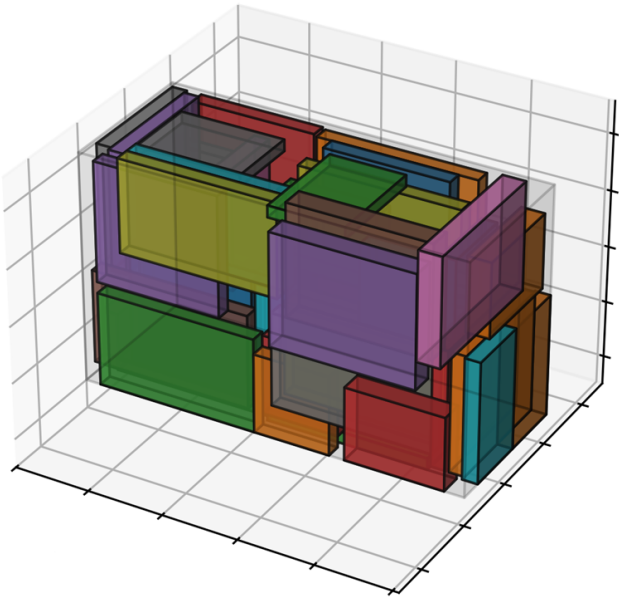
CHALMERS
UNIVERSITY OF TECHNOLOGY



Algorithmic Optimization of Box Selection for Warehouse Operations

Evaluating the 3D Bin Packing Problem in E-Commerce Supply Chains

Master's thesis in Systems, Control and Mechatronics

KALLE FUNCK
KRISTOFER GUSTAVSSON WAIDRINGER

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Algorithmic Optimization of Warehouse Box Selection

Evaluating the 3D Bin Packing Problem in E-Commerce Supply Chains

KALLE FUNCK
KRISTOFFER GUSTAVSSON WAIDRINGER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2025

Algorithmic Optimization of Warehouse Box Selection
Evaluating the 3D Bin Packing Problem in E-Commerce Supply Chains.
KRISTOFFER GUSTAVSSON WAIDRINGER
KALLE FUNCK

Supervisor: Johan Härdmark, Ongoing Warehouse AB
Examiner: Bengt Lennartsson, Department of Electrical Engineering

Master's Thesis 2025
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A packing solution generated by one of the algorithms.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Algorithmic Optimization of Warehouse Box Selection
Evaluating the 3D Bin Packing Problem in E-Commerce Supply Chains
KALLE FUNCK
KRISTOFFER GUSTAVSSON WAIDRINGER
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The efficiency of e-commerce logistics chains is heavily influenced by bin selection during the packing process, which can be formulated as a variant of the NP-hard three-dimensional bin packing problem: selecting the smallest possible combination of heterogeneous bins that can fit a given order. This thesis presents the implementation and evaluation of several solving techniques for this problem, with a focus on achieving runtimes suitable for real-world applications. Following a survey of what solutions currently exist in the literature, the four commonly found methods are heuristics, Genetic Algorithms (GAs), Deep Reinforcement Learning (DRL), and Mixed-Integer Linear Programming (MILP) solvers. These are all evaluated throughout this thesis, as well as the CP-SAT solver, which is capable of handling MILP problem formulations. Contrary to common findings within the literature, which consider GAs and DRL methods as state-of-the-art, this thesis shows that when replacing the traditional MILP solvers with CP-SAT, this becomes a viable alternative that can provide increased packing efficiency in a feasible amount of time. Heuristic solutions can also be a suitable option in certain scenarios, for instance, when minimizing computational load is highly prioritized.

This thesis concludes that it is possible to generate solutions to this bin packing problem using CP-SAT within feasible runtimes. While human workers may outperform the solver in some cases, this is primarily due to limitations within the current model. The solver is ready to be implemented if a warehouse meets a few key conditions: items should be well approximated by rectangular prisms, and the input data must be accurate and reliable. Given the NP-hard nature of the problem, performance is better when the number of items remains within normal operational bounds. Additionally, the availability of appropriately sized packing cartons is important. Without this, the benefits of the bin selection automation are significantly reduced.

Keywords: 3D-BPP, Bin Packing Problem, Heuristics, EMS, Genetic Algorithm, Deep Reinforcement Learning, CP-SAT, Proximal Policy Optimization, NP-Hard

Acknowledgements

We would like to extend our gratitude to our examiner Bengt Lennartsson and our supervisor Johan Härdmark for providing valuable insight and support throughout the entire thesis. We would also like to thank Ongoing Warehouse for making this thesis possible.

Kalle Funck, Kristoffer Gustavsson Waidringer, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis, listed in alphabetical order:

3D-BPP	Three-Dimensional Bin Packing Problem
BPP	Bin Packing Problem
BPS	Bin Packing Sequence
BCI	Bin Combination Index
BB	Bounding Box
CP	Constraint Programming
DBLF	Deep-Bottom-Left Fill
DoF	Degrees of Freedom
DRL	Deep Reinforcement Learning
DQL	Deep Q-Learning
DQN	Deep Q-Network
EMS	Empty Maximal Space
EP	Extreme Point
GA	Genetic Algorithm
GAN	Generative Adversarial Network
LCG	Lazy Clause Generation
LP	Linear Programming
MILP	Mixed-Integer Linear Programming
NN	Neural Network
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SAM	Surface Area Minimization
SF	Smallest First
VBO	Vector of Box Orientations
WMS	Warehouse Management System

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

$i,\ k$	Indices for products
j	Index for bins
t	Index for time step

Sets

$\mathcal{B}$	Set of available bins

Parameters

α	Learning rate
β	Maximum amount of bins to pack in
ϵ	Clipping fraction
γ	Discount factor
λ	Scaling factor
n_p	Number of products
n_b	Number of bins
k	Scaling parameter for the PF dataset
p_i	Product i
b_j	Bin j
G	Generations for the Genetic Algorithm
N	Population size for the Genetic Algorithm

Top	Number of Elites for the Genetic Algorithm
p_{mut}	Mutation rate for the Genetic Algorithm
k_{tour}	Tournament probability for the Genetic Algorithm
N_{bias}	Biased initiations for the Genetic Algorithm
σ_{mut}	Mutation deviation for the Genetic Algorithm
p_i, q_i, r_i	Length, width and height of product i within the MILP model
$L_j, W_j, H_j,$	Length, width and height of bin j within the MILP model

Variables

x_i, y_i, z_i	Continuous integer variables representing the position of product i within the MILP model.

Contents

List of Figures

List of Tables

1

Introduction

E-commerce is increasing and is putting constant pressure on supply chains to adapt [1]. A key component of these business-to-customer chains is the warehouses, whose primary responsibilities include storing and packing products. Once a customer places an order on an online website, it is eventually forwarded to a warehouse worker who is tasked with selecting a suitable set of boxes for packing the products [2]. These boxes can range in size, and the goal is to pack all ordered items in the selected ones [3], which can be described as a variation of the classical three-dimensional bin packing problem (3D-BPP), a well-known NP-hard problem [4]. To minimize the risk of being unable to fit all products, the worker may select a box with excess spare volume. This reduces transportation density, leading to an increased number of shipments, which finally results in a greater environmental impact and reduced profitability. Implementing a more effective box-picking strategy will therefore enhance the supply chain, providing benefits for both the warehouse and the environment.

The 3D-BPP involves fitting different items into bins, traditionally aiming to minimize the number of bins used [5]. This project addresses a variation of the problem, where instead of minimizing the number of identical bins, the objective is to minimize the total bin volume from a set of available bin sizes to better reflect e-commerce warehouses. Given that the 3D-BPP is NP-hard [4], achieving a solution that is both computationally efficient and practical for real-world warehouse operations is challenging. This project compares various algorithmic strategies and benchmarks their performance against the decisions made by warehouse workers. This evaluation is essential to determine whether or not the automation of the box selection process provides benefits. A previous master's thesis produced a well-functioning algorithm for this problem [6]; however, its high computational complexity resulted in long run times that limited its practical application. Building on that work and other research in the field, this project aims to implement an algorithm for the multiple bin size 3D-BPP that delivers an effective suggestion of bins while maintaining a feasible runtime. This project also seeks to explore and compare a wide range of methods that are presented throughout the literature.

This thesis was conducted in collaboration with Ongoing Warehouse which develops and maintains a cloud-based Warehouse Management System (WMS).

1.1 Research Question

This project aims to build a model of the problem and identify a suitable optimization algorithm that, within reasonable time limits, suggests boxes with the least amount of spare volume. While the project will touch on other areas, there is one main question that is the focus point of this report:

- Is it possible to implement a solving method for the multiple bin-size 3D-BPP that outperforms human workers in terms of packing efficiency while maintaining a feasible runtime?

Even though this thesis is centered around only one research question, it inevitably raises several secondary questions that must be addressed to fully explore and answer the main research question. These questions consist of what solvers are well-suited to this type of problem? How do human workers currently perform in comparison? And what is an appropriate way to model and evaluate the packing problem?

1.2 Limitations

One major limitation of this project is that all items will be considered rectangular prisms, as the available data only specifies the height, width, and depth. More information describing softness and irregular shapes could potentially improve the box-picking strategy and is, therefore desirable. Another limitation is that this project will not take into account if the packages are fragile, something that could alter the packing order and overall strategy. Weight and packing stability will also not be considered.

2
Theory

This chapter outlines the theoretical foundation for the work presented in this thesis. It begins with a review of the 3D-BPP and its variations. Following this, relevant technical background is presented to introduce concepts to the reader, but also to lay a foundation for the decisions made throughout the study. Readers already familiar with these technical concepts may choose to skim or skip this chapter.

2.1 The Three-Dimensional Bin Packing Problems

The 3D-BPP has been studied extensively, and many variations and solutions exist [7]. Although all 3D-BPPs consist of packing items in bins, there are more aspects to consider. During this section, the variety in which the problem can present itself will be reviewed, something that is essential to know to determine what methods are suitable for solving it.

2.1.1 Problem Definition Variations

The specific 3D-BPP formulation variations depend on the real-world application, which can consist of packing containers, warehouse packing, or even conveyor belt robot packing [7]. The real-world problem then gives rise to four key questions that must be considered to determine how the 3D-BPP should be described.

- What to optimize?
- Offline or Online?
- Regular shape or irregular shape?
- What rotations are allowed?

Commonly, the optimization either strives to do input minimization or output maximization [8]. Input minimization consists of trying to minimize the number of identical packing bins, whereas output maximization includes trying to maximize the number of packages in a set number of bins. Even though these two optimization goals are frequent, there are more variations to be found. As an example, the following solution explores minimizing the total bin surface area [9].

There exist solvers that focus on online problems [10, 11]. In these cases, all the package sizes are not known and present themselves as time progresses. One example of such real-world systems is packing from conveyor belts [12]. On the other

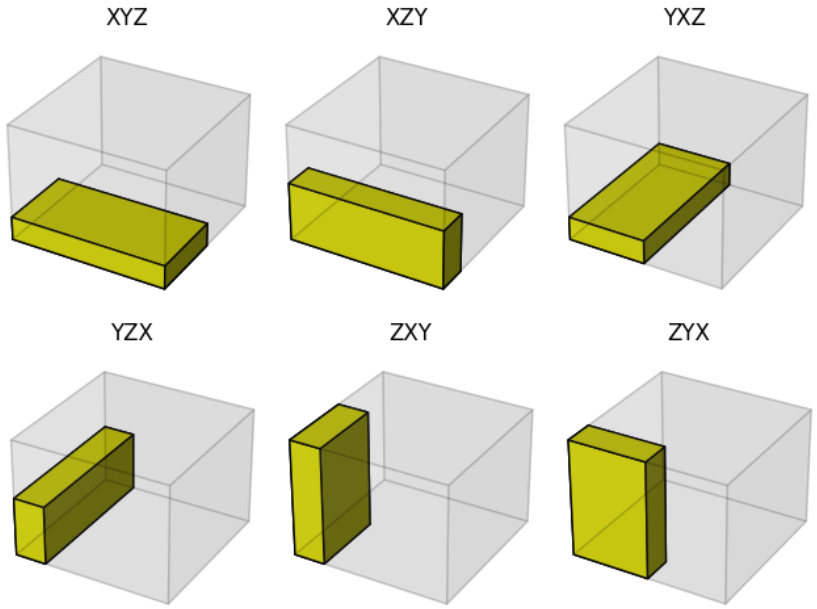


Figure 2.1: The six possible rotations when allowing for all orthogonal rotations.

end of this are offline problem descriptions, and in these cases, all the package sizes are known beforehand [7]. Such cases can present themselves when packing orders into containers [13, 14], and fitting boxes around orders in e-commerce [9]. Online systems are more difficult since less prior information is known [12].

Although much research done in the field revolves around rectangular prisms, there exists research where shapes are irregular [15][16]. This adds complexity but considering more shapes can provide a more practical picking strategy as it better reflects real-world scenarios [15]. An example of this is the packing of bolts, which can because of their hexagonal form, be packed in a qualitative manner that can not be replicated if approximated by rectangles.

Rotation is another aspect to consider where more rotation variations can improve packing density but at the cost of algorithmic computation time [17]. Some problem descriptions forbid all rotations [18], and some forbid rotations in certain dimensions [7]. This can be because certain packages need to be kept upright to minimize product damage [13]. Other 3D-BPP descriptions allow all possible orthogonal rotations, providing six possible poses for each package [19], which is visualized in Figure 2.1. It is even possible to include non-orthogonal rotations in the model [17].

2.1.2 Specific Project Problem Description

The specific variation of the 3D-BPP considered in this project is outlined below. It has been designed to align with the objectives and constraints of the study, and explanatory notes are included to give reasons for the decisions taken.

- **Objective:** The objective of this project is to minimize total bin volume from a set of known bins. Furthermore, it is not desirable to use too many bins when packing for two reasons: Firstly, the packing boxes come with a direct cost. Secondly, using a larger number of bins requires more cardboard, which harms both the environment and the packing efficiency. This thesis will primarily aim to decrease total bin volume, but it will try to do this while using a suitable number of bins. A suitable number ranges between one and two, but in some instances, three could be allowed.

- **Time frame:** Offline, when the order is placed by the customer, all packages and their sizes are known.

- **Shapes:** This project only focuses on rectangular prisms; this is a constraint from the available data. Width, height, and depth are the only known sizes.

- **Rotations:** Orthogonal rotations, allowing six possible orientations. This assumes that no risk is taken when placing an object upside down. The reason for not including more rotations is mainly for computational complexity, which is one key aspect of this thesis.

These constraints ensure the problem remains computationally feasible while reflecting e-commerce package deliveries. Now that the variations of the problems are described, the theory behind the methods capable of solving the 3D-BPP will be outlined.

2.2 Heuristic Methods

In terms of optimization, a heuristic algorithm operates until it can find a feasible solution and generally does not converge towards the optimal solution [20]. In 3D-BPP, a heuristic algorithm can present itself as a common packing strategy that functions for the general case. It is possible to merge different heuristics to form multi-heuristic algorithms, and they can also be combined with other algorithms, such as genetic algorithms (GA) [8], and deep reinforcement learning (DRL) [9]. In the 3D-BPP, heuristics are most frequently used combined with other optimization tools. The main benefit of these approaches is that they scale well since they build on logical, pre-defined methods, which limits the optimization search space. Ananno and Ribeiro even go so far as to state that: "For the general case, heuristic or hybrid approaches are the only reasonable strategy." [7] There are various forms of heuristics, and the following sections present some of the possible methods that can be utilized for the 3D-BPP.

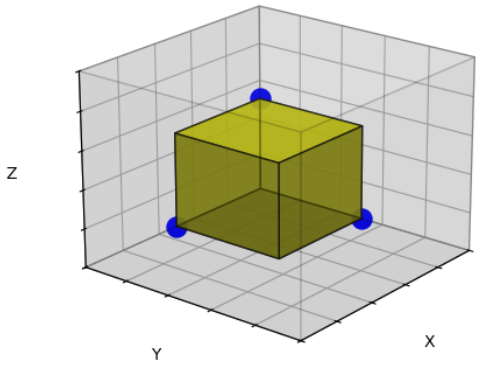


Figure 2.2: Visualization of the generated extreme points after one item placement iteration.

2.2.1 Heuristic Methods used for the 3D-BPP

This section aims to present commonly used heuristics found in the literature and explain their functionality. Since heuristic solutions rely on predefined strategies, they are often problem-specific. Therefore, the methods introduced here are general-purpose and well-suited to various 3D-BPP formulations.

