, the 50th, 75th, 95th, 99th and 99.9th percentile as well as the mean of the execution times are presented. These percentiles are denoted as P_ϕ, where ϕ is the percentile.

The execution times are collected on the same 100 000 orders for each heuristic.

Table 5.2: Heuristic execution time

Heuristic	**Mean** µs	P_{50} µs	P_{75} µs	P_{95} µs	P_{99} µs	$P_{99.9}$ µs
Full split	**30**	19	31	**65**	**129**	**395**
No split	103	**13**	**30**	130	429	2485
Split by volume	62	14	32	138	469	3170
Biggie smalls	446	26	77	571	3106	37 840
Group while fits	56	18	38	134	368	1562
Eliminate bad fill rate	74	25	54	181	490	2163
Random brute	136 845	126	3437	1 000 046	1 000 169	1 013 735

As can be seen, the *random brute* heuristic clearly stands out amongst the heuristics, with the mean and several percentiles having execution times several orders of magnitude higher than the others. The median value is still low and comparable to the other heuristics, but it scales much worse than the others. The reason it stops scaling when reaching around 1 000 000 µs = 1 s is that it is the maximum execution time allowed, as defined in section 4.12.7. Out of the 100 000 orders, 13 169 or 13.17 % had an execution time equal to or above this limit. More results on the runtime of the *random brute* heuristic can be found below.

Compared to the others, *biggie smalls* is the slowest and scales the worst as well. However, it is at $P_{99.9}$ still considerably faster than the *random brute*, only taking 37.84 ms. The best scaling on the other hand is *full split*, which does not even take 1 ms at the 99.9th percentile. This is considerably better than *no split*, even though they are both very simple heuristics. The rest of the heuristics performed similarly through the percentiles. There are some differences between the values, but they are not very large and the scaling is similar.

5.3.1 Random brute performance

As said, a time limit is implemented for the *random brute* heuristic. To further investigate the performance of this heuristic, especially for larger orders, the heuristic ran on a smaller subset of orders but without a time limit. In figure 5.2 below, the runtime of the *random brute* heuristic is shown for the orders with different numbers of products. As described in section 2.4.1, the number of partitions to test can still differ largely between different orders with the same number of products, depending on if the products are of the same type or not.

For each number between 2 and 10, 1000 orders with that number of products were randomly selected. The execution time is then recorded for each of these orders when optimizing it using *random brute*. In figure 5.2, the percentage of the 1000 orders which are done after a

specific time is shown. Notice that the x-axis is a logarithmic scale to be able to facilitate the complete range of execution times.

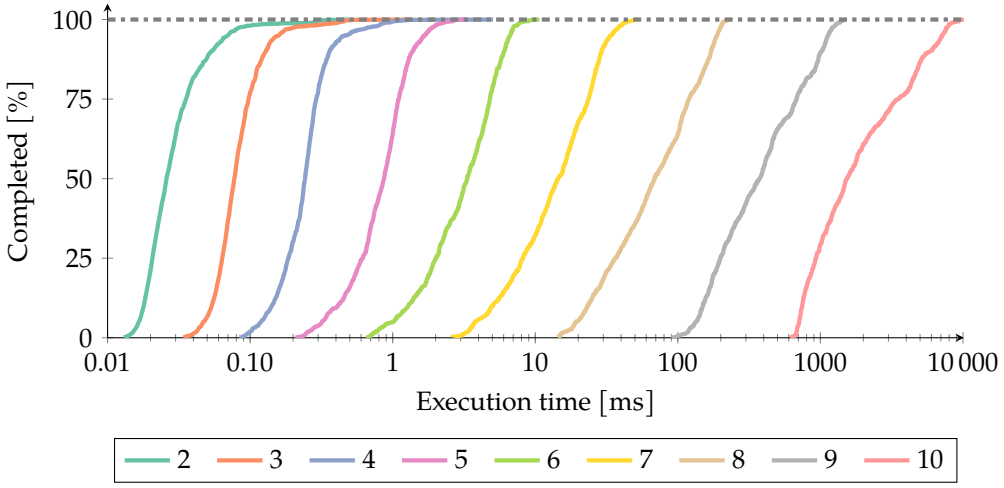


Figure 5.2: *Random brute* execution time for different numbers of products

As can be seen, the execution time is largely dependent on the number of products. However, at the same time, the execution time can also differ by around one order of magnitude for the same number of products. For smaller numbers of products, there also seem to be more corner cases with low and high execution times. They also have a very steep increase in between, meaning that many orders have similar execution times. For larger numbers of products, the outliers are fewer, and the line is straighter.

This can be used to decide on a time limit for the *random brute* heuristic. For example, given the time limit of 1000 ms used in *random brutes* means that most 9-product orders can be fully optimized, but only a smaller subset of the 10-product orders.

5.4 Robustness

The robustness of a heuristic method can be evaluated by measuring its failure rate, which indicates the number of times the method fails to produce a solution. There are two reasons that a heuristic can fail. The first is due to the heuristic taking too long to execute. For all the heuristics, a time limit of 5 s is used. After this time limit, the heuristic is killed and counted as a failure, since no result is produced. The second reason is that the heuristic is unable to find a solution. This is due to some of the products being too large to fit into any available package. The number of failures for each heuristic is presented in table 5.3.

Table 5.3: Heuristic failures

Heuristic	**Timeout**	**Product too large**
Full split	**0**	**7482**
No split	4	9394
Split by volume	5	**7482**
Biggie smalls	53	**7482**
Group while fits	1	**7482**
Eliminate bad fill rate	1	**7482**
Random brute	4	**7482**

As can be seen, there are 7482 orders which had one or more products that could not fit in any of the available packages. This is 0.37 % of the total number of orders. Apart from these orders where at least one of the products was too large, *no split* failed on an additional 1912 orders. This is due to *no split* not being able to split the products into more than one package when they don't fit in one.

As for timeouts, *biggie smalls* performs the worst, with 53 timeouts. The others only have between 0-5 timeouts, with *full split* having none. The dataset includes orders with a lot of products, which causes the SBPP algorithm to have too long execution times. *Full split* always splits the products into individual packages, which means that the SBPP algorithm only has to place one product in each package.

5.5 Emissions

As can be seen in figure 5.3, material emissions are responsible for the majority of the emissions. Excluding the baseline heuristics of *full split* and *no split*, the transport emissions are responsible for 21.65 % to 27.18 % of the total emissions. The same value for the material emissions is 72.82 % to 78.35 %.

For *full split*, both the transport emissions and material emissions are by far the largest among the heuristics. The others are more similar, with *group while fits*, and *eliminate bad fill rate* being a bit worse than the rest. These values correlate with the partitioning that the heuristics do, with partitions with more packages resulting in larger emissions.

Again, excluding the baseline heuristics, the difference in transport emissions between the heuristics is really small. The heuristic with the largest transport emissions, *eliminate bad fill rate*, is only 8.01 % greater than the one with the smallest, *random brute*. For the material emissions, the largest and smallest emissions come from the same heuristics. However, the difference is much larger, with *eliminate bad fill rate* having 42.97 % more emissions than *random brute*.

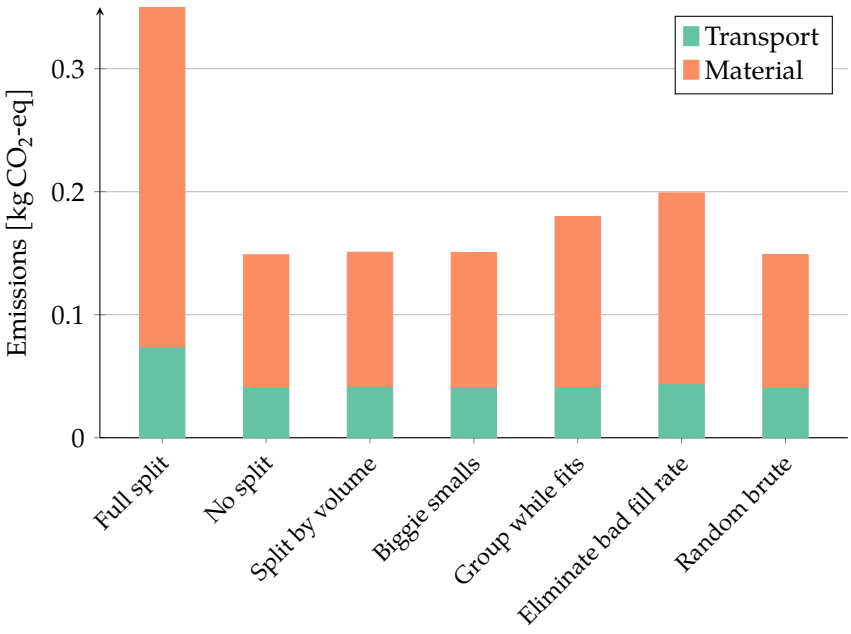


Figure 5.3: Heuristic emissions

5.6 Shipping cost

In figure 5.4 we present different shipping cost averages. As each order can consist of multiple packaging solutions and each packaging solution often has multiple available shipping options, it is difficult to compare the different heuristics. Therefore we aggregate the shipping results by calculating the cheapest, the most expensive, and the average available shipping cost for each package. Then these values are added together for each package in the order resulting in the lowest, highest, and average total shipping cost for each order. The figures presented in figure 5.4 are the average of these values for each heuristic. But as the values are not separated by transport method it's not possible to compare the different transport methods.