Extreme points

Extreme points (EPs) strive to locate the closest non-occupied points from the origin, as traversed through an orthogonal path [21]. If there exists one EP at the point $(x_{ep},\ y_{ep},\ z_{ep})$, placing a block with size $(l,\ w,\ h)$ at that point will generate new EPs at the following places:

$$p_1 = (x_{ep} + l,\ y_{ep},\ z_{ep}) \tag{2.1}$$
$$p_2 = (x_{ep},\ y_{ep} + w,\ z_{ep}) \tag{2.2}$$
$$p_3 = (x_{ep},\ y_{ep},\ z_{ep} + h) \tag{2.3}$$

To visualize this, an item placed in the origin will generate the blue EPs seen in Figure 2.2. The strategy is to place boxes at these extreme points, which will build up a tightly packed sequence. Only allowing the system to place products at EPs will also drastically reduce the number of options, decreasing the amount of computations needed. A decision on how to prioritize which EP is then necessary to make, which could be to choose the lowest point, for example.

Empty Maximal Spaces

The idea of an empty maximal space (EMS) is to provide available unused bin space where an item can be placed [22]. It works by tracking the largest possible rectangular blocks that are not occupied by any products for the current iteration. Which EMS to choose for placement, similarly to EP selection, depends on what prioritization is used. Examples could be to maximize the utilization rate of any EMS or

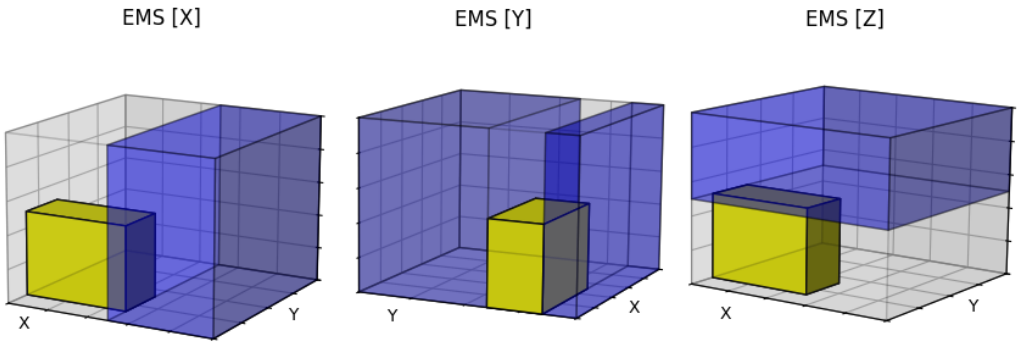


Figure 2.3: Visualization of Empty maximal spaces.

to choose the EMS that is closest to the bin corner. Figure 2.3 shows a yellow item placed at an arbitrary position, with surrounding EMSs illustrated by blue boxes. Both EPs and EMSs aim to provide suitable, strategic placement options.

Loading pattern generation

This strategy consists of building geometrical entities out of the products and iteratively filling out the bin with these entities. Two examples using this strategy are block-building and layer-building heuristics [7]. Block-building aims to create blocks out of similar packages and then place the blocks in the bin. It generates a list of items to create the box and then places it inside the bin. Successively, these blocks will fill out the bin. Horizontal layer building aims to create equal height layers to build upon [23]. Similar to block building, it dissects free volume in the bin and generates a list of items, but instead of creating blocks, it creates layers. This item layer is then placed in the free space of the bin, and the next iteration starts. This heuristic is especially suitable if there are a lot of similar items and if packing stability is of importance.

Placement priority

The deep-bottom-left fill (DBLF) strategy consists of always trying to place packages close to the bin corner. It is computationally efficient since only points in the bin that have the potential to be the most deep-bottom-left point have to be evaluated [24]. Since the packages are placed iteratively, it is possible to track these potential points through EPs or EMSs, and hence not having to search through the full bin. By greedily trying to keep items closer to the bin corner, it can also provide an effective bin packing sequence.

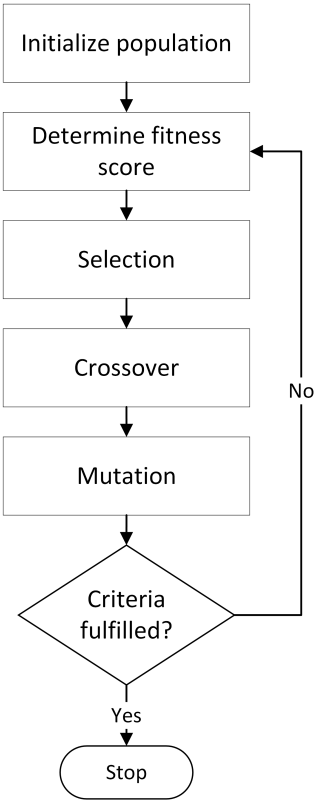


Figure 2.4: Flowchart of the genetic algorithm.

2.2.2 Final Remarks

Another concept worth mentioning is meta-heuristic algorithms. Meta-heuristics are solutions that rely on some kind of higher-level solver. Such as GAs, simulated annealing, or ant colony optimization. While meta-heuristic algorithms can rely on pure heuristic algorithms, they are not necessarily related. In summary, this section outlined key heuristic approaches commonly used in the literature, such as EPs, EMSs, block-building, and DBLF. More heuristics can be used, and a simple thing, such as sorting items in a specific size-wise manner, can also be considered a heuristic method. The next sections will explore methods categorized as either learning-based or search-based.

2.3 Genetic Algorithms

A GA is a stochastic optimization tool based on evolutionary theory and natural selection [25]. It is a method that can perform well at complex optimization problems even when the data set is large and unorganized. The full algorithm is illustrated by Figure 2.4 and is more thoroughly explained going forward.

2.3.1 Encoding and Initiation

Encoding is needed before implementing the iterative algorithm [26]. Each individual in the GA is defined by a set of parameters, encoded in a chromosome. These chromosomes are expressed as strings with bits, integers, or floats, depending on usage. An example relating to this project could be a string of integers containing information about each product's rotation, ranging from one to six [6]. This type of gene is known as a vector of box orientations (VBO), and to give an example, a chromosome when the number of items (n_p) is equal to eight items could look as follows.

$$\text{Example Chromosome: } \underbrace{\{0, 0, 1, 1, 5, 6, 5, 6\}}_{n_p=8}$$

The population is then randomly initialized to cover a wide search range of feasible solutions [27]. The total number of individuals makes up the population size (N) and is regarded as a tuning parameter. Increasing this parameter gives a wider initial search space, which increases the chance of finding the optimum [28]. The drawback is that the convergence time grows, mainly because of the necessity to compute the fitness score for each individual. After the population is initialized, it will run through the GA main loop for as many generations (G) as desired or until termination conditions are reached [26].

2.3.2 Fitness and Selection

The next step is to determine a fitness score, which is supposed to guide the population toward the optimization goal. In the case of standard bin packing, the fitness could be dependent on the number of bins used, which would cause the population to try and optimize this parameter [29]. The fitness can also penalize invalid solutions or behavior that is deemed as undesired by assigning it negative values.

During the selection phase, individuals are chosen for reproduction [30]. The goal of this process is to increase the likelihood of good genes being included in the offspring while allowing genes that affect fitness negatively to be phased out. However, it is not feasible to select only the individuals with the highest fitness in every iteration, as other chromosomes may contain genes that perform better in specific parts of the problem [27]. This becomes an exploration against exploitation problem, as too high a selection pressure can decrease population variety, limiting the spread in the solution space. In turn, this yields the risk of missing out on an optimum. A too-low pressure, on the other hand, can inhibit convergence since the optimization is not guided towards high-performing genes.

A common selection method is tournament selection [26]. In this approach, a fixed number of random individuals compete against each other, and the one with the highest fitness value is selected as a parent. This process is repeated until a suffi-

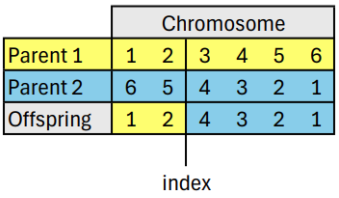


Figure 2.5: Single-point crossover for randomly chosen index.

cient number of parents have been chosen. The number of tournament participants directly affects selection pressure. Another form of selection method is roulette selection, where each gene has a chance equal to its normalized fitness score to be selected as a parent. During the selection phase, it is also possible to directly add the best solutions to be included in the new population. This process is known as elitism, which will stabilize the optimization since there is no risk of forgetting good solutions. However, sometimes it can be necessary to forget genes to get over a local optimum.

2.3.3 Crossover and Mutation

Once the parents are selected, the next step is to perform crossovers [25]. During this process, genes from parent individuals are combined to create offspring for the next generation. There are many variations of crossover methods, ranging in complexity. A common example is to choose a random crossover point and take the left part from one parent and the right part from the other. The method is known as single-point crossover and is illustrated in Figure 2.5. A slightly more complex example is a multi-point crossover, which involves taking multiple parts of the gene and recombining them. This increases offspring variety since the number of combinations in which a chromosome can be dissected increases.

To be able to discover new solutions, mutation is introduced, which is an event where an offspring's gene is altered randomly [26]. The simple form of this is to decide a mutation rate and then simply at random instances, flip parts of the chromosome. If the chromosome is continuous, mutation can be done through a random selection of Gaussian distributions centered around the current value. Similar to selection pressure, deciding on the mutation rate is a trade-off between exploration and exploitation. Where a high mutation rate might increase the chance of exploring the optimum, it decreases the chance that it will converge to it.

2.3.4 General remarks

Designing a genetic algorithm presents the developer with the challenge of tuning various parameters. This process can be quite problematic and often requires deep knowledge of the specific problem domain [26]. The No Free Lunch theorem states that no single set of parameters can be universally applied across different problems; what works for one problem may not work for another. Additionally, GAs

are stochastic, which means that on the same problem, the same GA can provide different results. Restarting the algorithm multiple times (at least five) can be beneficial, as the initial population has a significant effect on where the algorithm tends to settle. Its stochastic nature can negatively affect the algorithm's robustness since it is not possible to know if any amount of restarts or generations will improve the packing algorithm. However, a GA can offer a fast and efficient solution and is a commonly used method to approach the 3D-BPP. With the theory behind GA covered, the next section turns to background information on DRL.

2.4 Deep Reinforcement Learning

Another option for optimizing a packing algorithm is through DRL, which can be used to solve combinatorial problems such as the 3D-BPP [9]. A key advantage of DRL is its ability to be pre-trained, enabling it to learn general patterns and strategies. Once trained, it can be applied efficiently with relatively fast computation times. A GA, for example, can not learn from earlier tasks and tackles each problem individually in an exhaustive manner. Figure 2.6 shows a DRL algorithm which consists of an environment and an agent in the form of a neural network, where the environment provides the agent with an observation so that the agent can predict the best possible action to take and is then given a reward from the environment which allows the agent to learn from it's interactions with the environment. This section will explain these parts in depth starting with reinforcement learning (RL), in order to later explain DRL.

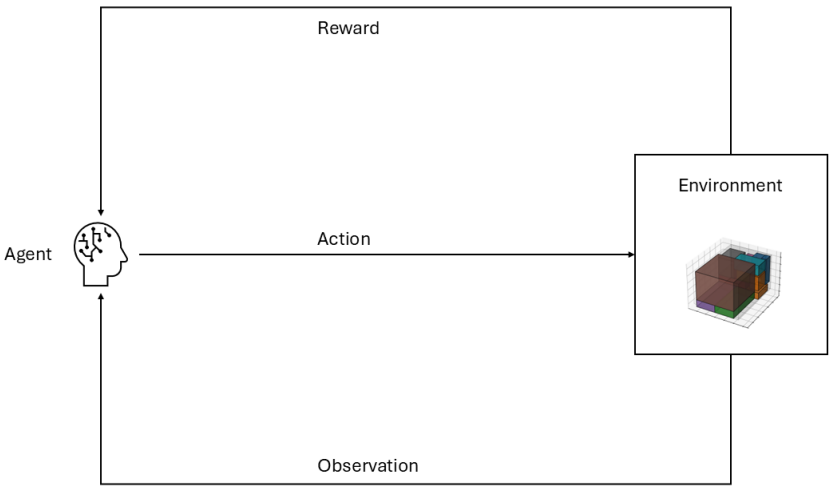


Figure 2.6: Deep Reinforcement Learning algorithm.

2.4.1 Reinforcement Learning Functionality

RL works by allowing an agent to interact with an environment and rewarding actions if they lead to desirable outcomes [31]. The full iteration consists of the

agent observing a state s_t, and based on that it decides on an action a_t. This action provides a reward r_{t+1} and then the environment, based on the state-action pair, (s_t, a_t), provides the agent with the next state s_{t+1}. The total set of states and actions is known as the action space (A) and state space (S).

States, Actions, and Rewards

The observed state is the information that is available to the agent [32]. In the context of an online 3D-BPP, this observation can include details of what bin spaces are free and what product is next to be placed [9]. Based on this knowledge, the agent will then try to make a good decision on where to place the next product. Therefore, it is beneficial to include only relevant information in the observed state; any data that does not influence the decision should be excluded from S to minimize computational load and to reduce the risk of the agent drawing incorrect conclusions. However, it is of utmost importance to include all data that is important for deciding on an action. Ultimately, it is the developer's responsibility to determine what the agent should observe [32].

The action should then contain information on how the agent can act in the environment. In the case of 3D-BPP, an action could, for example, describe the placement of an item by including information about its rotation and bin position coordinates. After the action has been conducted, it is then up to the RL designer to value how this action should be rewarded, i.e., deciding on $r(s, a)$. An action should be rewarded if it helps optimize the end goal. An example of this could be to minimize the number of bins if dealing with the standard 3D-BPP objective. If a solution has a lower number of bins compared to other ones, it should have a higher accumulated reward, although all rewards do not have to be directly associated with the final result. Intermediate rewards serve to guide the agent towards paths that have a higher potential to optimize the end goal. For example, one possibility is to include heuristics and provide a bonus for selecting an EP as the origin. It is also possible to rely more heavily on heuristics by outsourcing responsibilities. One example could be to do the sorting with DRL and then perform the rest with heuristic methods [9]. Undesirable behaviors or invalid solutions can be handled by assigning negative rewards. Additionally, certain states can be defined as terminal to stop the agent from exploring those paths. If actions that lead to invalid or undesirable behavior can be identified in advance, by utilizing action masking, it is possible to discard them. This reduces the number of available actions and, in turn, limits the search space.

Q-learning

Q-learning is a classical RL algorithm, and in this algorithm, the rewards are summed and discounted by a factor γ as the agent traverses further in the future. The discount factor is partly determined by the amount of uncertainty and risks the system has, since if the system is stochastic, future rewards are not guaranteed. A learning rate, α, is a hyperparameter that controls how much the algorithm adapts

to new information. It generally ranges between 0 and 1 and is often reduced over time. The purpose of the decrement is to allow finer adjustments as the model approaches convergence. The Q-value equation can be viewed below.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a \in A} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$$

In Q-learning, the action $a \in A$ that maximizes the Q-value is selected. To be able to find new paths, a chance of taking random actions every iteration is introduced. The tuning of the RL parameters is similar to the GA in the sense that both need to balance exploration and exploitation.

After the Q-learning agent is constructed, it iterates and creates a table where it saves the best Q-value for each state-action pair. The best policy is, hence, simply taking the action with the highest Q-value at each state until a terminal state is reached. While Q-learning might function for smaller problems where it is possible to traverse through all the states, it might not be possible for more complex systems. As the state and action space grows larger, the Q-table will expand exponentially. This major limitation creates the need for a more scalable approach, where neural networks (NNs) offer this through their ability to generalize, approximate, and handle high-dimensional data. Although Q-learning is not well-suited for solving the 3D-BPP due to the problem's high complexity and large state-action space [33], it serves as a useful foundation for understanding more advanced DRL methods explained in the following sections. Furthermore, the hyperparameters γ and α discussed earlier remain relevant for DRL approaches.

2.4.2 Neural Networks

A brief overview of NNs is important since they serve as a foundation for the understanding of DRL. There are different versions of DRL, but in all of them, NNs serve to try and suggest the best action from the current state.

Neural networks are similar to GAs, inspired by biology, but instead of focusing on evolution, they draw inspiration from the brain [34]. NNs consist of neurons, which can be described as a collection of functions. These functions are known as activation functions and take a vector of inputs X and a vector of weights W to produce an output y. The output is then sent to the neurons in the next layer with their own activation functions. This process is then done across all the neurons and can be described through the following equation.

$$y = f \left(\sum_i w_i x_i + b \right)$$

Introducing non-linearity to an NN enables it to learn more complex patterns. This is typically achieved by using a non-linear activation function such as Sigmoid, Tanh, or ReLU [35].

The neurons of modern networks are organized into layers, which determine in what order they will process the data. The first layer is known as the input layer, and this

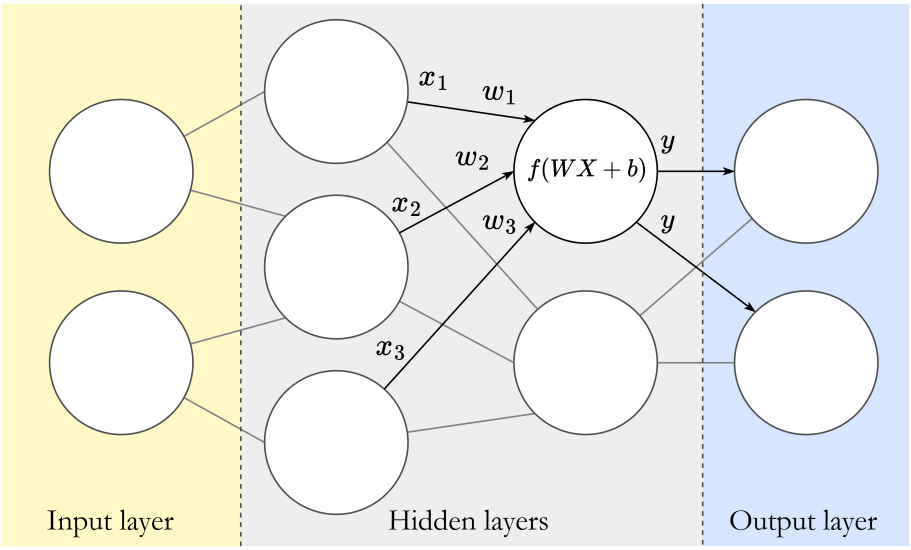


Figure 2.7: Visualization of a neural network.

is where the data is introduced. The number of neurons in this layer is therefore determined by the size of the data. After the input layer comes the hidden layers, which can be any number of layers with possibly varying dimensions. The optimal amount of hidden layers and neurons is a tuning problem that is often solved by trial and error [34]. NNs can have fully connected layers, where each neuron outputs data to all neurons in the next layer, as well as weakly connected layers, which connect to only a subset of the next layer. Figure 2.7 presents a fully connected NN with two hidden layers. The number of neurons for this example is arbitrarily chosen only to get a visual representation of a NN. Finally, the output layer is reached, where the data is presented in the format required. For a classification problem, the number of neurons usually equals the number of classes.

Since the outputs of a NN typically are raw values that represent different classes, it is useful to transform them into a format that resembles probabilities, constrained between 0 and 1. This is achieved using the Softmax activation function, as shown in the equation below. The benefit of this function is that it converts the network's outputs into a probability-like distribution, where the values sum to 1, making predictions easier to interpret.

$$\text{Softmax}(z_i) = \frac{e^{zi}}{\sum_j e^{z_j}}$$

Weights are a crucial component of the NNs, determining how much influence an input has on the output. The effectiveness of a neural network depends on how well the weights are optimized. The weights are optimized during the training process, which is illustrated in Figure 2.8. To exemplify the training process, we consider a NN tasked with predicting the best action a_t^* from a known state s_t, which aligns with this chapter's focus on DRL.

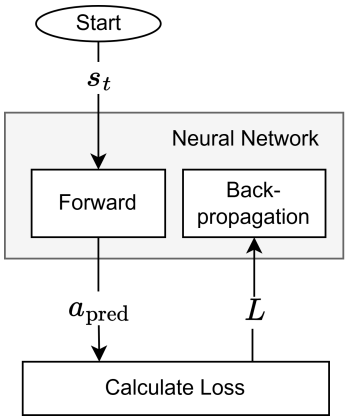


Figure 2.8: Neural network Training iteration.

During the training iteration, s_t is loaded into the input layer and propagated forward through the network. The output layer then generates the predicted optimal action $a_{t,\text{pred}}$, which is eventually evaluated by the loss function. The more $a_{t,\text{pred}}$ coincides with the optimization goal, the lower the loss it receives. This loss ultimately determines how much the weights should change during the next step of backpropagation. When the loss is returned, instead of going forward through the layers, the loss is traversed backward through the network, adjusting the weights using gradient descent [36]. The step size is determined by the learning rate α.

It is common to split the data into training and validation to avoid overfitting. The training iteration is then done once for every data point in the training set to complete an epoch. The training is a time-consuming task and requires the agent to iterate a large number of times, which can take hours or even days, depending on the complexity and volume of the data.

2.4.3 Deep Reinforcement Learning

As mentioned earlier, for large S and A, the number of possible actions becomes too large to effectively use Q-learning. That's where DRL comes in; instead of relying on exhaustive path traversal for a single problem, DRL utilizes a NN to estimate the best action for the current state based on the trained weights. There exist multiple different forms of DRL, but the ones that will be reviewed in this thesis will be Deep Q-learning and Proximal Policy Optimization (PPO). The reason for selecting these is that they are commonly found in the literature for combinatorial problems [37]. Another benefit of using these is that they are already implemented in Stable baselines [38], and can hence be easily compared without having to implement all algorithms from scratch. Using the library, it is also possible to easily experiment with different encoders, which are commonly used for combinatorial problems such as the 3D-BPP [37]. Encoders can consist of pointer networks, NNs, or even transformers [39]. Another common DRL method for the 3D-BPP is Advantage Actor-Critic [37]. However, its behavior can be effectively replicated through PPO [40], and hence it is not evaluated further.

Deep Q-learning

Contrary to standard Q-learning, where Q-values are learned by iteratively exploring all possible paths, DQL uses a neural network commonly referred to as the Deep Q-Network (DQN) to approximate the Q-values based on the state and the available actions [41]. In DQL, two neural networks are employed: the DQN, which estimates Q-values for a given state, and the target network, which is a copy of the DQN that updates less frequently. The use of a target network helps stabilize training by providing more consistent target values during learning.

Proximal policy optimization

PPO is an actor-critic form of DRL that makes use of two NNs, one actor deciding on actions, and one critic trying to estimate future rewards. The actor is hence a policy network that predicts what action would yield the largest reward, and the critic is a value network that predicts the reward for the current policy. PPO builds upon a policy gradient method, and its objective function is presented below.

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log\ \pi_\theta(a_t|s_t)\hat{A}_t]$$

While performing multiple optimization steps on this loss seems intuitive, using the same trajectory for each step can lead to large and destabilizing policy updates [42]. Therefore, trust region policy optimization introduces an objective function that is maximized under a constraint of how much the updated policy is allowed to differ from the old policy which is presented below.

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t] \\ &\text{subject to}\ \hat{\mathbb{E}}_t\Big[\mathrm{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\Big] \leq \delta \end{aligned}$$

where KL is the Kullback-Leibler divergence function [43], and by constraining the KL divergence the updated policy can't differ to far from the previous policy, the ratio $r_t(\theta)$, is the probability ratio that shows how likely the new policy is to take a specific action compared to the old policy. Shown below

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

The policy $\pi_\theta(a_t|s_t)$ gives the probability of selecting action a_t given that the agent is in the state s_t. The advantage estimate $\hat{A}_t$ tells us how much the reward from taking the action a_t differs from the expected value. Calculated from the temporal difference error δ_t as follows:

$$\delta_t = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$$

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

In this case, T is the number of timesteps the policy has been run before the next update. The discount factor, γ determines how much weight the future reward

holds compared to the immediate rewards, where a low γ yields higher focus on immediate rewards and a high γ yields higher focus on the future rewards. The scaling factor λ is a hyperparameter that can balance the bias and variance in the advantage estimate where $\lambda = 0$ uses only one step of δ_t and $\lambda = 1$ uses many steps to estimate the advantage i.e. a low λ gives a higher bias and lower variance in the advantage estimate whilst a high λ reduces the bias and increases the variance. If the advantage estimate is positive, the reward was better than expected, and if it is negative, the reward was worse than expected. With the use of this constraint however it introduces a need for second order optimization which drastically decreases the computational efficiency. What differs the PPO from trust region policy optimization is the introduction of a clipping function to remove the constraint. By removing the constraint the objective can now be optimized by a first order optimizer, the introduction of the clipping function looks like:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min\Big(r_t(\theta) \cdot \hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)]$$

The clipping function forces the loss to be constrained, which in turn forces the policy to only make small updates. This, in turn, provides a more stable training sequence. The hyperparameter ϵ is a small number, typically 0.1 or 0.2. L^{CLIP} can now be traversed backwards through the policy network using backpropagation.
The value network's task is to predict what value the current policy $\pi_\theta(s|a)$ will result in given the current state s_t. The Value networks objective is the normal MSE loss calculated as follows:

$$L^{VF} = \frac{1}{2}\sum_t \Big(V_\theta(s_t) - V_t^{target}\Big)^2$$

where $V_\theta(s_t)$ is the predicted value for the state s_t and V_t^{target} is the target value for state s_t. Since the Policy and Value networks often share layers the total objective function for the networks takes the following form:

$$L_t(\theta) = \hat{\mathbb{E}}_t\Big[L_t^{CLIP} - c_1 L_t^{VF} + c_2 S[\pi_\theta(\cdot|s_t)]\Big]$$

where c_1 is a scaling of how important the value loss is. The third term $S[\pi_\theta(\cdot|s_t)]$ is the policy entropy which can be included to encourage exploration and is calculated as follows:

$$S[\pi_\theta(a|s)] = -\sum \pi_\theta(a|s) \log \pi_\theta(a|s)$$

How much the entropy will affect the objective can be determined with the scaling factor c_2.

The theoretical background of DRL and its concepts has now been introduced. General aspects of states, actions, and rewards were first presented, followed by a more detailed walkthrough of selected DRL methods. Moving forward, the following section will present a search-based exact solving method.

2.5 Mixed-Integer Linear Programming and solver techniques

A Mixed Integer Linear Programming (MILP) problem is any mathematical model that can be formulated using the equations presented below [44]. The 3D-BPP is an example of such a problem, and hence any solver capable of approaching MILP problems can be used to solve it [6]. These solvers tend to be highly optimized, and contrary to the other methods presented earlier, a MILP-solver can prove an optimum solution if allowed to run forever. A solver like this aims to minimize an objective function (2.4), subject to integer and linear constraints (2.5) - (2.8).

$$\underset{x,t}{\text{minimize}}\ f(x,t) \tag{2.4}$$

$$\text{subject to } Ax + Bt \leq a \tag{2.5}$$

$$Cx + Dt = b \tag{2.6}$$

$$x \geq 0, \quad x \in \mathbb{R}^n \tag{2.7}$$

$$t_i \in \mathbb{Z}, \quad i \in \{1, \ldots, m\} \tag{2.8}$$

The minimization goal, $f(x,t)$ is a linear function, and in this case, the model should minimize the unused volume in the bins. The constraints will specify that it should pack all items within the bins and without overlap. There are multiple solvers capable of handling a MILP-problem formulation, such as Gurobi, SCIP, and CP-SAT [45, 46, 47]. These solvers combine different techniques to solve MILP problems, such as branch and bound, but some also incorporate techniques related to the Boolean satisfiability problem (SAT). The following two techniques will be explained.

2.5.1 Mixed-Integer Linear programming solvers

MILP solvers such as Gurobi or SCIP utilizes a branch and bound search algorithm to deal with the integer constraints. This algorithm will be described in the following section.

Branch and bound

Branch and bound is used when the solver does not know how to solve the problem directly [48]. Instead, the solver replaces all the integer constraints with relaxed real constraints; this is known as a Linear Programming (LP) relaxation. The solver then goes on to solve the new LP relaxed model, and when a solution is found, it checks if it fulfills the previously disregarded integer constraints. If the constraints are fulfilled, the solution is the optimal solution to the problem. Most iterations, they are not fulfilled, which the solver handles through a branching process that creates two new LP relaxed models based on the constraint that was not fulfilled by the previous solution. One of the new LP relaxations will have a constraint to be lower than or equal to the first integer below the returned fractional solution, and the

other will have the constraint to be larger than or equal to the first integer value above the returned solution. For example, if the first LP relaxation returned the integer variable $t = 4.5$, one of the new LP relaxations will get the constraint $t \leq 4$ and the other will get the constraint $t \geq 5$.

2.5.2 CP-SAT

CP-SAT is a Constraint Programming Boolean Insatiability solver. Although it's not a MILP solver it is capable of handling MILP problems. The strengths of CP-SAT lies within the combination of Constraint Programming (CP), SAT solving and use of LP relaxations.

Boolean satisfiability problem

Some solvers also incorporate techniques related to the SAT problem when tackling MILP problems. A set of Boolean expressions is satisfiable if there exists at least one feasible solution. Thus, a SAT solver simply aims to find any such solution [49]. For example, if given the expression $x \vee y$, the solver could find a feasible solution that might be $(x, y) =$ `(TRUE, FALSE)`. SAT solvers can be combined with other techniques, which allows for optimization instead of just finding a proven feasible solution.

Constraint Programming

A Constraint Programming solver takes a problem formulation consisting of a set of constraints, and like the SAT solver, it can search for a feasible solution that fulfills these constraints [50]. Unlike the SAT solvers, a CP solver can handle more types of constraints, not just Boolean; it can also handle integer and linear constraints. A CP solver can handle a constraint optimization problem that consists of a set of constraints and an objective function. The solver can then be used to find a feasible solution, find the solution that minimizes/maximizes the objective, prove the optimality of the solution, or prove that the problem is infeasible. It is therefore possible for CP to take on many different forms, but its central principle remains the same: The user specifies the constraints of a problem, and a general-purpose solver is used to find solutions that satisfy these constraints.

CP-SAT and Lazy Clause Generation

CP-SAT, included in Google's OR-Tools, is a Constraint programming saturated Boolean solver. This means that it combines the features from both a CP solver and a SAT solver [47]. CP-SAT utilizes branch and bound techniques, including LP relaxations, but also emphasizes Lazy Clause Generation (LCG), which is when a constraint on an integer variable is represented as two boolean expressions instead [51], this is done lazily, meaning that no clauses are generated until they are needed. For example, if the constraint on the integer variable x is:

$$x \leq a,\ x \in b, \ldots, c$$

it can be rewritten as the following constraints:

$$x = d,\ b \leq d \leq c$$

$$x \leq d,\ b \leq d \leq c$$

When these integer constraints are instead represented as two boolean constraints, it allows the solver to use SAT techniques to find a feasible solution. CP-SAT was built with parallelism in mind, making use of multi-threading, allowing for multiple workers to conduct parallel searches using different techniques for a better solution. Although this thesis have not been able to find an application of the CP-SAT solver to any kind of BPP it has shown to outperform traditional MILP solvers in other problems [52], which makes it interesting to apply it for the 3D-BPP. To summarize this section, a MILP is any type of problem that can be described through equations (2.4 - 2.8), and to solve these types of problems, there exist highly optimized solvers utilizing different methods to speed up the search.

2.6 Summary

In this chapter, a theoretical foundation has been laid, and techniques related to heuristics, GAs, DRL, and MILP-solvers have been presented. While this chapter has focused on describing the technical details, the next will focus on how these are applied to the 3D-BPP.

3
Survey

This chapter aims to present how the solvers described in the previous chapter are applied in practice throughout the literature. The purpose of this is to try and find what methods the solvers utilize for implementation, but also to review what solvers are suitable for what scenarios. While pure heuristic methods can be applied to the 3D-BPP, most modern heuristic approaches employ either search-based or learning-based optimization techniques on top of the heuristics used. Additionally, heuristic strategies such as EMSs have already been presented and will therefore not be repeated in this section.

3.1 Genetic Algorithm

One common method for solving the 3D-BPP is the use of GAs. To systematically compare the GA approaches, several key factors must be considered: The encoding scheme, the objective function, the applied heuristics, and overall algorithm performance. Other aspects, such as mutation, crossover, and selection strategies, could be explored further; however, these can be tested more flexibly once the core design decisions have been established. They can be seen as tuning parameters dependent on the principal design choices evaluated here.

GAs are often combined with a heuristic algorithm. While it is possible to apply GAs to the 3D-BPP without the use of heuristics, the vastness of the search space will lead to long convergence times. This inefficiency arises because GAs may require many generations, individuals, and restarts to find near-optimal solutions [26], especially in complex problems like 3D-BPP. Table 3.1 presents some of the heuristics used in combination with a GA but more options, such as layer-building and block-building, can also be used [7]. The construction of the chromosome and the components it includes are also evaluated. A common element is the bin packing sequence (BPS), which defines the order in which items are packed. This can be represented using floating-point numbers, where each value determines a sorted position in the sequence [53], or with integers that explicitly specify the packing order [6]. Another frequently included component is the container loading sequence (CLS), which indicates the assignment of items to bins. Lastly, many genetic algorithms incorporate a VBO, which tracks the rotation of each item. Whether this gene is included, and how it is represented, depends on what rotations are allowed. The final comparison involves evaluating computation time and performance across different studies. This can be challenging due to variations in methodology, result presen-

tation, datasets, and parameter settings. Nonetheless, a comparison is provided to offer a sense of what has been achieved in other approaches to the 3D-BPP.