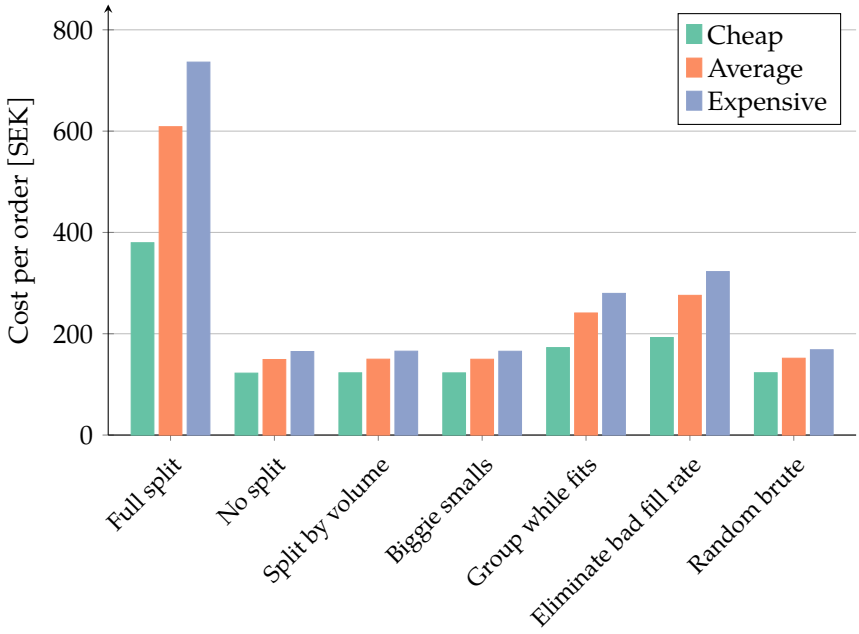


Figure 5.4: Heuristic cost

As depicted in figure 5.4, the average shipping cost per order is consistent across the heuristics *biggie smalls*, *no split*, *split by volume*, and *random brute*. This is because these heuristics present very similar packaging solutions.

5.7 Comparison to optimal solution

To compare the heuristics to the optimal solution, we use an *exhaustive search* approach as described in section 4.13. The heuristics are compared to three different optimal solutions which are developed using a separate goal function. These are only costs, only emissions, and a weighted, balanced, metric of these two. These results are presented below.

To compare the heuristics with the optimal solution, 10 000 orders are used. For each of these orders, the optimal solution is calculated using the *exhaustive search* method, and the heuristics are run on them. These are selected randomly with the constraint that there are between 2 and 10 products in each order. The distribution of products per order is presented in table 5.4.

As for all runs, there are certain orders which fail to be solved by the heuristics. All heuristics, except for *no split*, fail to solve 34 out of the 10 000 orders. This also includes the *exhaustive search* algorithm runs included below. The exception is *no split* which fails to solve 40 orders.

Table 5.4: Distribution of products per order for the orders used in the comparison to the optimal solution

Products in order	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Percentage of orders [%]	22.95	19.52	15.28	12.36	9.55	7.16	5.46	4.17	3.55

5.7.1 Shipping costs

As is found in section 5.6, there are a few heuristics that stand out significantly with high shipping costs. The rest do however have quite similar shipping costs, which all are low. When comparing all these heuristics to the optimal solution in figure 5.5, the good heuristics are close to the optimal solution when it comes to shipping cost. As an example, the optimal solution when weighted for only cost has an average of the lowest cost of 123.91 SEK while *biggie smalls* has an average of the lowest cost of 126.04 SEK. This is only an increase of 1.72 %.

Quite a few of the heuristics also have lower average costs when comparing the average and highest cost for each order. It seems that when optimizing for the lowest cost, as is done using the goal function, the average and highest cost per order increases. This can also be seen when looking at the balanced goal function, which includes both the lowest cost and emissions, and performs the best of the goal functions for the highest, and average costs.

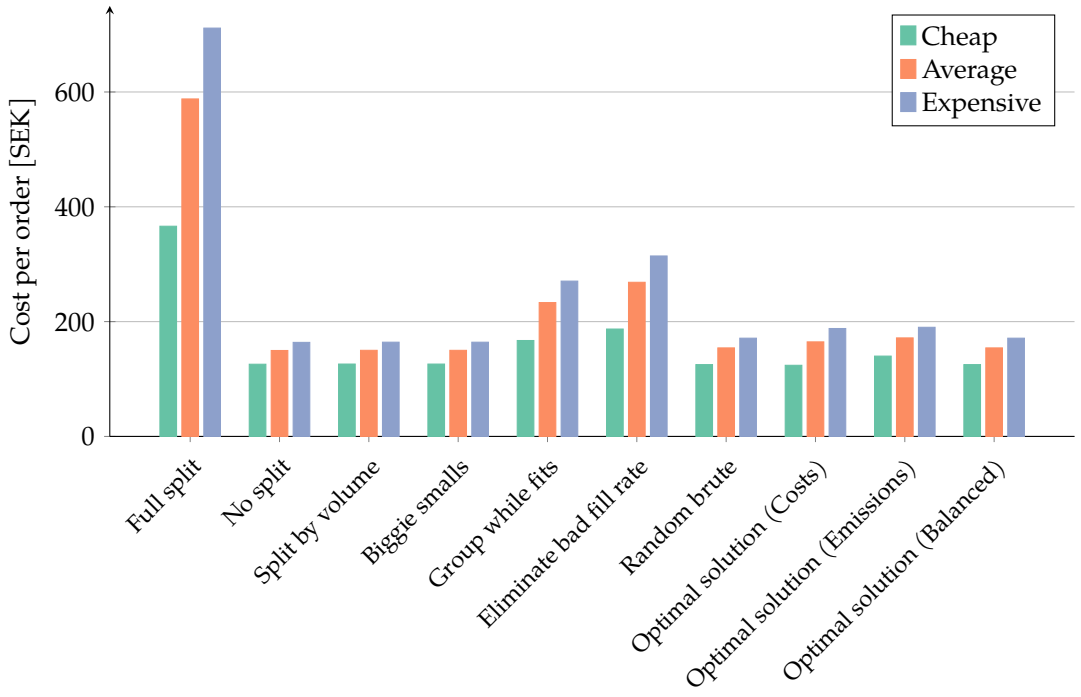


Figure 5.5: Heuristics costs compared to optimal shipping costs

5.7.2 Emissions

The story for emissions is quite similar to the one for shipping costs. There are a few heuristics that perform worse than most, namely *full split*, *group while fits*, and *eliminate bad fill rate*. The rest, however, perform very similarly to each other and the optimal solution. The difference is only a few percent, with *biggie smalls* only having 2.94 % worse emissions than the optimal solution as an example.

When comparing the different goal functions, the one optimizing for only costs performs the worst. It is 7.77 % worse, while the balanced only has 2.99 % higher total emissions.

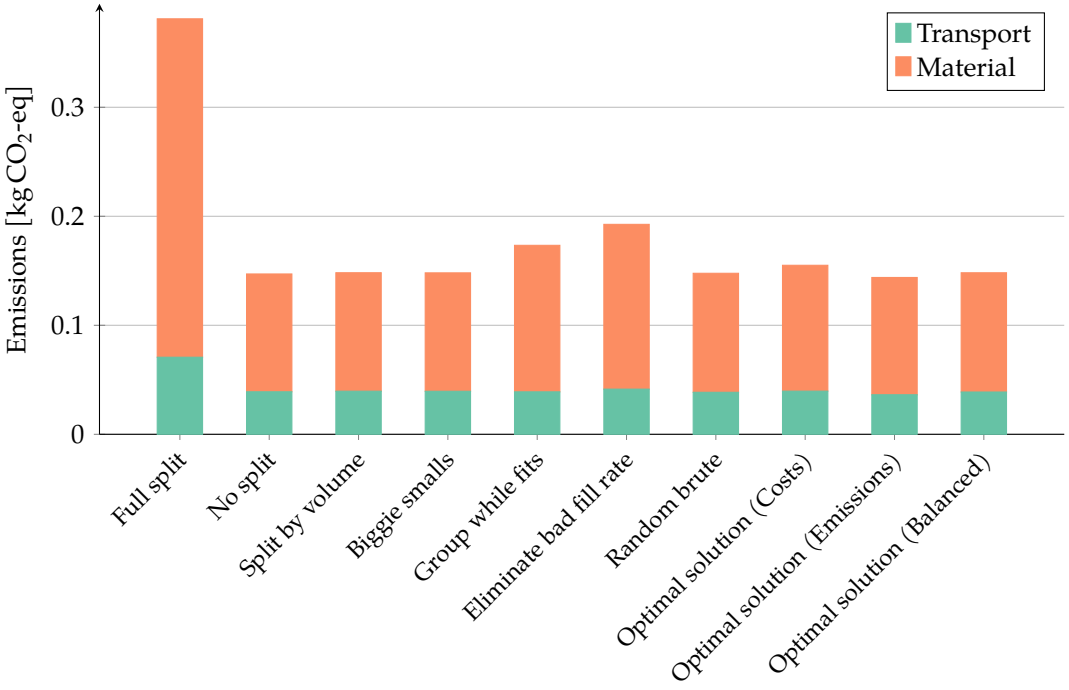


Figure 5.6: Heuristic emissions compared to optimal emissions

6 Discussion

The discussion chapter is where the core of the work is discussed and criticized. First of all, the results are discussed from several perspectives in section 6.1. Next, in section 6.2, the method used is discussed, including possible improvements and limitations. Finally, in section 6.3, the work and its implications are connected in a wider context.