Source	Objective	Heuristics
[8] (1)	Minimize waste volume	EMS prioritized by volume, margin from wall minimization
[8] (2)	Minimize waste volume	EMS prioritized by DBLF
[54]	Minimize number of bins	No usage of heuristics. Usage of NN and labeled data to learn feasible solutions.
[53]	Minimize number of bins	EMS prioritized by distance to top opposite corner
[55]	Maximize utilization rate	EMS prioritized by DBLF

Source	Encoding
[8] (1)	BPS, CLS
[8] (2)	BPS, CLS
[54]	Iterative binary vector representing if packed.
[53]	BPS, VBO
[55]	BPS, VBO

Source	Time	Performance
[8] (1)	Worst case: 107 s (78 items, 15 bins)	Waste space: 18-23% when items ≥ 20, Better than baseline algorithms, Found optimum when items ≤ 12
[8] (2)	Worst case: 60 s (78 items, 24 bins)	Waste space: 20-32% when items ≥ 20, Found optimum when items ≤ 12
[54]	Average computation time: 60 s (10-100 items)	Found optimum in 88-90% of cases. Improvement to benchmark algorithms.
[53]	Not specified	Average filling rate 92 % for perfect fit dataset
[55]	Worst case avg: 130 s (200 items)	Bin usage: Better than benchmark in 30/32 cases

Table 3.1: Comparison of various aspects of GA approaches found in the literature.

3.2 Deep Reinforcement Learning

This section aims to provide an objective evaluation of DRL-based methods and compare their performance, as summarized in Table 3.2.
One way to tackle the 3D-BPP using DRL is to combine it with a heuristic solution like the authors of [9], they used a heuristic solution for most of the problem and only

leaving the BPS up to the DRL, meaning that their heuristic solution chose what bin to use and also how the products should be rotated. The benefit of utilizing heuristics like this is that the state and action space becomes drastically smaller than if the DRL were to decide on everything, and the main drawback is that the limitations of the heuristic solver transfer to the DRL. Another way to handle the large action spaces can be to split the actions into sub-actions as done in [39], where they split the action of selecting a position within the bin into the three sub actions of selecting the position to place the item, select the item to be placed and lastly selecting the orientation for the item. The problem formulation of this work only included one bin to pack, i.e, there where no need for that action to include a selection of bins to place the products in. For the state representation, they split up the state into two parts, the state of the bin and the item state. The item state contained the information of the sizes of all different items to be packed, while the bin state was illustrated as a grid of the x-y-plane of the bin, where the cells within each grid described how high the products were packed there. This bin state aligned well with their optimization goal to maximize F_R by keeping the height of the bin low. They used an encoder-decoder architecture based on the original Transformer model introduced in [56]. There also exist DRL-solutions to the online 3D-BPP as in [57], where the state and action space are described by some basic rules of where the item can be placed. The Q-Network takes an input of possible future states derived from the basic rules and tries to predict what the optimal future state would be. Table 3.2 summarizes these solutions and gives a comparison between their results and the techniques used to achieve these results.

Source	**Objective**	**Optimization technique**
[9]	Minimize Surface area	Policy Optimization, Actor-Critic
[39]	Maximize F_R	PPO, Actor-Critic
[57]	Online F_R Maximization	DQL, Value method

Source	**Architecture**	**Heuristics**
[9]	Recurrent Neural Networks	SAM EMS
[39]	Transformer Encoder-Decoder	No Heuristic
[57]	Neural Network	Basic Rules

Source	**Time**	**Performance**
[9]	Not specified	2-3 Percentage points improvement from the Heuristic solution.
[39]	3.5s for 50 items	82.4% F_R for 50 items
[57]	34.2ms per item	82.8% Avg F_R

Table 3.2: Comparison of various aspects of DRL approaches found in the literature.

3.3 Mixed-Integer Linear Programming

There exist numerous implementations using a MILP solver. However, most of them suffer from excessively long runtimes due to the high computational complexity of 3D-BPP. To address this issue, MILP solvers are often combined with various techniques to reduce runtime. While these combinations and reformulations might not align perfectly with the optimization goal of this thesis, they show that it is possible to reduce the runtime of a MILP solver. The study in [6] employs a MILP solver to address the same optimization goals as this thesis. After observing excessively long runtimes, a GA is added to provide the solver with an initial solution, speeding up the process to some extent. An integer linear programming with a hybrid heuristic approach is introduced in [58]. That work deals with the traditional output minimization 3D-BPP, meaning that its objective is to minimize the number of identical bins. Their hybrid approach shows good results in terms of speed. In [59], the authors discuss the computational complexity of using a MILP for the one-dimensional BPP and replace the standard formulation with one containing equivalence classes. They also introduce a suboptimal formulation. These two contributions speed up the process for the one-dimensional BPP considerably. Table 3.3 presents a summary of various MILP-based solutions, outlining their objectives, any methods they were combined with, key limitations relevant to this thesis, and the solvers used. The solvers presented here aim to provide optimal solutions to the given problems, which they demonstrate the ability to do, thereby reducing the need for performance comparisons. Furthermore, since they are evaluated on different problem instances, direct performance comparisons are not entirely appropriate.

Source	**Objective**	**Combination approach**
[6]	Minimize waste volume	Added GA to provide initial solution
[58]	Minimize the amount of bins from a set of identical bins	Heuristics
[59]	Minimize waste volume in the 1D BPP	Equivalent classes and a suboptimal formulation

Source	**Limitation**	**Solver used**
[6]	Long runtimes	Gurobi
[58]	All bins are identical	CPLEX inspired methods.
[59]	Not directly transferable to 3D.	Gurobi

Table 3.3: Comparison of various aspects of MILP approaches found in the literature.

3.4 Comparison and summary

To conclude the initial literature review, a general overview is provided to highlight the strengths and weaknesses of different solver approaches. Understanding these trade-offs is also useful for remaining open to future decisions regarding which solvers to implement.

Method	Pros	Cons
Heuristics	- Fast and scales well. - Can provide intuitive, human-like solutions.	- No feedback or learning; performance remains static.
MILP	- Produces optimal solutions. - Works well for small instances. - No tuning required.	- Poor scalability.
GA	- Has the potential to scale well, depending on the heuristics and parameters used.	- Stochastic behavior - Convergence is not guaranteed - Tuning requires significant domain knowledge
DRL	- Fast and scales well. - Can continue to improve after implementation	- Requires network training. - Complex implementation. - Pre-learned strategies might not be sufficient to adapt to all new situations

Table 3.4: Comparison of different optimization approaches

It is also entirely possible to combine and adapt these solvers to better suit specific problem settings. This chapter has presented a brief survey of various methods for solving the 3D-BPP, laying the groundwork for informed decisions based on existing research. In the following chapters, the focus will shift toward describing and conducting the practical experiments.

4
Methods

This chapter outlines the methodologies employed in this thesis to address this 3D-BPP. First, the process of data generation used for training and evaluating the algorithms is described. Next, initial implementation details of the various algorithms are provided, and a description of how the experiments will be conducted. All components of the methodology are implemented in `Python`.

4.1 Datasets and Evaluation

Before outlining the methods for designing the solving algorithms, real data is analyzed to be able to create datasets for evaluation. The development of the algorithms was then tested and designed using these datasets, which directly affects their design. Three different categories of datasets will be used: A perfect fit dataset (PF), a generated real order dataset (GRO), and one based on real orders from a real warehouse (RO). The reason for including multiple datasets is that there are different aspects that each dataset can evaluate.

4.1.1 Warehouse Characteristics and Operational Challenges

Since this is supposed to be applied to a real-world problem, it is of interest to know how the real-world data is distributed, to be able to approximate it more closely. The daily orders of an e-commerce warehouse selling toiletries were analyzed for this, providing the results illustrated in Figure 4.1.

A total of 1594 different orders were analyzed, and it is possible to conclude that their orders are rather small. The major difficulty with solving the 3D-BPP is scalability because of its NP-hard complexity; the problem's difficulty drastically decreases with a product-order distribution looking like it does in Figure 4.1. Datasets with a higher number of products will be analyzed anyway to meet scalability and robustness requirements. Other customers may have a more variable number of products in their orders. Designing the solver to focus only on orders with fewer than 10 products, despite aligning with the data presented here, risks reducing flexibility and necessitating a redesign shortly. Regarding the number of bins, it varies from warehouse to warehouse between 4 to up to 277. Although the majority lies between 5 to 15. The data used to conclude this was gathered from anonymized customer

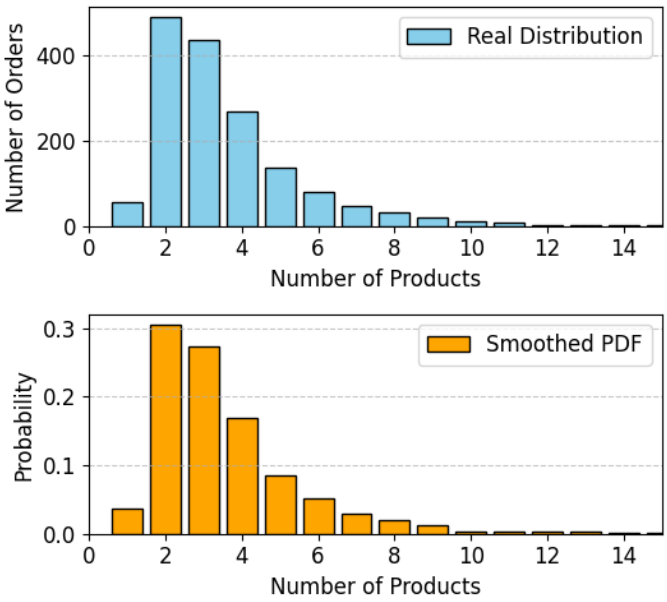


Figure 4.1: Histogram showing the distribution of products per order for a single warehouse.

records provided by Ongoing.

It is important to know that selecting the optimal set of bins to be available in a warehouse is an optimization problem in itself [60]. This will be known as the carton set selection problem from now on to distinguish it from the BPP. While this is outside the scope of this project, it remains an important consideration, as it can significantly impact the final results. The effectiveness of the bin-packing algorithm is ultimately constrained by the optimization of the carton set. For example, if a warehouse has only one packing bin with dimensions much larger than an item, then whenever that item is ordered individually, there will inevitably be a significant amount of unused space in the package.

4.1.2 Perfect Fit Dataset

This is the first presented dataset and is generated to ensure that all orders contain an optimal solution where it is possible to fill to 100%, which represents a scenario in which carton selection is ideal. Even though this scenario is not entirely realistic, it offers two major benefits. Firstly, the optimal solution is known; it serves as a reliable benchmark for development and testing. Secondly, evaluation becomes less dependent on the carton set selection, allowing for a more isolated ranking of bin-packing performance. The PF data generator algorithm is presented in Figure 4.2, and an example of a generated solution can be viewed in Figure 4.3.

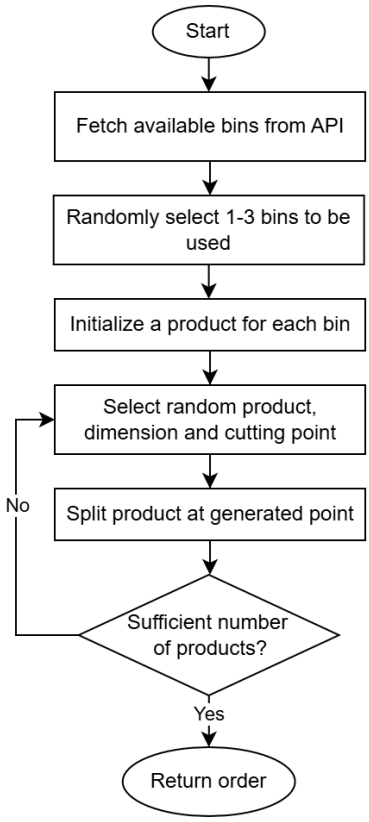


Figure 4.2: Perfect fit order generation.

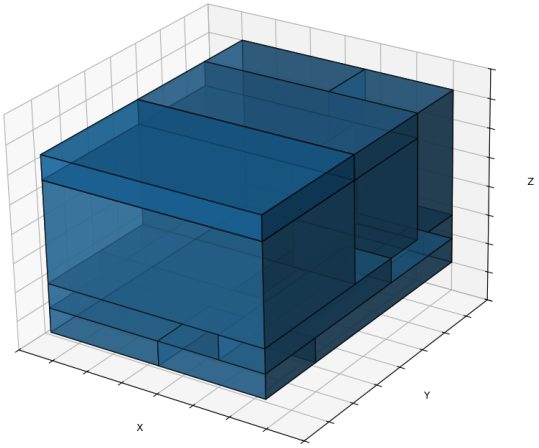


Figure 4.3: Perfect fit example.

The bins' dimensions are based on real packing boxes taken from real Swedish e-commerce packing boxes [61] and can be seen in Appendix A. A benefit of the generated data is that it can also create an almost infinite number of different orders, which is beneficial since it mitigates the risk of overfitting. To be able to compare different solvers, a seed was set for generation so that the same data was used. The number of products and bins was then randomly distributed as presented in the equations below. The reason for varying these numbers is that it is interesting to analyze how computational speed and filling rate depend on both n_p and n_b.

$$n_p \sim k \cdot \mathcal{U}_d(1, 20)$$
$$n_b \sim \mathcal{U}_d(5, 15)$$

The numbers chosen for the product distribution do not coincide with reality, but this dataset does not serve to show how it performs on the average case; it aims to test scalability and maximum performance when presented with an ideal set of bins. The scaling factor, k, aims to be able to provide later tests to conclude the algorithm's scalability for larger values of n_p.

Contrary to the products in an order, the set of bins available is within the control of the warehouse, and it is therefore not necessary to prove scalability concerning n_b. Although increasing the number of different bins indeed increases the difficulty of the optimization problem, the more concerning issue about including a high number of packing bins would more likely consist of the practical challenges. Firstly, maintaining a diverse set of bins requires a constant supply, which can be a logistical difficulty. Secondly, warehouses would need to store all these bins somewhere, and ensuring they are always available adds a layer of operational complexity. Thirdly, having a large number of bin options could increase the time and effort required for workers to locate and retrieve the correct bin. If bins are stored across different locations, workers may need to walk significant distances, reducing efficiency. Finally, expecting workers to recognize and correctly use many different bin types might not be feasible, potentially leading to errors and slower packing times. Because of these practical issues combined with its controllability, n_b only varies between common values.

4.1.3 Real orders

This dataset is the same as the one used in [6], and consists of three different subsets: small, medium, and large. Where the number of products, bins, and orders are presented below in Table 4.1. The benefit of utilizing this dataset is that it includes information about how a worker manages to fill it, and it includes real orders. The major drawbacks are that n_p does not follow the same distribution as observed in day-to-day operations, and the total number of orders is rather low (30). There is some stochasticity that might not be observed if evaluation were to only be done on these orders.

Dataset	n Products	n Bins	Orders
Small	3	28	10
Medium	10	28	10
Large	50	28	10

Table 4.1: Real order dataset.

4.1.4 Generated Real Orders

This dataset was created to better reflect real-world scenarios compared to earlier datasets. While the orders are artificially generated, both product and bin dimensions are based on real items and packing cartons. The number of products, n_p is estimated using the distribution in Figure 4.1. The number of bins, n_b was fixed at 15 as the warehouse lacked exact bin usage data, and this number is commonly found in similar operations and provides a sufficient range of combinations for testing. The same bin dimensions used in the PF datasets are used. A benefit of this dataset, similar to the perfect fit, is the ability to generate an almost infinite number of unique orders.

Although this dataset provides a setting that is well adapted to the specific warehouse where the orders and distribution of n_p are taken from, this configuration may not generalize to warehouses in other sectors. That said, this is not a critical limitation, as the dataset is primarily intended to test whether a solver can perform well in a specific, realistic setting. If good results are achieved, tests can later be diversified. A limitation of the dataset is that optimal solutions are unknown. This means it can only be used to compare relative performance between solvers if no optimum can be proven by the solvers themselves. Now that the three different datasets have been presented, it will be discussed how their results should be interpreted.

4.1.5 Evaluation Metrics

To evaluate the performance of the different solvers for the different datasets, the following two primary metrics will be used.

- **Computation time**: Comparing the computation time across solvers is crucial, but it presents several challenges. As will be shown in the following chapter, computation time is heavily influenced by n_p, and to a lesser extent by n_b. While average computation time provides some insight, the algorithm's computational complexity is often of greater interest. For small n_p, most solvers are capable of solving within seconds; however, computation time can grow rapidly as n_p increases. Attempts were made to fit exponential curves to the time data to get a measurement of the complexity, but these did not accurately reflect the trajectory as n_p increased. Furthermore, the computation time may not be consistent throughout, and it is also important to understand how it is distributed if any variance is present. As a result, computational complexity is decided to be interpreted visually as a function of time, rather than summarized by a single value, for cases where the results are not immediately clear.

- **Filling rate**: Increasing packing density is the primary objective, which is denoted by the sum of all products divided by the sum of all used bins.

$$F_R = \begin{cases} \left(\sum_{i=1}^{n} V(p_i)\right) \left(\sum_{i=j}^{m} V(b_j)\right)^{-1} & \text{if valid solution} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Worth stating that F_R is also heavily dependent on n_p, and in some cases n_b, making it infeasible to just examine an average. It is also vital to consider how F_R is disturbed to evaluate consistency and robustness. Likewise to the time constraints, a visual interpretation will be done to conclude which solver is the most suitable when analyzing scalability.

The solvers will then test these two evaluation metrics for the three different datasets. During this section, the PF, RO, and GRO datasets have been presented, each designed to evaluate different aspects of solver performance. The PF dataset is intended to test computational complexity and best-case performance. The RO dataset allows for comparison against current worker-based bin selection methods, while the GRO dataset assesses solver behavior under typical warehouse operations. Since performance depends on both n_b and n_p, and can vary significantly, it is infeasible to rely on a single performance metric for interpretation when analyzing scalability. These dependencies complicate the interpretation of results and make it difficult to summarize performance with a single number. Instead, a more visual and analytical evaluation approach must be adopted. In the following section, the implementation methods for the different solvers will be reviewed.

4.2 Heuristic

Heuristic algorithms were utilized for several reasons. Firstly, they offer excellent computational efficiency compared to most other solutions. Secondly, they serve as a strong comparative baseline. Finally, they can be integrated with other approaches, such as GAs and DRL, aiding in encoding simplification and reducing the state and/or action space. Hence, providing a good heuristic algorithm can also provide improvements while still utilizing meta-heuristics and machine learning. Methods based on both EPs and EMSs were chosen for implementation because of widespread usage and good performance.

4.2.1 Bottom deep left

The first implemented algorithm was the DBLF strategy based on EPs. However, to better align with the constraints of e-commerce warehouses, it was reordered to follow a Bottom-Deep-Left approach since a worker will fill up the box from the bottom up, but since the algorithmic differences are minimal, it will still be noted as DBLF going forward. The core idea of DBLF is to place items as close to the origin point of the bin, then update and sort the EPs by lowest height first, followed by width, and finally length. Popping the next EP then ensures that the desired EP is chosen for packing. The full loop of this algorithm is described by Algorithm 1.

The function `update_candidates` then adds the three extreme points, p_e, according to equations (2.1)-(2.3), sorted by the bottom-deep-left prioritization. If the distance between the bin wall and the p_e is smaller than all product dimensions, it is disregarded. To allow this to be packed in two bins, it is adjusted to first try to

Algorithm 1 Bottom Deep Left Fill Placement

Input: Products array P with sizes (l_i, w_i, h_i), Bin size (l_b, w_b, h_b).
Initialize candidate position to include origin $C_P \leftarrow (0, 0, 0)$
Initialize failed positions $F_P \leftarrow \emptyset$
for Product p in P **do**
    while C_P is not empty **do**
        $p_c \leftarrow$ Pop best candidate from C_P
        Origin of product $p_i \leftarrow p_c$
        if do_product_fit(prod) **then**
            add_product(prod)
            update_candidates(prod)
        else
            Append p_c to F_P if position inside bin
        end if
    end while
    for all $pos \in F_P$ **do**
        add_candidate(pos)
    end for
end for
return all_products_could_be_packed()

pack it in one bin until it is full or an item can not be placed, and then move on to the next one. This will cause the model to be guided to try to use one bin if it can, which is a conscious design choice since fewer bins used are preferred. The bin combinations are also sorted by volume ascending; the ones with a volume less than the total product volume are dismissed. This manner of always trying one bin until full will remain throughout the EMS-based implementations described in the following section.

4.2.2 Empty maximal spaces

It was then decided to proceed with a heuristic utilizing empty maximal spaces, similar to the approach described in [9], as it has already been demonstrated that the heuristic can be combined effectively with deep learning techniques, combined with the fact that EMSs are commonly found in the literature. The primary advantage of using EMS over EPs is that EMSs generate more placement positions, which provides greater potential since more options can be considered. EMS implementations can vary across different methods [62], and the following implementation introduces its variation.

In this approach, EMSs are represented by tracking the coordinates of the bottom-deep-left corner and the upper-right corner. After placing the product within an EMS, the space is removed, and three new spaces are generated along each dimension. If a product *intrudes* on another EMS for which it was not designated, the intruded one splits to accommodate the item. This results in the EMS splitting

up to six new ones to fit both high and low-axis values along all three dimensions. Generally, most can be disregarded early on because they have dimensions equal to zero or other invalid behavior. This behavior is illustrated in Figure 4.4, where an item is placed in the center of an empty bin to visualize the resulting EMS splits. If any EMS cannot fit any item based on volume and dimensions, it is deemed invalid and is hence removed [53]; EMSs that are subsections of others are also omitted. This EMS generation process was inspired by the difference process described in [63]. It is worth noting that the computational complexity can become quite high when using EMSs, as their number may increase rapidly with each placed item. The number of EPs also increases with each item placement, but at a slower pace. However, EMSs offer the advantage of not requiring collision checks, as each EMS is guaranteed to represent an unoccupied space. In summary, both EPs and EMSs have their own computational drawbacks, and to determine which approach is more suitable, they will be compared through testing.

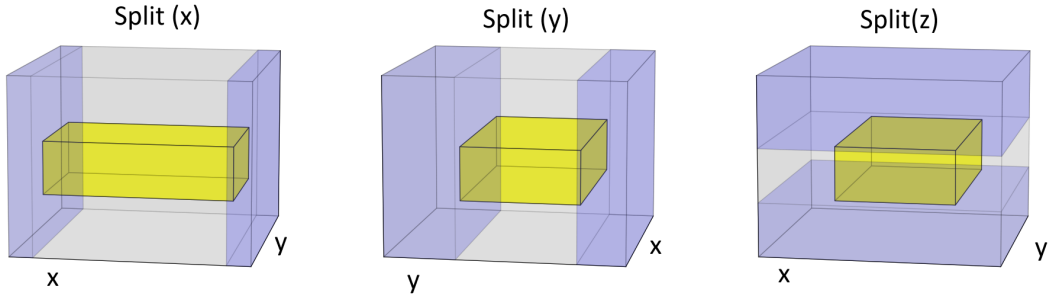


Figure 4.4: Six newly generated EMSs in blue after the initial EMS, filling out the bin gets intruded by the yellow item.

The full EMS-based algorithm iteration then works by placing a product p with rotation r in the EMS origin, then testing all EMS and rotation combinations, and finally deciding on the pair that yields the best results. The best result is determined by an objective function, ϕ, and this project suggests these three alternatives for deciding which EMS is the best.

- Smallest fit (SF)
- Surface area minimization (SAM)
- Bounding Box minimization (BB)
- DBLF

Selecting the smallest EMS has the benefit that it can terminate early and not have to test every rotation, since if the EMSs are sorted by size, the first time an item can fit, it is in fact inside the smallest EMS. This will decrease computation time, but it does not generally push the system to fill in a DBL manner since it only focuses on filling the smallest gaps. This can potentially lead to the solver sometimes including large gaps in the middle of the bin, surrounded by already placed products.

Algorithm 2 Update EMSs after placement of product

```
1: Input: Designated EMS object des_ems containing origin and end coordinates.
2: emss ← emss \ des_ems
3: new_emss ← Create new EMSs at the right, back, and top of the product
4: for all EMS in new_emss do
5:     if EMS is invalid (doesn't fit the product or is too small) then
6:         new_emss ← new_emss \ EMS
7:     end if
8: end for
9: splitted_emss, intruded_emss ← SplitIntrudedEMSs(emss)
10: emss ← emss \ intruded_emss
11: new_emss ← new_emss ∪ splitted_emss
12: for all ems in new_emss do
13:     for all ems_ in new_emss ∪ emss do
14:         if ems ⊂ ems_ then
15:             new_emss ← new_emss \ ems
16:             break
17:         end if
18:     end for
19: end for
20: emss ← emss ∪ new_emss
21: Sort emss in ascending order of volume
22: return emss
```

The second proposed objective function is SAM, as used in [9]. Area minimization encourages tighter packing of products and can therefore also be a beneficial method. Compared to SF, it does not have the same risks of leaving gaps in the middle of the bin since it benefits mostly from packing next to an already placed item. Compared to packing a product against a wall, hiding a surface against another product will half the visible area since two faces are now hidden. The third idea is to try and minimize the volume of a bounding box that can contain all the placed items. The fourth and final idea is to place them in a DBLF manner, prioritized by the EMS origin coordinates.

The products are sorted from largest to smallest since it is more probable to fit the larger items early on, but this sorting might not always be the optimal one. It would be desirable to test multiple ordering methods, but since the number of orders possible to arrange the products for larger numbers of items grows in a factorial manner, $O(n!)$, it is simply not suitable to try them all in the same exhaustive manner that EMSs and rotations are tried. The full EMS algorithm can be viewed below.

4.2.3 Box building

To reduce computational load, it was decided to also implement an algorithm that took products that had equal faces and merged these into one product. This was

Algorithm 3 Bin Packing with Empty Maximal Spaces

Input: Array P, with product dimensions (l_i, w_i, h_i). Set B with dimension (l_j, w_j, h_j).
Augment B with a null variable and define a combination set C_B that includes every possible combination of two bins from B_{aug}.
Sort C_B by ascending volume.
Sort P by descending volume.
for c_B in C_B **do** ▷ Iterate over bin combinations
    for p in P **do**
        if Volume(c_B) < Volume(P) **then**
            Continue
        end if
        Initialize the objective function optimum ϕ^* as the worst case possible.
        for b in c_B **do**
            for each EMS $es \in ES$ and rotation $r \in R$ **do**
                Place product with rotation r in (es, b).
                if $\phi(p, es, r, b) > \phi^*$ **then** ▷ Objective function
                    $(r^*, es^*) \leftarrow (r, es)$
                end if
                Remove p from b.
            end for
            if Could be placed **then**
                Place p according to (r^*, es^*)
                Jump to next p
                UpdateEMS(p, es^*)
            else
                Continue
            end if
        end for
    end for
    if All products could be placed **then** Return Successful
    end if
end forreturn Failure

done recursively until all products that could be merged had been merged. Identical products got merged in a manner so that their largest surfaces were placed against each other. Building entities can be beneficial in warehouses where there are a lot of similarly sized products. This is done before placement algorithms do the job of actually deciding the position and orientation of the items.

During this section, the heuristic algorithms that will be reviewed have been presented. One that utilizes EPs and three different variations utilizing EMSs. It will be experimented whether it can be useful to include box-building before the placement heuristic is utilized. The design choices have been made according to what is commonly found in the 3D-BPP literature. Both the GA and DRL presented in the

following sections will utilize some version of the heuristics presented here.

4.3 Genetic Algorithm

A GA was chosen for implementation due to its widespread use in the literature and its demonstrated effectiveness in solving the 3D-BPP. Metaheuristics such as GAs can provide scalable approaches and have the potential to improve on already existing heuristic methods. This section outlines the initial design choices and presents the methodology used for implementation and refinement.

4.3.1 Encoding

The encoding of the GA was inspired by [6], but instead of representing the bin usage through a bin index gene-set, this thesis uses a Bin Combination Index (BCI). After observing stochastic issues related to this gene, it was explored whether bin selection could be outsourced using the same trial-and-error approach applied in the heuristic. Additionally, it was investigated whether restricting the GA to operate solely on item order could be beneficial, similar to the approach in [9]. This exploration resulted in four different chromosome designs with varying degrees of freedom (DoF), representing different levels of reliance on the GA versus the heuristic, where a higher DoF represents more GA responsibility.

$$
\begin{aligned}
\text{1-DoF GA Chromosome:} \quad & \underbrace{\{0.1, 0.3, 0.2\}}_{\text{BPS}} \\
\text{2-DoF VBO GA Chromosome:} \quad & \underbrace{\{0.1, 0.3, 0.2\}}_{\text{BPS}}, \underbrace{\{5, 1, 0\}}_{\text{VBO}} \\
\text{2-DoF BCI GA Chromosome:} \quad & \underbrace{\{0.1, 0.3, 0.2\}}_{\text{BPS}}, \underbrace{\{6\}}_{\text{BCI}} \\
\text{3-DoF GA Chromosome:} \quad & \underbrace{\{0.1, 0.3, 0.2\}}_{\text{BPS}}, \underbrace{\{5, 1, 0\}}_{\text{VBO}}, \underbrace{\{6\}}_{\text{BCI}} \\
& \underbrace{\{0.1, 0.3, 0.2\}}_{\text{BPS}}, \underbrace{\{6\}}_{\text{BCI}}, \underbrace{\{5, 1, 0\}}_{\text{VBO}}
\end{aligned}
$$

The size of the order gene and the rotation gene equals n_p. The order gene includes floating point numbers, and the sorted indexes in ascending order represent what order the bins should be filled. In this case, the first item has the lowest order gene and should hence be placed first, followed by the third item. The rotation gene refers to a rotation index where the number at position i ranges from 0 to 5 and represents one of the six possible rotations the item can have. Lastly, the bin gene represents which bin combination is being used and ranges from 0, the smallest possible, up to the number of possible combinations.

4.3.2 Further Initial Design Choices

Further decisions related to the GA are presented below.

- **Packing algorithm:** To pack the items, the DBLF-EMS heuristic, not utilizing box-building, is used since it has the best time complexity out of the tested ones. This will be shown in the following chapter. To calculate the fitness for all epochs and individuals, the heuristic will have to be evaluated once for every individual for every generation ($N \times G$), and hence the heuristic computation time is of utmost importance.

- **Fitness evaluation:** After the items had been packed, it was time to evaluate how fit the chromosome was, which was set to equal the filling rate each individual could achieve.

- **Crossover:** Single-point crossover was chosen after initial testing, where it was found that it had a higher convergence rate compared to multi-point crossover.

- **Selection:** Tournament selection was used, as it is widely adopted in the 3D-BPP literature and balances selection pressure well. The tournament is tuned by two parameters, the tournament selection probability p_{tour} and the number of tournament participants k_{tour}.

- **Elitism:** The GA included elitism to help stabilize convergence by preserving the best-performing individuals across generations. The number of elitist selections that are directly included in the next epoch is denoted by TOP.

- **Biased Initialization:** Some chromosomes were initialized with a bias to mimic heuristic behavior, given that heuristic-based solutions often yield good results. Examples include pre-sorting the order gene by descending volume or fixing all rotation values. It is not recommended to include bias in a GA [26], but in this way, it is possible to allow the model to outperform pure heuristic methods more consistently. The number of biased individuals was treated as a tunable parameter (N_{bias}), and the specific type of bias applied was chosen at random.

- **Mutated Elites:** Some elite individuals were selectively mutated to explore the neighborhood of top-performing solutions, providing local exploration. The number of elitist mutants is denoted by TOP_{mut}.

- **Mutation**: The mutation rate is denoted by p_{mut}. For integer genes, each gene has a mutation probability of b_g/N_i, where N_i is the number of integers in that gene and b_g represents a scaling factor used to balance this individual set of genes' mutation rate. The order gene, a floating-point number, is mutated by adding a value sampled from a normal distribution $\mathcal{N}(0, \sigma_{mut})$ to its current value.

4.3.3 Experiment Steps

After these general design choices had been made, the algorithm was evaluated through a series of steps. First, a set of tuning parameters that showed reasonable performance in preliminary tests was carried forward. Using these fixed parameters, the different number of parameters, and given the relatively long runtime of each GA instance, chromosome configurations were tested against each other. The best-performing chromosome was then tuned more carefully using an iterative grid search. Ideally, each GA variant would be individually tuned for a fairer comparison. However, this was not feasible due to computational constraints. The computational complexity of grid search grows exponentially, with performing exhaustive parameter tuning for every chromosome configuration was impractical. The initial parameters were inspired by the parameters in [6], and then improved further by iterative testing. The parameters can be viewed in Table 4.2. It was also tested whether multiprocessing could be used to calculate the fitness scores for the GAs, which can be done independently without the need for scheduling.

Notation	Name	Value
G	Generations	10
N	Population size	100
Top	Number of Elites	1
p_{mut}	Mutation rate	0.4
k_{tour}	Tournament size	3
p_{tour}	Tournament probability	0.9
N_{bias}	Biased initiations	12
σ_{mut}	Mutation deviation	1
(b_{BPS}, b_{BCI}, b_{VBO})	Mutation rate biases	(1, 1, 1)

Table 4.2: Default parameter values for the genetic algorithm used in experiments.

After the encoding has been decided on, the grid search tuning will take place. The order in which this is done is by taking the most interconnected parameters and running them together or individually for 3 to 5 different settings. The best result is then selected, and a local search around that pair of best-performing parameters is performed. This is done one or two iterations, depending on the ambiguity of the results. The order in which the parameters are tuned is the following:

- Tournament parameters. (p_{tour}, k_{tour})
- Mutation parameters. (p_{mut}, σ_{mut})
- Mutation rate bias. (b_{BPS}, b_{BCI})
- Elitist selections. $(\text{Top}, \text{Top}_{mut})$
- Biased Initiations. N_{bias}

During this section, the initial GA design choices have been described, followed by a discussion on how it is planned to evaluate and improve them. As a result of the evaluations and tests described in this section, the final GA design is presented after the evaluations done in Section 5.3.