6.1 Results

First, the heuristics are discussed from one factor at a time to then be discussed from a wider perspective. The discussion also brings in any earlier presented theory where applicable.

6.1.1 Partitioning

One interesting point from the partitioning data is the patterns that the different heuristics produce. As is presented in section 5.1, some heuristics group the products similarly to each other, even though they are quite different from each other.

As the *random brute* will by design try to achieve the optimal results, given the goal function, it is interesting to see that it decides to very rarely partition the orders into more than one package. Compared to the heuristics which splits the most, *group while fits* and *biggie smalls*, the *random brute* is much more likely to use two packages, but not more than that.

6.1.2 Fill rate

As is found in section 5.2, apart from the *full split* heuristic, most heuristics have quite similar fill rates. *Group while fits* performs the best, both for the cumulative and average fill rate, with a cumulative fill rate of 41.62 %. As described in section 3.8.1, a fill rate of 41.62 % represents a usage of $\sqrt[3]{41.62\,\%} = 74.66\,\%$ of the packaging in each dimension.

The cumulative fill rate is consistently higher than the average fill rate. As average fill gives smaller packages a larger weight compared to the cumulative, a lower average fill rate implies that smaller packages are the ones with the lowest fill rate. This could be described by the smaller relative steps in volume between the larger packages over the smaller packages. If a group of products does not fit in a certain package, but the next larger package is large enough,

it's preferable in terms of fill rate that the relative difference between the packages is as small as possible.

What is clear is that none of the heuristics achieve close to the 60 % as proposed by the European Union [10]. However, this is most likely not up to the heuristics nor the SBPP algorithm. To achieve a fill rate of 60 %, it's most likely that the packaging would need to change. If one would allow the packages to be height adjustable the fill rate would increase as the height would be able to be adjusted to fit the products better. Nevertheless, it's hard to say, but unlikely, that this would be enough to reach the 60 % goal.

6.1.3 Performance

The performance of the heuristics shows some interesting results. First of all, most heuristics have low execution times, even at the 99.9th percentile. Most of the simpler heuristics have an execution of at most a few milliseconds, which would most likely be better than needed for most business cases. With most mean and median execution times being under 100 µs for the simpler heuristics, it is clear that there are a large amount of simpler problems that can be solved very quickly, faster than is most likely needed.

It is the performance of *random brute* that results in more interesting considerations when it comes to use cases. In some cases, a mean execution time of 136 845 µs and a median of 126 µs might be too slow, but in other cases, it might be acceptable. *Random brute* does have the possibility to tune it to fit the use cases better. Firstly, the time limit can be increased or decreased to fit the use case. In this thesis, a limit of 1 s is used, but it could very much be changed. The limit for when to go from generating all partitions and iterating over them to creating random partitions could also be changed. Generating all partitions takes time as there are a lot of them, even for quite a small number of products. To then filter them to only include unique ones given the products in the order takes even more time. However, changing these could result in worse proposed packing, but also faster execution times.

Random brute differs from the other heuristics in another important way, namely that it depends on calculating costs and emissions to decide which partition to use. The others only depend on the products in the order and packages available. The calculations in this thesis are simple and fairly fast, but that might not be the case. If the calculations required API or database calls, or if they simply had higher time complexities, *random brute* might not be feasible to use. One could instead use simpler calculations to estimate the costs and emissions and use those in the goal function, but that might result in worse results.

It should be noted that the performance of the heuristics is heavily dependent on the algorithm used for SBPP. All the heuristics require running the SBPP algorithm at least once for each problem. The algorithm used in this thesis has a time complexity of $O(n^2)$. Other algorithms might have different time complexity and performance, and can also give different packing results. As the SBPP algorithm is so integral it can affect the validity of the results.

There are also multiple potential improvements if higher performance is needed. This applies both to lower the execution time of the simpler heuristics and to increase the number of tested partitions in the *random brute*. First of all, the code in this thesis is not very optimized, and multiple optimization techniques could be utilized to improve the execution time. Furthermore, the hardware used for the performance tests, as described in section 4.2.3 is limited, and more powerful hardware could easily be used to improve the performance. The heuristics are also implemented in Go, which is not the fastest language. Instead, a language like C or Rust could be used to improve the performance [62]. The same goes for the SBPP algorithm, which could benefit from the same techniques as the heuristics themselves to improve the performance of the heuristics.

6.1.4 Robustness

Robustness for these heuristics is measured by using the failure rate. As the heuristics are deterministic, they will always give the same result for the same input. However, the failure rate can vary for cases where the heuristic fails due to time out. This is primarily the case for *random brute*, as the partitions are generated randomly. But it's also true for the other heuristics as part of the performance will be dependent on the computing environment. The variation in failure rate has not been present in our simulations but could be present in a real-world scenario. As mentioned in section 3.9.4 it's important to try to reduce the variation in the results. How we control variation in the timings is described in section 4.2.3.

From the results in table 5.3, it's clear that the failure rate is very low for all heuristics. The failure rate also includes problem instances that are not solvable such as when there is a product that is too large to fit in any package. *No split* fails the most as it's more likely to be impossible to fit all products in a single package. The *full split* will fail the least, as it is both fast and only fails when the order contains a product that is too large to fit in any package. Both these heuristics help provide reasonable upper and lower bounds for the other heuristics. By comparing the results in table 5.3 to the results in table 5.2, it's clear that execution time is the driving factor for the failure rate.

6.1.5 Emissions

The emissions show similar patterns to the other metrics. Most of the heuristics exhibit similar emission results. However, there are a few outliers. For emissions, it's again *full split* which is the worst, while *group while fits* and *eliminate bad fill rate*, are better, but not close to as good as the others.

From the results, it's also evident that material emissions are the largest contributor to the total emissions. On average across the heuristics, not including *no split* and *full split*, the material emissions are 75.60 % of the total. Similar values are found from the optimal solution using the balanced goal function. This means that low material usage is the most important factor when it comes to reducing emissions. This is aligned with previous findings, such as by Pålsson et al. [9] which show that material usage is the largest contributor to the total energy usage for e-commerce orders.

As is described in section 4.7, the material emission is proportional to the surface area of the packaging. As the size of the package increases, its volume increases faster than its surface area, which would mean that fewer large packages are generally better than more, smaller packages. This is found when comparing the number of packages per order to the material emissions for the heuristics. The heuristics which tend to use more packages also have larger material emissions. The same goes for the weight per area of the cardboard, or grammage as it's called. By decreasing the amount of cardboard by using lighter cardboard, the material emissions are also decreased. This does however come at the expense of potentially less protective packaging. This balance is also discussed by Pålsson et al. [8], who brings up that the packaging needs to be selected with the products in mind.

For the transport emissions, the difference is much smaller. Excluding *full split*, the difference between the heuristics with the lowest and highest transport emissions is only 3.20 g CO_2-eq per order. This also boils down to the way the transport emissions are calculated. As presented in section 4.8, the transport emission is calculated based on the largest of the physical weight and the dimensional weight of the packages. If the physical weight were always the largest of the two, shipping emissions would be the same, excluding the difference in package weight, which is relatively small compared to the content. This is due to the content of the orders always having the same weight. What instead drives the transport emissions for the *full split* is that since it has such a low fill rate, the transport emissions do instead depend on the dimensional weight. This drives the transport emissions up, as the dimensional weight

includes a theoretical usage of space that could otherwise be used for other packages in the trucks.

6.1.6 Shipping cost

In figure 5.4 the most interesting data point is the average lowest shipping cost for each heuristic as that allows us to compare the lowest achievable cost. The disparity between the different solutions is very small not including the *full split*. And the best way to reduce shipping costs is to reduce the number of order partitions.