4.4 Deep Reinforcement Learning

Since DRL is commonly found within the literature, one was implemented for this specific problem. Multiple different environments and architectures were tested. A DQN was implemented without showing any tendencies to learn. Multiple PPO encoders were tested, such as Transformers, Pointer networks, CNN autoencoders, and standard fully connected networks. Experiments were made with different environments, where some environments relied more heavily on the heuristics and some relied more heavily on the DRL algorithm. This thesis found the most success with a DRL algorithm using PPO to optimize and learn, while relying heavily on a heuristic solution like done in [9]. The DRL algorithm presented here, and later evaluated in the following chapter, has shown the most promise. However, it is important to note that several other DRL algorithms were initially tested and subsequently discarded, highlighting the challenges and uncertainties that have been encountered during the implementation of DRL approaches. This algorithm will be explained further throughout this section by first describing the environment, followed by a description of the network architecture, and finally, describing the training procedure.

4.4.1 Environment

The environment was implemented using `Gymnasium.env` as a starting point [64]. This provided a shell structure to build the custom environment in. The observation space was defined to consist of all the products' and bins' dimensions. The agent then arranges a BPS, which shows what order the products are placed in. The BPS was implemented as a list of floats, and the length was equal to n_p, where the higher values were to be packed first and the lower values to be packed last. The actions then become to increase or decrease a value in the BPS to change the order, or to stop when the agent no longer believes any improvements can be made. After every action the heuristic DBLF EMS packs the order and returns the F_R to the environment. The reward structure was taken from [39] to give the agent continuous reward updates during the episode, where the rewards are based on the improvement made, which can be defined by the following equations.

$$g_t = \sum_{j=1}^{m} V(b_j) - \sum_{i=1}^{n} V(p_i)$$

$$r = g_{t-1} - g_t$$

This reward structure allows for rewards at every time step, which is based on the improvement made by the new packing order, allowing for instant feedback after every action taken. The observation space where padded with zeroes to allow the network to have input of equal size.

4.4.2 Networks

The actor-network and the critic-network shared the layers and only used a different final layer, where the actor's final layer was a fully connected layer shaped as `64 -> n_actions`. Applying a softmax activation to this output gives the probabilities for taking each action. The critic's final layer was a fully connected layer of the shape `64 -> 1`, resulting in the predicted value for the current policy. The shared layers between the network were two fully connected layers consisting of 64 neurons using the tanH activation function. The shared architecture is shown in Figure 4.5. Where the blue layer is specific for the policy network, whilst the red layer is specific to the value network.

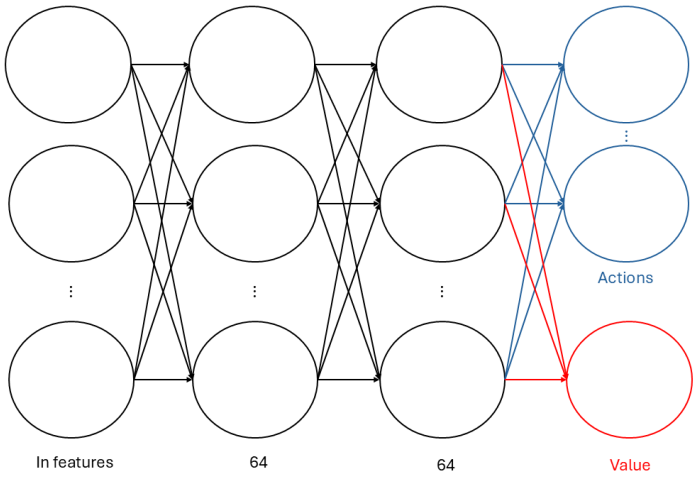


Figure 4.5: The network architecture.

4.4.3 Training

The networks where trained until convergence on 10 000 orders generated from the PF dataset using PPO which took about 600 000 timesteps. The parameters used during training can be seen in Table 4.3. It was tested to decrease α as the training progressed, but once it had converged, it did not provide much benefit.

Parameter	Value
α	$3 \cdot 10^{-4}$
c_1	0.5
c_2	0.01
γ	0.99
ϵ	0.2
λ	0.95

Table 4.3: Training parameters.

This section described the implementation of the DRL algorithm used for this project. Starting with the environment, and also describing the network architec-

ture used for the agent. Lastly, the training procedure was described. The following section will show the implementation of the MILP problem formulation.

4.5 Mixed-Integer Linear Programming Formulation

MILP-solvers are commonly found within the literature. Therefore, this thesis also formulated a MILP model which could be solved using solvers like SCIP or Gurobi, but also with the CP-SAT solver, since it has been shown that it can outperform traditional MILP-solvers in some cases. This section will present the MILP problem formulation as well as the implementation of the SCIP and CP-SAT solvers. Gurobi is disregarded because of its license cost.

4.5.1 MILP Formulation

The MILP formulation used in this thesis is based on the one used in [6]. All the variables, parameters, and constraints used are listed below:

Variables and parameters

Table 4.4 shows variables and parameters used in the MILP model with their descriptions.

Notation	Description
n_p	Number of products to be packed
n_b	Number of bins available
p_i, q_i, r_i	Length, width, height of product i
L_j, W_j, H_j	Length, width, height of bin j
$x_i \geq 0$	Continuous variable indicating the x-position of product i's origin
$y_i \geq 0$	Continuous variable indicating the y-position of product i's origin
$z_i \geq 0$	Continuous variable indicating the z-position of product i's origin
M	Arbitrary large integer constant used to disable constraints
β	Integer constant limiting the number of bins used for packing

Table 4.4: Variables and parameters with their descriptions.

State variables

The following variables are boolean status descriptions defined as binary integer variables, to describe the different states the products and bins can have. The variable s_{ij} presented below keeps track of where the products are placed, and v_j tracks which boxes are used.

$$s_{ij} = \begin{cases} 1 \text{ if product } i \text{ is placed in box } j, \\ 0 \text{ otherwise.} \end{cases}$$

$$v_j = \begin{cases} 1 \text{ if box } j \text{ is used} \\ 0 \text{ otherwise} \end{cases}$$

Where $i \in \{1, \ldots, n_p\}$ and $j \in \{1, \ldots, n_b\}$. The following three variables are included to keep track of orientation for all three dimensions, with $a \in \{x, y, z\}$.

$$l_{ai} = \begin{cases} 1 \text{ if the length of product } i \text{ is parallel to the } a\text{-axis} \\ 0 \text{ otherwise} \end{cases}$$

$$w_{ai} = \begin{cases} 1 \text{ if the width of product } i \text{ is parallel to the } a\text{-axis} \\ 0 \text{ otherwise} \end{cases}$$

$$h_{ai} = \begin{cases} 1 \text{ if the height of product } i \text{ is parallel to the } a\text{-axis} \\ 0 \text{ otherwise} \end{cases}$$

The following 6 variables are included to keep track of where the products are placed in the bins with respect to one another, with $i, k \in \{1, \ldots, n_p\}$ and $i < k$.

$$a_{ik} = \begin{cases} 1 \text{ if product } i \text{ is placed on the left side of item } k \\ 0 \text{ otherwise} \end{cases}$$

$$b_{ik} = \begin{cases} 1 \text{ if product } i \text{ is placed on the right side of item } k \\ 0 \text{ otherwise} \end{cases}$$

$$c_{ik} = \begin{cases} 1 \text{ if product } i \text{ is placed behind product } k \\ 0 \text{ otherwise} \end{cases}$$

$$d_{ik} = \begin{cases} 1 \text{ if product } i \text{ is placed in front of product } k \\ 0 \text{ otherwise} \end{cases}$$

$$e_{ik} = \begin{cases} 1 \text{ if product } i \text{ is placed below product } k \\ 0 \text{ otherwise} \end{cases}$$

$$f_{ik} = \begin{cases} 1 \text{ if product } i \text{ is placed above product } k \\ 0 \text{ otherwise} \end{cases}$$

Objective and constraints

With the variables described, the full MILP model is defined through the following objective and constraints. Explanations of all the equations' significance are found on the next page.

$$\underset{\substack{x_i,\, y_i,\, z_i,\, l_{ai},\, w_{ai},\, h_{ai},\, s_{ij},\\ a_{ik},\, b_{ik},\, c_{ik},\, d_{ik},\, e_{ik},\, f_{ik},\, v_j}}{\text{minimize}} \quad \sum_{j=1}^{n_b} v_j \cdot L_j \cdot W_j \cdot H_j \tag{4.2}$$

subject to

$$x_i + p_i \cdot l_{xi} + q_i \cdot w_{xi} + r_i \cdot h_{xi} \leq x_k + (1 - a_{ik}) \cdot M \quad \forall i, k \in \{1, \ldots, n_p\},\ i < k, \tag{4.3}$$

$$x_k + p_k \cdot l_{xk} + q_k \cdot w_{xk} + r_k \cdot h_{xk} \leq x_i + (1 - b_{ik}) \cdot M \quad \forall i, k \in \{1, \ldots, n_p\},\ i < k, \tag{4.4}$$

$$y_i + p_i \cdot l_{yi} + q_i \cdot w_{yi} + r_i \cdot h_{yi} \leq y_k + (1 - c_{ik}) \cdot M \quad \forall i, k \in \{1, \ldots, n_p\},\ i < k, \tag{4.5}$$

$$y_k + p_k \cdot l_{yk} + q_k \cdot w_{yk} + r_k \cdot h_{yk} \leq y_i + (1 - d_{ik}) \cdot M \quad \forall i, k \in \{1, \ldots, n_p\},\ i < k, \tag{4.6}$$

$$z_i + p_i \cdot l_{zi} + q_i \cdot w_{zi} + r_i \cdot h_{zi} \leq z_k + (1 - e_{ik}) \cdot M \quad \forall i, k \in \{1, \ldots, n_p\},\ i < k, \tag{4.7}$$

$$z_k + p_k \cdot l_{zk} + q_k \cdot w_{zk} + r_k \cdot h_{zk} \leq z_i + (1 - f_{ik}) \cdot M \quad \forall i, k \in \{1, \ldots, n_p\},\ i < k, \tag{4.8}$$

$$a_{ik} + b_{ik} + c_{ik} + d_{ik} + e_{ik} + f_{ik} \geq s_{ij} + s_{kj} - 1 \quad \forall i, k, j,\ i < k, \tag{4.9}$$

$$\sum_{j=1}^{n_b} s_{ij} = 1 \quad \forall i \in \{1, \ldots, n_p\}, \tag{4.10}$$

$$\sum_{i=1}^{n_p} s_{ij} \leq M \cdot v_j \quad \forall j \in \{1, \ldots, n_b\}, \tag{4.11}$$

$$x_i + p_i \cdot l_{xi} + q_i \cdot w_{xi} + r_i \cdot h_{xi} \leq L_j + (1 - s_{ij}) \cdot M \quad \forall i \in \{1, \ldots n_p\},\ \forall j \in \{1, \ldots, n_b\}, \tag{4.12}$$

$$y_i + q_i \cdot w_{yi} + p_i \cdot l_{yi} + r_i \cdot h_{yi} \leq W_j + (1 - s_{ij}) \cdot M \quad \forall i \in \{1, \ldots n_p\},\ \forall j \in \{1, \ldots, n_b\}, \tag{4.13}$$

$$z_i + r_i \cdot h_{zi} + q_i \cdot w_{zi} + p_i \cdot l_{zi} \leq H_j + (1 - s_{ij}) \cdot M \quad \forall i \in \{1, \ldots n_p\},\ \forall j \in \{1, \ldots, n_b\}, \tag{4.14}$$

$$l_{xi} + l_{yi} + l_{zi} = 1 \quad \forall i \in \{1, \ldots, n_p\}, \tag{4.15}$$

$$w_{xi} + w_{yi} + w_{zi} = 1 \quad \forall i \in \{1, \ldots, n_p\}, \tag{4.16}$$

$$h_{xi} + h_{yi} + h_{zi} = 1 \quad \forall i \in \{1, \ldots, n_p\}, \tag{4.17}$$

$$l_{xi} + w_{xi} + h_{xi} = 1 \quad \forall i \in \{1, \ldots, n_p\}, \tag{4.18}$$

$$l_{yi} + w_{yi} + h_{yi} = 1 \quad \forall i \in \{1, \ldots, n_p\}, \tag{4.19}$$

$$l_{zi} + w_{zi} + h_{zi} = 1 \quad \forall i \in \{1, \ldots, n_p\}, \tag{4.20}$$

$$\sum_{j=1}^{n_b} v_j \leq \beta \tag{4.21}$$

Explanations

- Equation (4.2) is the objective function for what is to be minimized, in this case it's the total volume of the used bins.

- The constraints (4.3)-(4.8) make sure none of the products packed overlap with each other. They are only active under specific conditions, as example, equation (4.3) is only active if $a_{ik} = 1$, which is if product k is on the left side of product k

- Constraint (4.9) ensures that if two products are placed in the same bin one of the relations a, b, c, d, e or f needs to be active. This makes sure that they are not placed at the same spot or overlap with each other.

- Constraint (4.10) makes sure that a product has to be placed in a bin but can only be placed in one bin, whilst (4.11) sees to it that a bin containing a product is considered used.

- Constraint (4.21) is an optional constraint that can be activated to set a maximum of allowed bins to be used when packing an order. This allows the user to specify the problem according to their preferences.

- The bins' capacity is not to be exceeded thanks to equations (4.12 - 4.14) which keep the items within the boundaries of the bins, they are only active under the conditions that product i is placed in bin j, i.e, $s_{ij} = 1$.

- Constraints (4.15) - (4.20) makes sure that a product is oriented in one of the six possible rotations allowed.

These constraints were then implemented using the linear solver submodule from `ortools`. Allowing for the use of both the CP-SAT and SCIP solvers. These solvers can also be used with a time limit, stopping them from going on for hours and instead returning the best solution found until the time limit is reached. Since CP-SAT is developed with parallelism in mind, it was allowed to use 12 cores when running.

4.6 Summary

This chapter has presented the datasets, evaluation criteria used throughout the study, and implementation approaches for various solvers. The solvers to be evaluated include Heuristics, GA, DRL, SCIP and CP-SAT. Additionally, the initial design choices and methods for future development have been outlined. The following chapter will evaluate these different methods.

5
Experiments and Results

This chapter presents the results of the methods described in Chapter 4 and also provides brief interpretations to give the results a bit more depth. It presents experiments done during the development of the algorithms to highlight why certain decisions have been made. At the end of this chapter, a final evaluation of the fully developed algorithms will be conducted.

5.1 Hardware

All experiments were conducted on laptops running 64-bit Windows 11 with 32-GB of RAM. Each device was equipped with one of the following processors:

- Intel® Core™ Ultra 7 165U @ 2.10 GHz
- Intel® Core™ i7-1370P @ 1.90 GHz

Worth noting is that all comparisons that lays foundation to a design decision is ran on the same processor.

5.2 Heuristics

After the heuristic algorithms had been designed, they were tested to be able to conclude which was suitable for further use. This evaluation was conducted using the PF data set with $k = 1$. Since the heuristic methods work rather fast, the number of orders (n_o) was set to 10 000. Which resulted in the following results.

Objective function	Mean FR [%]	Mean time [ms]	Invalid Solutions [%]
Bounding box EMS	74.7	28.8	5.0
SAM EMS	74.6	25.2	5.9
DBLF EMS	73.8	5.3	6.1
SF EMS	73.8	7.2	6.0
EP	72.7	13.9	6.5

Table 5.1: Comparison between different datasets for heuristic algorithm.

After this had been implemented, they were retested with a block-building heuristic that merged products that had equal faces, which gave the results shown in Table 5.2.

Objective function	Mean FR [%]	Mean time [ms]	Invalid Solutions [%]
SAM Block	85.5	4.0	5.0
Bounding Box Block	85.4	3.5	5.0
SF Block	85.3	2.0	5.0
DBLF Block	85.3	2.0	5.0
EP Block	85.1	2.9	5.1

Table 5.2: Comparison between different datasets for block-based heuristic algorithm.

As observed, the block-building demonstrates significantly improved performance. However, in real-world scenarios, assuming that every item can form groups with all others is unrealistic, even though it is possible with PF. Therefore, despite its strong performance, continued evaluations will be done both with and without block-building.

To evaluate scaling capabilities, the value of k is increased to 5. For this test, SF and DBLF are selected due to their speed. Multiple methods were tested using block-building, but because of identical behavior, only the DBLF-block is visualized, while BB is included for its slightly better performance compared to the other methods. The results are presented below in Figure 5.1.
As shown in Figure 5.1, the performance plots appear to converge when block-building is not used, showing no downward trend. However, when block-building is applied, performance drops rapidly, suggesting that it is not feasible to use block-building for $n_p > 30$ on the PF dataset. The PF dataset is unique in the context of block-building, as it allows the creation of a single merged object before optimization begins. As n_p increases, the risk grows that this merged object may be shaped in a way that does not fit well within the bin. Therefore, it is still unclear whether block-building is beneficial in real warehouse scenarios. To determine this, further experimentation with GRO is necessary.

Among the three solvers not utilizing block-building, DBLF emerges as the best-performing heuristic, which indicates great scalability. It achieves the highest F_R, tied with BB, while also demonstrating the lowest computation time. Further testing using heuristics will be done using the DBLF-EMS heuristic, both with and without block-building. Moreover, the heuristic methods presented here are relatively insensitive to n_b in terms of computational complexity. Filling rate can, nevertheless, significantly increase when including more bins. To get a clear sense of the performance variability, Figure 5.2 shows box plots regarding the DBLF heuristic.
The variability in performance is rather high, which is sub-optimal. Although the median filling rate is stable and rather high for this number of products, the lower whisker stretches down to 0 for almost all instances. This indicates that it is not very robust. The time plot shows good results, indicating that it is far below one second for n_p stretching up to 100. This shows that the heuristics implemented have good scalability, but lack performance consistency. The next test uses the GRO dataset to evaluate whether block-building is useful in practice. The results can be seen

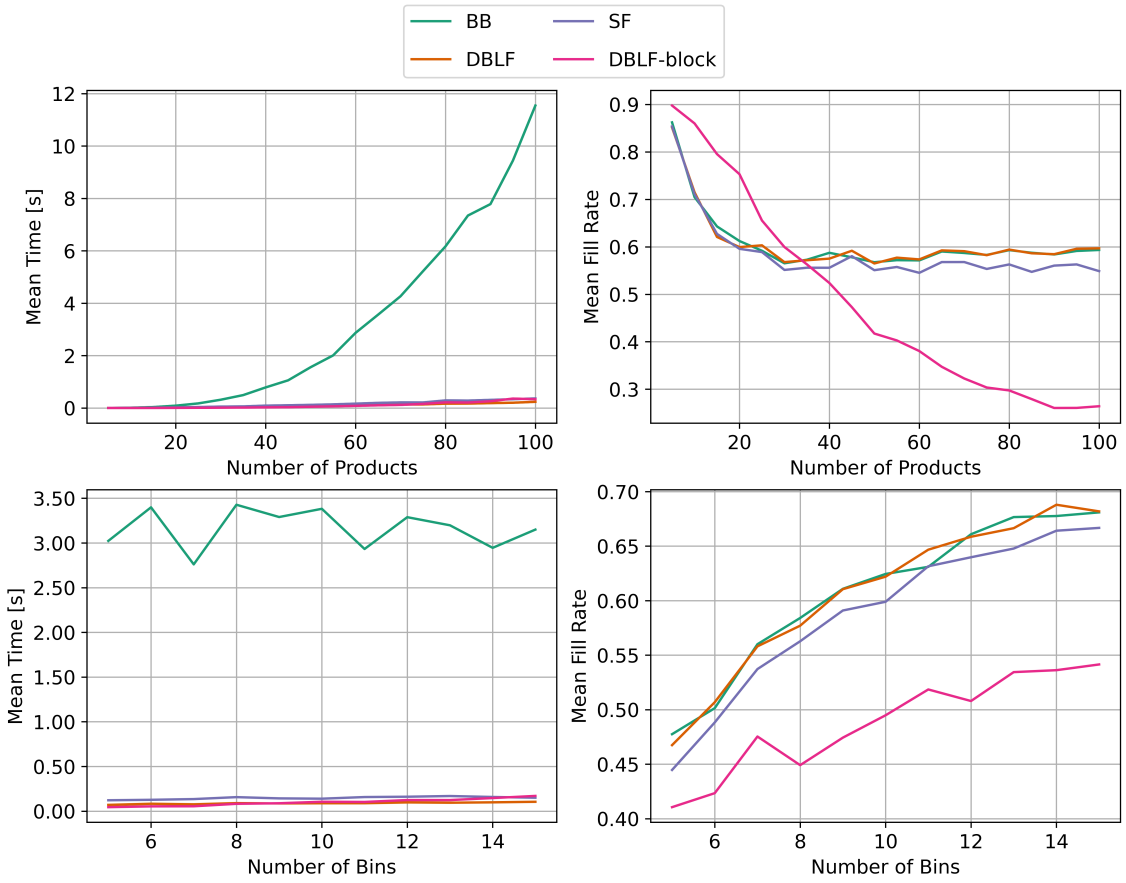


Figure 5.1: Heuristic algorithm evaluation.

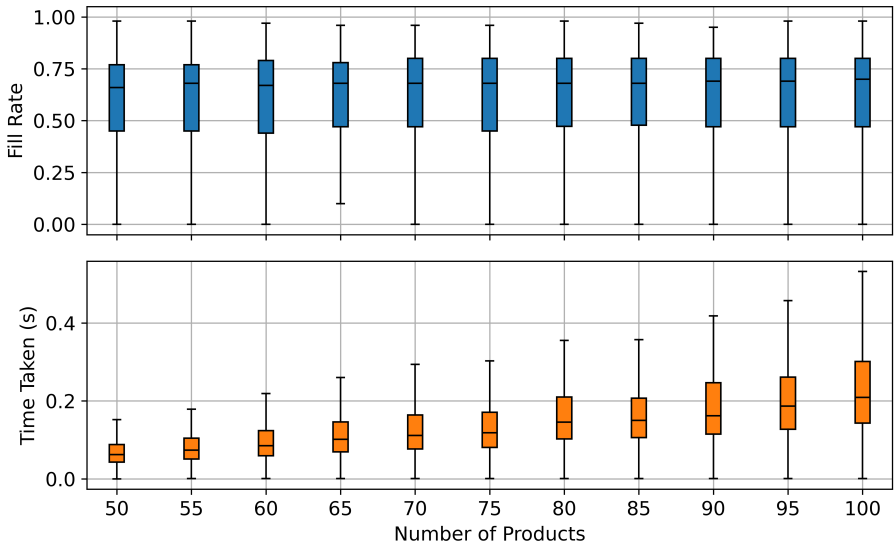


Figure 5.2: Box plot distribution DBLF-EMS solution not utilizing block-building.

below.

As observed, the overhead introduced by searching for products with equal areas and

Solver	Mean solving time [ms]	Mean F_R
DBLF EMS	1.38	36.7
DBLF EMS With Block-building	14.6	34.3

Table 5.3: Block-building evaluation using GRO.

attempting to merge them adds significant runtime. For GRO, these equalities are not consistently present, and hence, the reduction of n_p does not occur as frequently as in the case with PF. This test clearly shows that block-building negatively impacts performance in these cases. Given its poorer performance on the GRO dataset, combined with its poor scalability, block-building is excluded from further consideration.

This section has presented the results for the heuristic methods and identified the DBLF-prioritized EMS algorithm as the top performer for the PF dataset. It demonstrates both speed and reasonable performance, with the median F_R, converging around 0.7 when block-building is not applied. This suggests a moderate level of space utilization as it is neither poor nor optimal. Although it is not finding optimal results consistently, the results indicate good scalability, as performance remains stable beyond $n_p > 20$, with consistently fast computation times. A more notable concern is the high variance in performance. Despite this, the DBLF heuristic shows promise and could potentially deliver more consistent results when integrated with additional optimization techniques. Currently, it is limited to exploring only a single item placement order, which could be improved by a GA or DRL.

5.3 Genetic Algorithm

This section aims to present the results of the different tests regarding the GA, followed by a description of the final algorithm design. First, the testing for which chromosome should be utilized will be performed, followed by an iterative grid search to conclude which final parameters should be used.

5.3.1 Encoding

The results regarding the chromosome design can be viewed below.

Solver	Mean FR [%]	Mean time [s]	Invalid Solutions [%]
1-DoF	84.4	12.6	2.7
2-DoF BCI	77.7	3.3	2.3
2-DoF VBO	67.7	31.1	8.6
3-DoF	49.4	4.4	13.8

Table 5.4: Comparison between GAs using different chromosomes.

As seen in Figure 5.3, chromosomes that do not utilize the VBO gene outperform those that do in terms of packing efficiency. This is likely due to a low (G, N) parameter combination, as the solution space for the 1-DoF and 2-DoF BCI is a subset

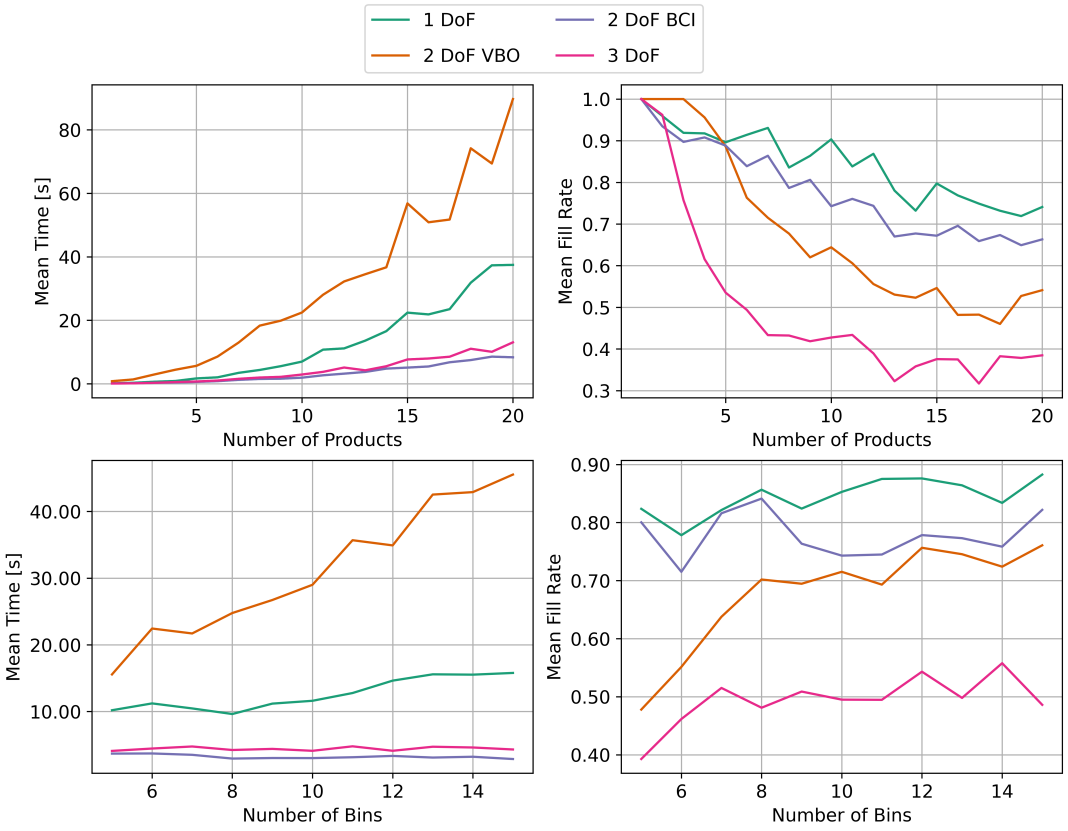


Figure 5.3: Different Chromosome Evaluation.

of the one produced when the VBO gene is included. In theory, adjusting these parameters could improve the performance of the more advanced solvers; however, it is unclear to what extent this would impact the mean solving time. Given the already high computational cost, it may be preferable to exclude the VBO gene. For further testing, the 3-DoF and 2-DoF VBO are hence disregarded.

The BCI gene significantly improves runtime performance but at the cost of a reduced F_R. This trade-off can potentially be mitigated by increasing the (G, N) values. The advantages of including the BCI gene are that it offers greater control over the GA, enabling further optimization strategies, such as implementing a more guided mutation method for the BCI gene, and that the n_b does not negatively affect runtime as it does with the 1-DoF. The drawback is that, because of GA stochasticity, the risk of not finding the optimum is expected to increase since now every combination will not be tested every iteration, such as with 1-DoF. However, because of the increased runtime, the BCI is decided to be included.

5.3.2 Multiprocessing

The computation time complexity is rather high, showcasing that these types of GAs may not be suitable as n_p increases. These tests were done using only one core,

and hence, it was tested whether using multiprocessing improves upon that. Multiprocessing was used to calculate the fitness scores since it is the most demanding task, and order is unimportant, causing no need for synchronization. The results for the multi-processed 2-DoF BCI GA in comparison with the regular one can be seen in Figure 5.4.

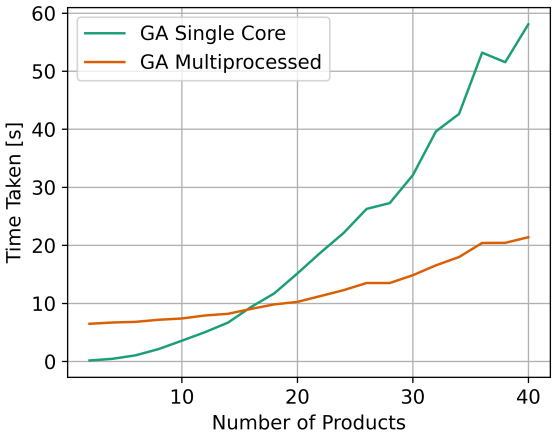


Figure 5.4: Multiprocessing Evaluation.

As can be seen, there is a breakpoint at 16 products, where the overhead associated with distributing tasks across multiple CPUs becomes outweighed by the gains in computation time. For the final GA, it is decided to utilize multiprocessing whenever $n_p \geq 16$.

5.3.3 Tuning

To begin the tuning process, the parameters G and N are analyzed, as they primarily influence the overall runtime. While other parameters can also impact runtime, they are quite insignificant in comparison. As stated earlier, the most time-consuming task is calculating the fitness scores, which solely depend on the product of $G \times N$. With this as background, the project will now test different combinations of G and N to try and find the combination that provides the best results within a reasonable time. This test was done with a scaling factor of 2 and on 200 orders. During these tests, early termination was not allowed, which provided the following results.

The results indicate that $N = 100$ consistently delivers the best performance in terms of execution time across the majority of test instances. Although $N = 5$ occasionally outperforms other settings, this is likely due to its extremely high selection pressure. With a tournament size of 5, the algorithm becomes fully elitist at each iteration, which is an unconventional approach but includes cases with good performance. Despite these occasional benefits, $N = 100$ provides more reliable and stable performance overall. As such, it will be adopted as the default configuration going forward. While exploring population sizes beyond 100 could offer further

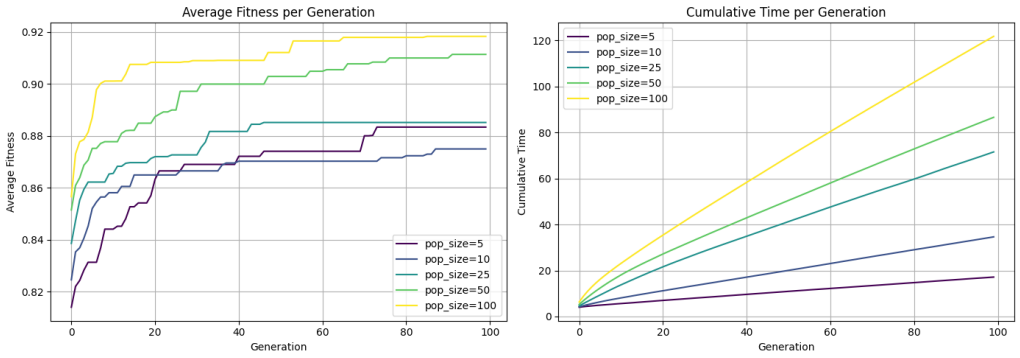


Figure 5.5: Population and generation effect on performance.

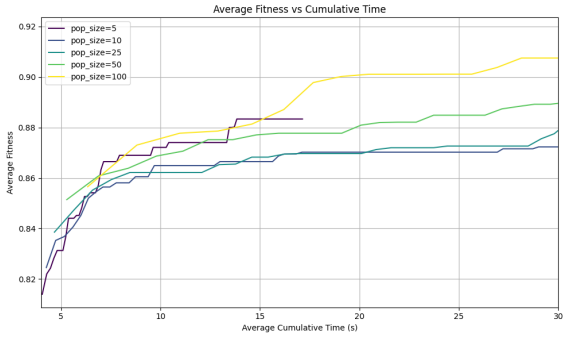


Figure 5.6: Fitness depending on time for different N.

insights, the current minimum mean cumulative runtime is already around 7 seconds. Increasing the population size beyond this point may reduce responsiveness, especially in scenarios where early termination is necessary to meet warehouse or customer demands. Additionally, convergence appears to slow down after $G > 10$, with the average filling rate increasing by only 0.02 over the remaining 90 generations. While further parameter tuning probably can improve convergence, the already high cumulative runtime makes it impractical to significantly increase the number of generations. From a scalability standpoint, it is therefore not advisable to exceed 10 generations in practical applications significantly. However, to better understand the convergence behavior, the tests going forward will primarily use $G = 30$. This allows us to observe potential improvements beyond 10 generations, even if the final implementation is unlikely to use such a high value.

The following tuning will be done through iteratively tuning parameters, because of long runtimes, it is not feasible to test all combinations at once; therefore, they are taken one by one or two by two.