The other metrics other than the average lowest shipping cost is average, and average expensive shipping cost. These metrics give a better picture of the range of shipping costs and can help estimate shipping costs when using an arbitrary shipping method. However, there is still a strong correlation between the different shipping cost metrics. One interesting thing to note is that the heuristic optimizing for lower shipping cost slightly breaks this correlation and has slightly higher average and expensive values. It's also interesting to see that the trade-off between optimizing for emission and shipping cost is very small as there is a strong correlation between the two. We did not compare or analyze the specific shipping methods as that would not assist in evaluating the heuristics and is therefore considered out of scope.

6.1.7 Comparison to optimal solution

As stated in section 3.9.4 comparing the quality of solutions to an optimal result is important and helps improve the reliability, and validity of the results. The optimized solutions have yielded intriguing outcomes, highlighting the efficacy of simplistic heuristics in the order partitioning process, which demonstrate only slight differences compared to the optimal solution. Except for the *full split*, *group while fits* and *eliminate bad fill rate* heuristics, all other approaches show a close approximation, within a 10 % range, of the optimal solution's performance in terms of emissions and costs.

It is important to acknowledge that the optimal solution is assessed solely based on a limited dataset of 10 000 orders. Furthermore, this evaluation is limited to certain orders, products, and packages exclusively obtained from a single customer. The aforementioned constraints impose challenges in inferring any broad inferences from the results. Nevertheless, it is noteworthy that the outcomes suggest that simplistic heuristics may potentially yield satisfactory results in broader, real-world situations.

As previously mentioned, the optimized solution utilizing the balanced goal function exhibited decreased mean costs in the average and high-cost scenarios, compared to the cost goal function. The cost goal function used the lowest available transportation cost, thus neglecting the average and high-cost scenarios when optimizing. In a similar vein, the emissions goal function revealed comparable findings in terms of lower discrepancies between the lowest and highest costs. This pattern is also discernible across various heuristics, wherein the minimum, average, and maximum shipping costs display decreased intervals.

The balanced goal function appears to offer an equitable compromise between costs and emissions. Regarding emissions, the balanced solution bears a greater resemblance to the emissions goal function than the cost goal function. Conversely, for costs, the balanced solution aligns more with the cost goal function than the emissions goal function. While further experimentation and fine-tuning of the goal function may lead to an improved equilibrium between costs and emissions, the current rudimentary approach exhibits favorable outcomes.

6.1.8 Comparison between heuristics

Among the different measured metrics, some clear patterns are emerging. Firstly, *full split*, *group while fits*, and *eliminate bad fill* rate show similar results for the partitioning, emissions, and shipping cost metrics. They all partition orders into more packages, and at the same time

have worse emissions and shipping costs. This shows that the correct approach is to try to minimize the number of packages, as this will also minimize the emissions and shipping costs. The other heuristics all have similar results, which seem to stem from similar partitioning characteristics.

Disregarding *no split* as it will often not generate a solution, the other heuristics which had lower partitioning willingness also performed very close to the optimal solution. This is encouraging for the possibility to create a heuristic that is performant and provides near-optimal results. Many factors differ between different retailers and use cases, but for the data in this thesis, it is possible to achieve near-optimal results with a simple heuristic.

6.2 Method

Below, the method used in this thesis is discussed and criticized. Apart from discussing each part of the method, a criticism of the sources used is also included.

6.2.1 Method theory

Almost all of the method theory in section 3.9 stems from one guideline developed by Montgomery [56] and adapted by Barr et al. [57]. The original outline in the book by Montgomery [56] has been cited over 50 000 times, a testament to its credibility. Furthermore, the guideline by Barr et al. [57] is adapted to increase the applicability of the method theory to the field of computer science. However, even though the basis can be thought of as very credible, it can be considered a limitation that the method theory uses a limited number of sources.

6.2.2 Dataset

The data utilized in this thesis comprises real-world orders from a Swedish e-commerce company, which were provided by Skrym. To ensure that the research can be replicated, all data is anonymized and published together with the thesis. Table 4.1 illustrates that the dataset consists of a large number of orders, with product quantities ranging between 1 and 1000. However, it is more common for orders to contain a smaller number of products. As previously discussed in section 2.4.1, the number of partitions is greatly influenced by the number of products in each order. Therefore, the results are biased towards orders with a smaller number of products, which further indicates that the difference between the heuristic results is lower than it would have been if the dataset were more balanced. As briefly mentioned in section 3.9.2, it is vital to comprehend how these factors impact the behavior of the heuristics. Section 3.9.3 addresses the possibility of generating new orders with a more balanced distribution of product quantity. Although such an approach may offer a more comprehensive understanding of how heuristics operate in a wider range of scenarios, it may come at the expense of comprehending how the heuristics function in the real world. As the primary aim of this thesis is to understand the heuristics behavior in real-world scenarios, we chose not to generate any new orders and instead only use the real-world dataset. Section 4.1.2 mentions that the product volume is estimated by taking the cuboid closure of the product dimensions, as this is just an estimate it may lead to fill-rate results not being accurate. It's not just the number of products in each order that affects the number of partitions, but also product size and available package sizes. The number of partitions is affected as having larger products and or smaller packages means that not as many products can fit in each package and therefore forces the number of partitions to increase. Having products that are larger than the largest available package size means that the product can't be shipped at all and affects the number of failed orders. This inflates the number of failed orders and therefore also lowers the percentage difference between the robustness of the heuristics. The packages in the original dataset are height adjustable, meaning that they allow for a more compact fit than the estimation we used. For each height-adjustable package, we instead chose to use the maximum

height, minimum height, and mean height and declare them as three separate fixed-height packages. Using height-adjustable packaging would only increase the granularity of which we divide one height-adjustable package into separate fixed-height packages. Using an algorithm that can handle height-adjustable packages as the ones described in section 2.3 would not cohere with our goals to keep the results realistic. This is because the algorithms have infinite granularity when adjusting the height dimension. While it's not feasible to adjust a real-life package's height with infinite precision. Meaning that better solutions can be possible other than the ones we found using a limited set of packages. However, the implementation of heuristics for partitioning in this context would be very similar to the one we used and would not affect our goal in a significant way.

6.2.3 Simulation tool

The simulation tool is separated into two sections, the optimization section and the analysis section as can be seen in section 4.2.

Having the analysis section completely separate from the optimization section makes it easier to re-run analysis on the same data without the need to run costly simulations over and over. However, being able to use the cost and emission approximations as input in the heuristics allow for more targeted stopping functions and can lead to better results. This approach is still possible but it can be argued that it breaks the described model as data would flow backward from the analysis section to the optimization section. This would not be the case if both sections shared helper functions instead. An example of when the analysis is used in the optimization section is the goal function used in the *random brute* heuristic and the *exhaustive searches*.

The structure could also be changed so that it accommodates even more granularity in the data. For example, saving the results of each heuristic in different phases of execution. However, incremental improvement is only possible for the heuristics *random brute* and the *exhaustive search*. It would also be beneficial if the testbed stored information about the simulation environment to provide more transparency by default. The system in place now does not store environmental data but it's noted down and presented together with the results.

The simulation tool is also responsible for selecting the test cases. Selection is performed using a seed value to partition the dataset in batches, meaning that we could ensure that the test cases were the same for all heuristics while still being able to test pseudo-random test-case orders and batch sizes. This approach ensured better replicability and also low variation which is in line with the method theory on choosing design in section 3.9.3 and variability in section 3.9.4. However, to better test for corner cases and robustness the blocking data could have been used as discussed in the same section. The importance of data presentation orders is also mentioned in section 3.9.2, but as all the tests are containerized the order has a limited effect on the results.

As described in section 4.2.3 the simulation tool is designed to be run in a cloud environment. By running on a publicly available cloud provider such as GCP, reproducibility is increased as anyone has access to the same hardware aligning with the theory in section 3.9.4.

Also, with the increased interest in cloud computing in logistics [63], using cloud computing for the simulation tool is a step towards real-world applicability.

Running multiple processes in the same environment affects performance results as the processes compete for the same resources. Only allowing one concurrent request means that the tests are running isolated from each other, which further reduces the variability between the performance results, aligning with the theory in section 3.9.4.

6.2.4 Defining goals

The goals of the thesis have been established to correspond with the methodology theory in section 3.9 and research questions in section 1.3. The goals are categorized into two groups:

impact measures and real-world applicability. Both of these would be classified as performance measures under section 3.9.2 as they both evaluate the quality of solutions. This approach also aligns with the theory on variability mentioned in section 3.9.4.

The impact measures are fill-rate, emissions, and shipping cost and aim to evaluate if the heuristics are highly impactful as is mentioned as a criterion in section 3.9.1. However, several other factors such as shipping time, shipping quality, and the number of partitions can influence the efficiency of the heuristics solutions. These factors also have an impact on end-customer satisfaction. Assessing the impact of these factors on end-customer satisfaction is challenging without proper user studies. Given their impact on end-customer satisfaction, these factors fall somewhere between impact measures and real-world applicability. Consequently, to avoid making the thesis more complex than necessary, they were not considered.

The scope of real-world applicability in this thesis is limited to the performance of the heuristics, specifically resource efficiency, and robustness. In section 3.9.2 we mention that Barr et al. [57] also recommend using variance as a performance measure, but all the heuristics are deterministic and therefore have no solution variance for specific problem instances. They do have variance in the timings, but the variance is low as the tests are containerized, see section 4.2.3. However, it is important to note that other factors may also affect the real-world applicability of the heuristics, such as simplicity and generalizability. While simplicity is not considered in detail for most of the heuristics, it could be discussed further for the *exhaustive search* and *random brute* heuristics, which utilize data from the impact calculations. Furthermore, utilizing emissions and cost calculations in the real world would require more accurate estimations and could also impact the computational effort. As there is a low disparity between the test cases, the generalizability is only reflected in the failure rate, meaning that it is the same as the robustness.

In the evaluation process, the method theory in section 3.9.4 serves as the main reference for assessing the heuristics. To gain a comprehensive understanding of their performance, the heuristics are not only compared to one another but also optimal solutions, enabling the establishment of an upper limit. Additionally, to establish a lower limit, the heuristics are compared to greedy heuristics. However, it is worth noting that there is potential for improving the evaluation process. For instance, incorporating additional measures to record the impact and applicability of the heuristics could provide a more thorough evaluation. Furthermore, continuously testing and refining the heuristics as more results become available can further improve their effectiveness. When reiterating the development of the heuristics, it is important to carefully consider the possibility of overfitting the heuristics to the test cases.

6.2.5 Placement of warehouse

Due to the complexity in calculating both the cost and emissions for the different warehouse locations we only had the resources to use one location. To reduce the impact of this we chose a location based on the demographic center of Sweden as we deemed this would give the most widely applicable results given that placing a warehouse close to where people live increases the chance of having short delivery routes. This assumption is not guaranteed to be true as the population distribution is not uniform and the distribution of customers does not necessarily correlate with the population distribution. The population is also calculated using data from 2006 further increasing the risk of the population distribution being inaccurate. Using only one warehouse placement affects the results and depending on the location of the real-life warehouse and customers the results could be very different. The description of where the warehouse is placed is described in depth in section 4.5 and the theory of demographic center is described in section 3.5.1.

6.2.6 Shipping cost calculation

The cost calculations are only done using one logistics provider, PostNord. PostNord is used as it is by far the largest logistics provider in Sweden in terms of the volume of packages delivered, with a market share of 65 % to 70 % [64]. In reality, other providers and methods would be available as well, making the cost calculations more complex but also more realistic. It might also be the case that the end customer selected a specific shipping option, which has an impact on the cost. The possibility to only use a specific shipping option is not available in the simulation tool. The use cases differ between retailers, and the tool should be adopted to fit the specific use case.

As the cost is very dependent on the shipping locations (both destination and origin) they must be generalized to not affect variability. We chose to use one imaginary warehouse location as mentioned in section 4.5 and then generalize the destination based on the population division in Sweden. This means that the cost calculations are not representative of the real cost but rather a generalization of the cost. As the placement of the warehouse also is based on the population distribution the generalized cost can be skewed to the lower end (delivering orders to customers close to the warehouse). This reduces the variability of the study and increases the replicability. However, if the population does not reflect the distribution of the customers the cost calculations will accurately reflect the real world.

6.2.7 Material emission estimation

The material emission estimation requires first understanding the amount of material used in each package and then also knowing the carbon footprint of the material. Both of these factors can vary between cases so we use international standards to get both the average weight and emission factor. All this information is publicly available increasing the replicability further, generalizing lowers the variability but also affects the accuracy of the results.

We already use a quite low emissions factor per gram of cardboard for the calculations. We use the value 0.82 kg CO_2-eq/kg, which is taken from Brogaard et al. [38]. In their literature study, the range of values is quite large, meaning it is hard to get an accurate value to use. The material emission factor used is the emission for recycled cardboard and if the cardboard is not recycled the emission factor would instead be higher as discussed in section 3.3.1. This factor could also affect the application of the estimations in a real-life scenario as the cardboard used might not be recycled. With this in mind, retailers should use actual values received from their packaging suppliers to get a more accurate value.

This thesis does not consider any filling material that might be needed to protect the products. It also does not include any tape or similar materials used for sealing the package. To achieve more accurate results these factors should be included in the calculations. Retailers can most likely get ahold of this information from their suppliers. However, they still need to estimate the amount of filling and sealing material used for each package.

6.2.8 Shipping emission estimation

The shipping estimation calculations are based on a method developed by NTM. NTM is not a peer-reviewed source but instead, a non-profit organization using feedback and data from industry partners and stakeholders. As mentioned in section 3.4.1 the data provided by them is not based on research or statistical analysis. Their data is based on approximations from members that used volumetric weight for analyzing cargo with low density in Swedish conditions. The members and stakeholders might have different biases, agendas, and methods for collecting data. All of these factors affect the accuracy of the data and the validity of the results.

The driving factor that affects the emissions is the transport distance. The distance is based on the location of the warehouse and the destination of the package and the PostNord terminals. In extension, the distance between these transport legs also decides the vehicle used.

The PostNord terminal locations (see table 4.5) are provided by Skrym but we were not able to verify the information with PostNord as they did not respond to our inquiries. Therefore the information might be outdated and not reflect the current state of the PostNord terminals. Furthermore, there might be other terminals not included in the dataset. If the data is not correct it would affect the accuracy of the transportation emission calculations. The heuristics are however compared using the same data, meaning that the results are still comparable.

The leg distances are also generalizations as the locations are not available in the order data. Generalizing the locations makes the results of individual test cases less accurate but increases the replicability and reduces the variability. Further, it still provides an estimate that can be used to compare the heuristics to each other. The generalization is a combination of theory in sections 3.4.3 and 3.5 The actual distance is then calculated using the Haversine formula which is a good approximation of the distance between two points on a sphere but does not take into consideration the actual route taken.

The vehicle choices are based on recommendations from NTM and may be inaccurate. All the vehicles have LCU which has been weighted for dimensional weight, making the estimations much more suitable for our purposes compared to using purely weight-based utilization. We are required to use the dimensional weight to calculate LCU of the vehicles since the total weight of each order is unaffected by the heuristics. What type of vehicle is used is also very dependent on the location of the warehouse and the destination of the package. As both of these may vary greatly as discussed in section 6.2.5 and previously in this section the vehicle choice may not be representative of the real world but rather a generalization.

6.2.9 Single bin packing problem algorithm

As previously stated, the SBPP algorithm used in this thesis is the one presented by Dube and Kanavathy [20]. Its simplicity and availability as a package in Go made it a good choice for this thesis. There are multiple heuristics and algorithms available for solving SBPP and the choice of algorithm may affect the results.

However, the aim of this thesis is not to evaluate the SBPP part of BPP, but rather to evaluate methods for partitioning the orders. This means that as long as the SBPP algorithm can solve SBPP acceptably, it would work for the thesis. The exact impact of the choice of an SBPP algorithm is unknown, it might be that a different choice of algorithm would improve or worsen the results of certain heuristics in this thesis.

6.2.10 Goal function

To find the optimal solution when dealing with multiple goals, it is necessary to establish a goal function and consider the relative importance of each goal. In this thesis, we employed a normalization process followed by a weighting method to balance the goals. However, obtaining accurate normalizations can be challenging, as it requires determining the average value of each goal. Finding the exact average value for each goal is not feasible, further compounding the difficulty of normalizing the weights. To obtain these averages, we used a smaller batch of 10 000 orders. Using a smaller batch will introduce a bias in the results and affect the study's reliability and validity. However, it's not possible to get the exact average of a goal in real-life as the average value will change over time so it would not be fair for us to do it either. It would be possible to use a larger sample size to determine the average value of each goal, further, enhancing the generalizability of our findings and minimizing potential biases. But using a large sample size and then evaluating the heuristics on the same dataset would not be representative of real-life scenarios and would over-fit the goal function to the dataset. Additionally, it is crucial to evaluate the goals in the same context in which the heuristics will be used, and a rolling average could be a potential solution to address the challenge of fluctuating costs, emissions, and fill rates over time.

Once the goal function is defined, we evaluate the heuristics by using different weightings for each goal, including a balanced weight and a fully weighted approach. By doing so, we were able to assess the performance of the heuristics in various scenarios.

To follow the recommendations in section 3.9.4 we have specified all the tuning parameters used in the goal function. We have also analyzed the robustness, generalizability, and interaction between the parameters in the goal function. However, we have not provided an estimate of the time required to fine-tune the parameters meaning it can be time intensive to replicate in a real-world scenario.

6.2.11 Timing

The timing method of the heuristics is based on section 3.9.4 and it records complete runtimes and for some heuristics also times for incremental improvements. The exact method is detailed in section 4.10 further improving replicability. However, all the timings include the time to run the SBPP algorithm and it would be interesting to separately record the time for the SBPP algorithm and the time for the heuristic. This would allow for a detailed analysis of how the choice of the SBPP algorithm affects the total runtime of the heuristics.

6.2.12 Heuristics

In the process of developing heuristics for our thesis, we found that some heuristics such as *no split* and *full split* were relatively simple to implement and served as a useful baseline for our thesis. By focusing on simplicity, we were able to align with both the theoretical foundations of heuristic development outlined in section 3.9.1 and the goals we set out in section 4.3. Moreover, maintaining a consistent input and output format for the heuristics facilitated comparison and implementation in our testbed, while ensuring deterministic results enhanced the replicability of our study.

Despite *random brute* deviating from the principles of simplicity and consistency to some extent, it remained relatively straightforward to understand and implement. The results from *random brute* are not deterministic unless all the possible partitions are evaluated. This scenario renders it to produce equivalent results to the *exhaustive search* method. Notably, both *exhaustive search* and *random brute* are the only heuristics that progressively improved the solution iteratively.

The advantage of having an iterative heuristic is that it allows for the evaluation of the heuristics' performance over time, as depicted in figure 5.2. This approach enhances the reliability and validity of our study, enabling us to measure the efficacy of the heuristics and provide insights into their relative effectiveness.

While our approach to heuristic development and implementation had some advantages, it is important to acknowledge its limitations and potential drawbacks. For instance, focusing on simplicity and consistency may have resulted in overlooking other critical factors that could affect the heuristics' performance or applicability in real-world scenarios.

Moreover, the use of a testbed environment may not fully capture the complexity and variability of real-world situations, and there is a risk of overfitting the heuristics to the specific problem instance. This could limit the generalizability of our findings and may not necessarily translate to other problem domains or datasets. However, we did not iterate on the heuristics so there is a smaller risk of overfitting. Furthermore, performance metrics used to evaluate the heuristics' effectiveness may not fully capture the practical implications of the solutions generated by the heuristics in real-world scenarios. Therefore, while our method had its merits, it is important to interpret the results with caution and consider the potential limitations and challenges that may arise when applying the heuristics in practice.

6.3 The work in a wider context

The logistics sector has a large problem at hand. While e-commerce is growing [1, 2] and more packages are being delivered [64], the transport and retail sector needs to decrease their emissions significantly to meet regulation targets, such as those set by the European Union [65, 10]. One way for these sectors to reduce emissions is to optimize the packaging of goods, which is the main focus of this thesis.

At the same time, using heavy computations has its own environmental issues. For example, cloud computing uses a lot of energy [66, 67]. Therefore, efficient algorithms are also needed to reduce the environmental impact of the computations. Using the performance in conjunction with metrics for emissions and shipping costs in this thesis, one can find good heuristics that both perform well while being efficient at the same time.

Three of the most influential factors for driving e-commerce are convenience, simplicity, and price [68]. By improving the packaging of orders in e-commerce, especially the price can be further lowered as shipping is made cheaper. This means that implementing packaging solutions similar to the ones developed in this thesis can increase the usage of e-commerce, which in turn has a plethora of other societal effects. One such is the potential for further decreased emissions in some contexts [7, 3].

7 Conclusion

The purpose of this thesis is to investigate the impact of order partitioning heuristics as a part of 3D-BPP and to explore the real-world applicability of this approach. To achieve this purpose, the following research questions were formulated:

RQ1. How do different order partitioning heuristics affect shipping costs, material emissions, and transport emissions?

RQ2. How can a heuristic for order partitioning be implemented so that it functions in a real-time system, given time and data constraints applicable to e-commerce logistics?

Contrary to what could easily be assumed, fill rate is not the best indicator of lower shipping costs and emissions. Instead, the number of packages per order proposed by the heuristics correlates better. As is found, the heuristics that split orders into more packages per order give higher shipping costs and emissions. For example, *eliminate bad fill rate* and *group while fits*, which both had great average fill rates and partitioned more, had worse results in other metrics. The same goes for the extreme case, where the *full split* heuristic results in more than double the shipping costs compared to all other presented heuristics. Other heuristics that partition less, namely *no split*, *split by volume*, *random brute*, and *biggie smalls* instead give the lowest overall shipping costs and emissions and were quite similar to each other. These heuristics also showed to give results that were close to the optimal solution found using the *exhaustive search* algorithm.

The EU proposal described in section 3.1.1 suggests a required minimum of 60 % fill rate for all packages. This fill rate is not feasible to achieve with the dataset used in this thesis, as the heuristics analyzed in this thesis only achieve fill rates close to 40 %. Further, the results show that optimizing for a better fill rate could result in higher shipping costs and emissions.

So how does the heuristics perform in terms of their ability to be implemented in a real-time system? The real-time applicability is in this thesis evaluated based on three criteria: time complexity, data requirements, and robustness. All the heuristics provide solutions for 99.9 % of the orders in the dataset used in less than 1 second. It should be noted that for the *random brute* heuristic, the timeout is set to one second, making the performance results a bit arbitrary. However, it is also this timeout that enables *random brute* to fit into multiple contexts, since it can be tuned to either fit better solutions or faster performance.

To see how the heuristics perform as the number of products per order increases, time complexity analyses of the heuristics are included. *Full split*, *no split*, and *split by volume* all have very resonable time complexity of at most $O(n^2 \log n)$. *Biggie smalls*, *eliminate bad fill rate*, and *group while fits*, *exhuastive search* scale much worse. However, as the number of products per order is relatively small for most orders, a bad time complexity does not necessarily mean a slow execution time.

The data requirements are also very reasonable for most of the heuristics. All heuristics but *exhaustive search* and *random brute* only require the product and package dimensions. *Exhaustive search* and *random brute* also require data to calculate shipping costs and emissions, which could or could not be easily accessible. The emissions and costs can be hard to calculate and therefore increase the difficulty of implementing these heuristics in a hard real-time system. Instead, estimations could be used or the heuristics could be used in a non-real-time or soft real-time system where the time constraints are not as strict. Furthermore, all heuristics are very robust and generalizable, except for *no split* that fail for all orders that require partitioning.

To conclude, optimal solutions are possible to find within a reasonable amount of time for smaller orders, as is shown with *random brute*. To achieve good results with simpler heuristics, they should partition an order into more than one group as rarely as possible to achieve low emissions and shipping costs. One example of a heuristic that does this is *split by volume*, which also has the advantage of being very fast and robust.

7.1 Future work

There are multiple potential ways to continue the work in this thesis. Firstly, one limitation of this thesis is that while it uses a large set of orders for analysis, they are all from the same retailer. No two retailers have the same order patterns, product assortment, and package sizes, meaning that the heuristics will not necessarily perform the same. To further validate the results of this thesis, the heuristics could be tested on orders from multiple retailers, preferably from different sectors. It would also be interesting if these orders were from different countries, as this would allow for more generalizable results.

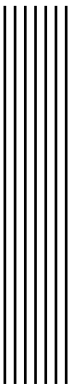
To also make it more verifiable, more retailer-specific contexts could be considered in future work. This includes more accurate data on emission factors, transport distances, shipping costs, and delivery alternatives. This applies especially to the delivery methods, as this thesis only considers options from PostNord. By including more delivery methods, the shipping cost results would be more generalizable. It might also be the case that end customers can decide on the used shipping method in the checkout step, which would mean that the cost calculations need to support fixed choices. This would in turn affect the results of the heuristics.

Secondly, one point that is discussed in section 3.9.3, but not utilized in this thesis is the use of blocking data. Blocking data would allow exploration of corner cases, further validating the results and enabling a deeper understanding of the heuristic's behaviors.

The results in this thesis are only compared to the other heuristic and the optimal solution. No efforts are made to compare the results to the current way of packing orders. It would be of interest to do so since it would give data on how much the emissions and shipping costs could be reduced by implementing the heuristics into their logistics workflow. We see this as a necessary step to ensure relevant stakeholders have the information required to make decisions regarding the packaging problems studied in this thesis.

As is the case for the customer utilized in this thesis, retailers, and warehouses do not only use fixed package sizes. Other types of packages can be used, such as packages with adjustable dimensions. By being able to vary one or more dimensions, the fill rate could be improved. Supporting variable dimensions in the simulation tool would change the problem from BPP to ODP and require a different algorithm for SBPP. However, instead of using a

different algorithm that supports variable dimensions, the current SBPP algorithm could be used if the adjustable packages were split into multiple discrete package sizes. Lastly, the only two metrics used in this thesis are emissions and shipping costs. In reality, e-commerce retailers have more metrics to consider to achieve customer satisfaction such as delivery times, customer closeness to their pick-up point, risk of damaged products, support for returns, and much more. While some of these metrics might be hard to measure, it would be of interest to include more of them in future work to get a more complete picture of the heuristics' and their impact.



Bibliography

[1] Eurostat. *E-commerce statistics for individuals*. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=E-commerce_statistics_for_individuals (visited on 14/11/2022).

[2] U.S. Department of Commerce The International Trade Administration. *eCommerce Sales & Size Forecast*. URL: https://www.trade.gov/ecommerce-sales-size-forecast (visited on 10/05/2023).

[3] Riccardo Mangiaracina, Gino Marchet, Sara Perotti and Angela Tumino. "A review of the environmental implications of B2C e-commerce: a logistics perspective." In: *International Journal of Physical Distribution & Logistics Management* 45 (July 2015), pp. 565–591. DOI: 10.1108/IJPDLM-06-2014-0133.

[4] Stanley Frederick W.T. Lim, Xin Jin and Jagjit Srai. "Consumer-driven e-commerce: A literature review, design framework, and research agenda on last-mile logistics models." In: *International Journal of Physical Distribution & Logistics Management* Forthcoming (Dec. 2017). DOI: 10.1108/IJPDLM-02-2017-0081.

[5] Eric H. Grosse, Christoph H. Glock and W. Patrick Neumann. "Human factors in order picking: a content analysis of the literature." In: *International Journal of Production Research* 55.5 (2017), pp. 1260–1276. DOI: 10.1080/00207543.2016.1186296.

[6] Nils Boysen, René de Koster and Felix Weidinger. "Warehousing in the e-commerce era: A survey." In: *European Journal of Operational Research* 277.2 (Sept. 2019), pp. 396–411. DOI: 10.1016/j.ejor.2018.08.023.

[7] Heleen Buldeo Rai, Sabrina Touami and Laetitia Dablanc. "Not All E-commerce Emits Equally: Systematic Quantitative Review of Online and Store Purchases' Carbon Footprint." In: *Environmental Science & Technology* 57.1 (Dec. 2022), pp. 708–718. DOI: 10.1021/acs.est.2c00299.

[8] Henrik Pålsson, Christian Finnsgård and Carl Wänström. "Selection of packaging systems in supply chains from a sustainability perspective–the case of Volvo." In: *Packaging Technology & Science* 26.5 (2013), pp. 289–310. DOI: 10.1002/pts.1979.

[9] Henrik Pålsson, Fredrik Pettersson and Lena Winslott Hiselius. "Energy consumption in e-commerce versus conventional trade channels - Insights into packaging, the last mile, unsold products and product returns." In: *Journal of Cleaner Production* 164 (Oct. 2017), pp. 765–778. DOI: 10.1016/j.jclepro.2017.06.242.

[10] European Commission, Directorate-General for Environment. *Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on packaging and packaging waste, amending Regulation (EU) 2019/1020 and Directive (EU) 2019/904, and repealing Directive 94/62/EC*. 2022. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52022PC0677 (visited on 05/05/2023).

[11] Silvano Martello, David Pisinger and Daniele Vigo. "The Three-Dimensional Bin Packing Problem." In: *Operations Research* 48.2 (Apr. 2000), pp. 256–267. DOI: 10.1287/opre.48.2.256.12386.

[12] Gerhard Wäscher, Heike Haußner and Holger Schumann. "An improved typology of cutting and packing problems." In: *European Journal of Operational Research* 183.3 (2007), pp. 1109–1130. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2005.12.047.

[13] Harald Dyckhoff. "A typology of cutting and packing problems." In: *European Journal of Operational Research* 44.2 (Jan. 1990), pp. 145–159. DOI: 10.1016/0377-2217(90)90350-k.

[14] Andreas Bortfeldt and Gerhard Wäscher. "Constraints in container loading – A state-of-the-art review." In: *European Journal of Operational Research* 229.1 (Aug. 2013), pp. 1–20. DOI: 10.1016/j.ejor.2012.12.006.

[15] Edgar den Boef, Jan Korst, Silvano Martello, David Pisinger and Daniele Vigo. "Erratum to "The Three-Dimensional Bin Packing Problem": Robot-Packable and Orthogonal Variants of Packing Problems." In: *Operations Research* 53.4 (Aug. 2005), pp. 735–736. DOI: 10.1287/opre.1050.0210.

[16] Silvano Martello, David Pisinger, Daniele Vigo, Edgar den Boef and Jan Korst. "Algorithm 864." In: *ACM Transactions on Mathematical Software* 33.1 (Mar. 2007), p. 7. DOI: 10.1145/1206040.1206047.

[17] Oluf Faroe, David Pisinger and Martin Zachariasen. "Guided Local Search for the Three-Dimensional Bin packing Problem." In: *INFORMS Journal on Computing* 15.3 (Aug. 2003), pp. 267–283. DOI: 10.1287/ijoc.15.3.267.16080.

[18] Guido Perboli, Teodor Gabriel Crainic and Roberto Tadei. "An efficient metaheuristic for multi-dimensional multi-container packing." In: *2011 IEEE International Conference on Automation Science and Engineering*. IEEE, Aug. 2011. DOI: 10.1109/case.2011.6042476.

[19] Teodor Gabriel Crainic, Guido Perboli and Roberto Tadei. "Extreme Point-Based Heuristics for Three-Dimensional Bin Packing." In: *INFORMS Journal on Computing* 20.3 (Aug. 2008), pp. 368–384. DOI: 10.1287/ijoc.1070.0250.

[20] Erick Dube and Leon R. Kanavathy. "Optimizing three-dimensional bin packing through simulation." In: *Sixth IASTED International Conference Modelling, Simulation, and Optimization*. 2006.

[21] Sandra Brüel Grönberg and Kajsa Hulthén. "Disembedding air from e-commerce parcels: A joint challenge for supply chain actors." In: *Industrial Marketing Management* 107 (Nov. 2022), pp. 396–406. DOI: 10.1016/j.indmarman.2022.10.012.

[22] Guntram Scheithauer. "A Three-dimensional Bin Packing Algorithm." In: *Elektronische Informationsverarbeitung und Kybernetik* 27 (Jan. 1991), pp. 263–271.

[23] Flavio Keidi Miyazawa and Yoshiko Wakabayashi. "An algorithm for the three-dimensional packing problem with asymptotic performance analysis." In: *Algorithmica* 18.1 (1997), pp. 122–144.

[24] Ping Josephine Xu, Russell Allgor and Stephen C. Graves. "Benefits of Reevaluating Real-Time Order Fulfillment Decisions." In: *Manufacturing & Service Operations Management* 11.2 (Apr. 2009), pp. 340–355. DOI: 10.1287/msom.1080.0222.

[25] Stefanus Jasin and Amitabh Sinha. "An LP-Based Correlated Rounding Scheme for Multi-Item Ecommerce Order Fulfillment." In: *Operations Research* 63.6 (Dec. 2015), pp. 1336–1351. DOI: 10.1287/opre.2015.1441.

[26] Eric T Bell. "Exponential Polynomials." In: *The Annals of Mathematics* 35.2 (Apr. 1934), p. 258. DOI: 10.2307/1968431.

[27] H. W. Becker and John Riordan. "The Arithmetic of Bell and Stirling Numbers." In: *American Journal of Mathematics* 70.2 (Apr. 1948), p. 385. DOI: 10.2307/2372336.

[28] Rein Peterson, David F. Pyke and Edward A. Silver. *Decision systems for inventory management and production planning*. en. 3rd ed. Brisbane, QLD, Australia: John Wiley and Sons (WIE), Jan. 1998. ISBN: 978-0471119470.

[29] Ronald L. Graham, Donald E. Knuth and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd ed. Reading, MA: Addison-Wesley, 1994. ISBN: 0-201-55802-5.

[30] The On-Line Encyclopedia of Integer Sequences. *A000110 - Bell or exponential numbers: number of ways to partition a set of n labeled elements.* URL: https://oeis.org/A000110 (visited on 12/04/2023).

[31] George E. Andrews. *The theory of partitions*. en. Cambridge, England: Cambridge University Press, July 1998. ISBN: 052163766X.

[32] Donald E. Knuth. *The art of computer programming*. en. 3rd ed. Boston, MA: Addison Wesley, July 1997. ISBN: 0-201-89683-4.

[33] The On-Line Encyclopedia of Integer Sequences. *A000041 - $a(n)$ is the number of partitions of n (the partition numbers)*. URL: https://oeis.org/A000041 (visited on 12/04/2023).

[34] Patricia van Loon, Lieven Deketele, Joost Dewaele, Alan McKinnon and Christine Rutherford. "A comparative analysis of carbon emissions from online retailing of fast moving consumer goods." In: *Journal of Cleaner Production* 106 (Nov. 2015), pp. 478–486. DOI: 10.1016/j.jclepro.2014.06.060.

[35] PostNord. *Price list for credit customers*. As the prices often update, this URL might be broken. We used the price lists that went into effect on 02/05/2023. The latest price list can be found on https://www.postnord.se/en/sending/prices-and-terms/price-lists-for-service-agreement-customers. 2023. URL: https://api2.postnord.com/rest/customer/v2/ptm/file/download/5347.26747?disposition=inline (visited on 16/05/2021).

[36] PostNord. *Price lists*. 2023. URL: https://www.postnord.se/en/sending/prices-and-terms/price-lists-for-service-agreement-customers (visited on 16/05/2021).

[37] The European Federation of Corrugated Board Manufacturers. *Annual statistics 2021*. 2022. URL: https://www.fefco.org/sites/default/files/files/Fefco_AnnualEvaluation_2021%288%29.pdf.

[38] Line K. Brogaard, Anders Damgaard, Morten B. Jensen, Morton Barlaz and Thomas H. Christensen. "Evaluation of life cycle inventory data for recycling systems." In: *Resources, Conservation and Recycling* 87 (June 2014), pp. 30–45. DOI: 10.1016/j.resconrec.2014.03.011.

[39] The European Federation of Corrugated Board Manufacturers. *FEFCO Code*. 2022. URL: https://www.fefco.org/sites/default/files/documents/FEFCO%20Code_WEBprotected.pdf.

[40] The Network for Transport Measures. *About NTM*. URL: https://www.transportmeasures.org/about-ntm/ (visited on 22/03/2023).

[41] Paul G. Boulter and Ian. S McCrae. "ARTEMIS: Assessment and reliability of transport emission models and inventory systems-Final Report." In: *TRL Published Project Report* (2008). ISSN: 0968-4093.

[42] The Network for Transport Measures. *Calculation method issues*. URL: https://www.transportmeasures.org/wiki/manuals/road/calculation-method-issues/ (visited on 23/03/2023).

[43] *Fuel consumption*. URL: https://www.transportmeasures.org/wiki/manuals/road/fuel-consumption/ (visited on 23/03/2023).

[44] The Network for Transport Measures. *Description of the road transport system*. URL: https://www.transportmeasures.org/wiki/manuals/road/description-of-the-road-transport-system/ (visited on 22/03/2023).

[45] *Vehicle type characteristics and default load factors*. URL: https://www.transportmeasures.org/wiki/manuals/road/vehicle-types-and-characteristics/ (visited on 24/03/2023).

[46] *Road cargo transport baselines Sweden*. URL: https://www.transportmeasures.org/wiki/evaluation-transport-suppliers/road-transport-baselines-sweden/ (visited on 24/03/2023).

[47] Statistics Sweden. *Beskrivning av Sveriges befolkning 2006*. Swedish. 2007. URL: https://share.scb.se/ov9993/data/publikationer/statistik/_publikationer/be0101_2006a01_br_be0107text.pdf (visited on 17/05/2023).

[48] Statistics Sweden. *Open data for grid statistics*. URL: https://www.scb.se/en/services/open-data-api/open-geodata/grid-statistics/ (visited on 31/03/2023).

[49] Open Geospatial Consortium. *GeoPackage*. URL: https://www.geopackage.org/ (visited on 31/03/2023).

[50] Michael Sipser. *Introduction to the Theory of Computation*. 3rd ed. Cengage Learning, June 2012. ISBN: 978-1-133-18779-0.

[51] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990. ISBN: 0-7167-1045-5.

[52] Alan Burns and Andy Wellings. *Real-time systems and programming languages*. 4th ed. International Computer Science Series. Addison-Wesley Educational, May 2009. ISBN: 978-0-321-41745-9.

[53] Erik Svanes, Mie Vold, Hanne Møller, Marit Kvalvåg Pettersen, Hanne Larsen and Ole Jørgen Hanssen. "Sustainable packaging design: a holistic methodology for packaging design." In: *Packaging Technology and Science* 23.3 (Feb. 2010), pp. 161–175. DOI: 10.1002/pts.887.

[54] Daniel Hellström and Mazen Saghir. "Packaging and logistics interactions in retail supply chains." In: *Packaging Technology and Science* 20.3 (2007), pp. 197–216. DOI: 10.1002/pts.754.

[55] Dwi Arman Prasetya, Phong Thanh Nguyen, Rinat Faizullin, Iswanto Iswanto and Edmond Febrinicko Armay. "Resolving the shortest path problem using the haversine algorithm." English. In: *Journal of Critical Reviews* 7.1 (Jan. 2020), pp. 62–64. ISSN: 2394-5125. DOI: 10.22159/jcr.07.01.11.

[56] Douglas C. Montgomery. *Design and analysis of experiments*. 9th ed. John Wiley & Sons, Inc., 2017.

[57] Richard S. Barr, Bruce L. Golden, James P. Kelly, Mauricio G.C. Resende and William R. Stewart. "Designing and reporting on computational experiments with heuristic methods." In: *Journal of heuristics* 1 (1995), pp. 9–32. DOI: 10.1007/BF02430363.

[58] Catherine C. McGeoch. "Toward an experimental method for algorithm simulation." In: *INFORMS Journal on Computing* 8.1 (1996), pp. 1–15. DOI: 10.1287/ijoc.8.1.1.

[59] Encore. *Infrastructure Provisioning*. URL: https://encore.dev/docs/deploy/infra (visited on 02/05/2023).

[60] Dean M. Tullsen, Susan J. Eggers and Henry M. Levy. "Simultaneous multithreading." In: *Proceedings of the 22nd annual international symposium on Computer architecture - ISCA '95*. ACM Press, 1995. DOI: 10.1145/223982.224449.

[61] Google Cloud Platform. *CPU platforms*. 2023. URL: https://cloud.google.com/compute/docs/cpu-platforms (visited on 25/05/2023).

[62] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes and João Saraiva. "Energy efficiency across programming languages: how do energy, time, and memory relate?" In: *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. ACM, Oct. 2017. DOI: 10.1145/3136014.3136031.

[63] Sven Winkelhaus and Eric H. Grosse. "Logistics 4.0: a systematic review towards a new logistics system." In: *International Journal of Production Research* 58.1 (May 2019), pp. 18–43. DOI: 10.1080/00207543.2019.1612964.

[64] Isabelle von Gertten, Anders Hildingsson, Joakim Jägare, Joakim Levin, Emma Maraschin, Cecilia Nyqvist, Erika Nysäter, Anna Pettersson, Helene Rosang, Simon Törnvall, Alexander Viksten, Zdravka Zulj and Anna Åberg. *Svensk postmarknad 2023*. Swedish. Post- och telestyrelsen. 5th Apr. 2023. URL: https://www.pts.se/globalassets/startpage/dokument/icke-legala-dokument/rapporter/2023/post/svensk-postmarknad-2023.pdf (visited on 17/05/2023).

[65] European Parliament, Council of the European Union. *Regulation (EU) 2019/1242 of the European Parliament and of the Council of 20 June 2019 setting CO2 emission performance standards for new heavy-duty vehicles and amending Regulations (EC) No 595/2009 and (EU) 2018/956 of the European Parliament and of the Council and Council Directive 96/53/EC*. 2019. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32019R1242 (visited on 17/05/2023).

[66] Avita Katal, Susheela Dahiya and Tanupriya Choudhury. "Energy efficiency in cloud computing data center: a survey on hardware technologies." In: *Cluster Computing* 25.1 (Oct. 2021), pp. 675–705. DOI: 10.1007/s10586-021-03431-z.

[67] Avita Katal, Susheela Dahiya and Tanupriya Choudhury. "Energy efficiency in cloud computing data centers: a survey on software technologies." In: *Cluster Computing* (Aug. 2022). DOI: 10.1007/s10586-022-03713-0.

[68] Živilė Baubonienė and Gintarė Gulevičiūtė. "E-commerce factors influencing customers' online shopping decision." In: *Social Technologies* 5 (2015). DOI: 10.13165/st-15-5-1-06.

A Generating all possible partitions of an order

Algorithm A.1 Python code for generating all possible partitions of an order

```
1: def partition(collection: list[str]) -> list[list[str]]:
2:     if len(collection) == 0:
3:         yield []
4:         return
5:
6:     if len(collection) == 1:
7:         yield [collection]
8:         return
9:
10:    first = collection[0]
11:    for smaller in partition(collection[1:]):
12:        for n, subset in enumerate(smaller):
13:            yield sorted(smaller[:n] + [[first] + subset] + smaller[n+1:])
14:        yield sorted([[first]] + smaller)
15:
16:
17: def get_splits(collection: list[str]) -> list[list[str]]:
18:    return unique(partition(collection))
```

B Time complexities of all heuristics

Table B.1: Time complexities of the heuristics.

Name	Time complexity
Full split	$O(n)$
No split	$O(n^2)$
Split by volume	$O(n^2 \log n)$
Biggie smalls	$O(n^4)$
Eliminate bad fill rate	$O(n^4)$
Group while fits	$O(n^4)$
Exhaustive search	$O(B_n n^3)$