In each grid evaluation, the best parameters were chosen as the basis for the next search. Overall, it was difficult to observe clear trends in the parameter space, as the best values in most grids appeared scattered. This could be attributed to statistical noise, as most tuning runs were conducted on instances including 200 orders, and since the GA is stochastic, it could perhaps benefit from increasing this number.

Parameters	Iteration	Parameter Grid	Best parameters
$\begin{bmatrix} p_{tour} \\ k_{tour} \end{bmatrix}$	1	$\begin{bmatrix} 0.25 & 0.5 & 0.75 & 1.0 \\ 5 & 10 & 25 & 50 \end{bmatrix}$	$\begin{bmatrix} 0.5 \\ 5 \end{bmatrix}$
$\begin{bmatrix} p_{tour} \\ k_{tour} \end{bmatrix}$	2	$\begin{bmatrix} 0.3 & 0.4 & 0.5 & 0.6 \\ 2 & 4 & 6 & 8 \end{bmatrix}$	$\begin{bmatrix} 0.6 \\ 2 \end{bmatrix}$
$\begin{bmatrix} p_{mut} \\ \sigma_{mut} \end{bmatrix}$	1	$\begin{bmatrix} 0.1 & 0.25 & 0.5 & 0.75 \\ 0.25 & 0.5 & 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 0.75 \\ 1 \end{bmatrix}$
$\begin{bmatrix} p_{mut} \\ \sigma_{mut} \end{bmatrix}$	2	$\begin{bmatrix} 0.6 & 0.7 & 0.75 & 0.8 \\ 0.75 & 0.9 & 1 & 1.2 \end{bmatrix}$	$\begin{bmatrix} 0.6 \\ 0.75 \end{bmatrix}$
$\begin{bmatrix} b_{BPS} \\ b_{BCI} \end{bmatrix}$	1	$\begin{bmatrix} 0.75 & 0.9 & 1 & 1.1 & 1.25 \\ & 0.75 & 0.9 & 1 & \end{bmatrix}$	$\begin{bmatrix} 1.25 \\ 0.9 \end{bmatrix}$
$\begin{bmatrix} \text{Top} \\ \text{Top}_{mut} \end{bmatrix}$	1	$\begin{bmatrix} 0 & 1 & 3 & 5 & 10 \\ 0 & 1 & 3 & 5 & 10 \end{bmatrix}$	$\begin{bmatrix} 5 \\ 0 \end{bmatrix}$
$\begin{bmatrix} \text{Top} \\ \text{Top}_{mut} \end{bmatrix}$	2	$\begin{bmatrix} 5 & 6 & 7 \\ 0 & 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 7 \\ 0 \end{bmatrix}$
$\begin{bmatrix} N_{bias} \end{bmatrix}$	1	$\begin{bmatrix} 1 & 10 & 20 & 40 \end{bmatrix}$	$\begin{bmatrix} 10 \end{bmatrix}$
$\begin{bmatrix} N_{bias} \end{bmatrix}$	2	$\begin{bmatrix} 5 & 8 & 11 & 14 \end{bmatrix}$	$\begin{bmatrix} 5 \end{bmatrix}$

Table 5.5: Iterative Grid Search.

Ideally, more extensive tuning on more orders would provide more reliable conclusions, but this was not feasible due to time constraints. Another possible reason for the difficulty in identifying clear trends is the interdependence of the parameters. Rather than each parameter having an isolated effect, their influence is often dependent on several other GA settings, making it difficult to observe clear correlations. Additionally, the performance differences between parameter settings were generally only a few percentage points, which increases the risk of stochastic variation. This suggests that the overall effectiveness may be more attributable to the large number of heuristic variations applied, rather than to specific behaviors intrinsic to the genetic algorithm. In other words, it might be the large number of runs of the heuristic that causes the major benefits, rather than fine-tuned genetic dynamics. Based on the iterative grid search, the final parameter values are summarized in Table 5.6:

To summarize this section, the final GA configuration will utilize two genes: one representing product placement order and one representing bin combination index. It will employ the parameters shown in Table 5.6. With this algorithm developed, the following section presents the results of the DRL approach.

Notation	Name	Value
G	Generations	30
N	Population size	100
Top	Number of Elites	7
p_{mut}	Mutation rate	0.6
k_{tour}	Tournament size	2
p_{tour}	Tournament probability	0.6
N_{bias}	Biased initiations	1
k_{mut}	Mutation step	1

Table 5.6: Final GA Parameters.

5.4 Deep Reinforcement Learning

The DRL algorithm was evaluated on 1000 orders of the PF dataset, and compared to the SF heuristics which it was built upon. The comparisons where made to see wether or not the DRL achieved an increase in performance.

Solver	Average F_r [%]	Average time [ms]
DRL	74.58	8.4 ms
DBLF EMS	74.55	3.3 ms

Table 5.7: Comparison between the DRL and SF heuristic.

Table 5.7 shows that the DRL made very small improvements compared to the heuristic solution alone. Indicating that it is hard to generalize and learn a strategy for what order works best in conjunction with the SF heuristics. Although the results are slightly disappointing the increase in F_R indicates that it is possible to learn a strategy that improves upon the heuristic. Based on these results the DRL agent will not be evaluated further.

5.5 Mixed-Integer Linear Programming

Firstly the CP-SAT and SCIP solvers were evaluated against each other by running them both on 1000 orders on the PF dataset with a time limit of 60 seconds. At this time, it will return the best solution it currently has found. The average times and FRs are shown in Table 5.8, as can be seen, CP-SAT performed very well whilst SCIP was struggling to keep up with the pace and performance of CP-SAT. Worth noting is that CP-SAT solved all problems whilst SCIP could not find a feasible solution for 1.7% of the orders. Based on this result, no further efforts will be made using SCIP.

CP-SAT was evaluated further to see how well it scales to unusually large orders, therefore more tests were conducted on the PF dataset with $k = 5$. During these tests, different time limits were tested to find a nice balance between performance

Solver	Mean F_R [%]	Mean time [s]	Invalid Solutions [%]
CP-SAT	99.8	1.3	0.00
SCIP	71.4	29.0	1.7

Table 5.8: Comparison between CP-SAT and SCIP.

loss and speed. The average time and performance with respect to how many products made up the orders are shown in Figure 5.7. Table 5.9 then shows the average over all orders for the different time limits. A box plot of the runs is also shown in Figure 5.8, to visualize and evaluate robustness. It can be seen that a time limit of 10 seconds performs poorly on larger orders, however the time limits of both 30 and 60 seconds can still provide rather high filling rates, Figure 5.8 shows that the largest difference between 60 and 30 seconds comes down to the variance in the solutions where 60 seconds has a smaller variance in most cases leading to a higher mean value of F_R. Also quite interesting is that the time limit of 120 seconds has a very low improvement compared to 60 seconds and most importantly there are the same amount of invalid solutions between them.

Time limit [s]	Mean F_R. [%]	Mean time [s]	Invalid Solutions [%]
10	66.2	7.8	1.4
30	86.2	21.9	0.2
60	88.3	41.6	0.1
120	89.4	79.5	0.1

Table 5.9: Comparison between different time limits for CP-SAT.

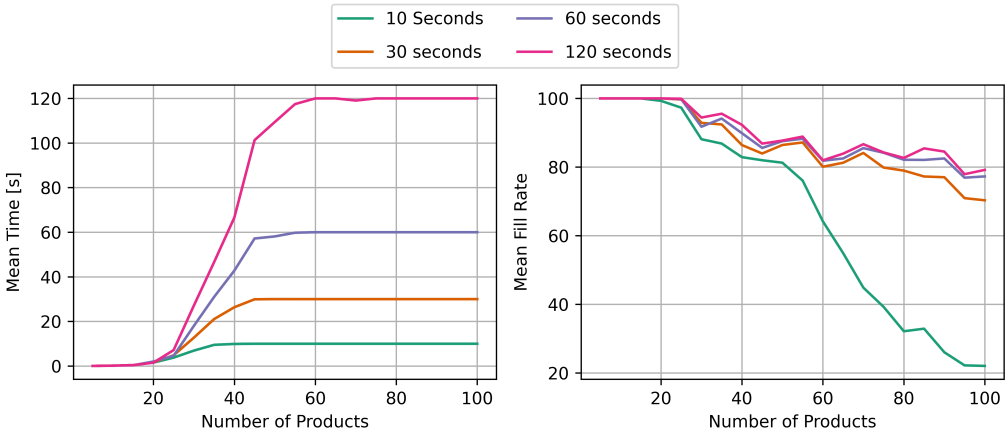


Figure 5.7: Average fill rate and time taken on the PF dataset using different time limits for CP-SAT.

Further on the evaluations will be performed on CP-SAT using a time limit of 60 seconds, mainly because 10 seconds does not scale well as illustrated in Figure 5.8 and because the variance and most importantly the amount of invalid solutions where lower compared to the 30 second time limit while staying the same as for the 120 second one. CP-SAT will also terminate before the time limit when a solution is

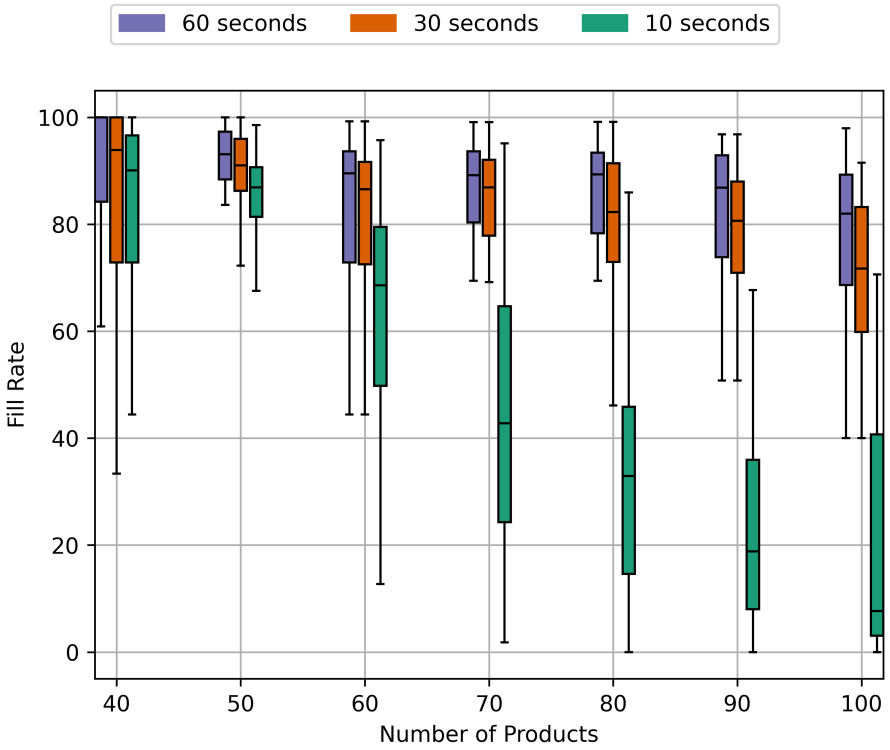


Figure 5.8: Box plots of the CP-SAT F_R on the PF dataset using different time limits.

proven optimal which it shows to do for smaller orders only leaving the longer time limit there for a safety measure when dealing with the occasionally large orders.

5.6 Evaluation of final algorithms

Now that the design and development of the algorithms have been presented, a final evaluation and comparison will be conducted. The RO and GRO datasets will be used for this; all the algorithms results on PF have already been presented and the RO and GRO datasets resemble the real world scenarios where the algorithms are meant to be use.

5.6.1 Perfect Fit

For the comparison of all solvers on the PF dataset, a scaling factor of $k = 2$ was used for all methods except DRL, which ran with $k = 1$. The results, shown in Figure 5.9, reveal that both the heuristic solver (DBLF EMS) and the DRL approach solve problems exceptionally fast, albeit with a lower F_R compared to the GA and CP-SAT solvers. Notably, the DRL method shows no improvement over the heuristic solver. Meanwhile, the GA achieves a significant improvement over the heuristic method but still falls short of CP-SAT's performance. Additionally, GA's runtime for larger orders approaches that of CP-SAT, and it even solves smaller orders more slowly. In contrast, CP-SAT solves nearly all problems with $n_p < 20$ optimally,

maintains a high F_R for larger orders, and keeps runtimes within a feasible range.

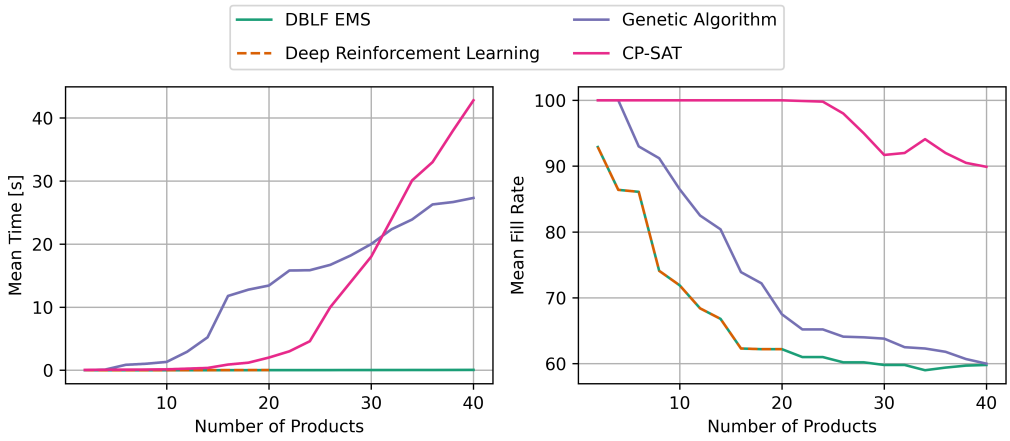


Figure 5.9: The best performing algorithms results on the PF dataset.

5.6.2 Real Orders

To evaluate the solvers on real orders and get a comparison between human workers and the solvers, the RO dataset was used. The results can be seen in the following tables. Results marked by a star (*) denote a proven optimum. The timeout used by CP-SAT was 60 s, and times marked with a line (-) means that the solver timed out before finding and proving an optimum.

Order	CP-SAT		GA		DBLF EMS		Human
	F_R [%]	t [ms]	F_R [%]	t [ms]	F_R [%]	t [ms]	F_R [%]
1	53.6*	61	53.6	586	53.6	17	461.3
2	51.6*	67	37.7	317	37.7	12	82.4
3	18.3*	48	18.3	437	18.3	9	57.2
4	12.3*	49	12.3	54	12.3	2	12.3
5	32.3*	49	32.3	167	32.3	~ 0	32.8
6	22.6*	48	22.6	97	22.6	1	22.6
7	19.6*	65	19.6	135	7.4	1	20
8	80.3*	49	61.8	576	61.8	2	494.3
9	91.6*	66	91.6	98	91.6	4	3.1
10	41.2*	48	40.9	970	40.9	1	3.0
Average	42.3*	55	39.1	344	37.9	6	118.9

Table 5.10: Results for small dataset.

Table 5.10 - 5.12 shows that CP-SAT performs better than the other algorithms in real-world scenarios. However, one can also observe that the performance differences between the solvers are decreased for the RO-dataset compared to the PF-dataset. It can also be seen that the heuristic is drastically faster than the other solvers.

Order	CP-SAT		GA		DBLF EMS		Human
	F_R [%]	t [ms]	F_R [%]	t [ms]	F_R [%]	t [ms]	F_R [%]
1	28.1*	339	28.1	209	28.1	3	28.1
2	87.9*	583	67.3	1389	67.3	2	54.0
3	69.7*	1810	60.7	2397	60.7	14	469.4
4	43.8*	381	43.8	252	43.8	3	43.8
5	45.7*	297	45.7	2926	45.7	2	102.8
6	9.3*	225	9.3	2943	9.3	13	98.4
7	30.1	–	30.3	3635	30.1	3	67.7
8	54.1*	372	53.8	264	53.8	2	54.1
9	10.1*	420	10.1	2783	10.1	30	37.3
10	30.1*	443	30.1	203	30.1	3	13.4
Average	40.88	6847	37.9	1900	37.9	8	96.9

Table 5.11: Results for medium dataset.

Order	CP-SAT		GA		DBLF EMS		Human
	F_R [%]	t [ms]	F_R [%]	t [ms]	F_R [%]	t [ms]	F_R [%]
1	71.1	–	71.1	13564	53.1	25	4.0
2	70.4	–	70.4	18167	70.4	19	101.7
3	29.9*	6910	29.9	17205	29.9	47	4.2
4	83.6	–	83.6	17075	83.6	24	12.4
5	84.1	–	64.4	20566	64.4	27	124.5
6	70.8	–	70.8	23990	70.8	16	101.9
7	83.9	–	64.3	17217	64.3	34	279.6
8	67.4	–	80.5	10490	80.5	11	79.8
9	63.5	–	63.5	14608	63.5	33	154.5
10	49.4	–	49.8	9578	49.8	5	111.2
Average	67.3	54691	64.8	16246	63.0	24	89.3

Table 5.12: Results for large dataset.

As for the human F_R it is observed to exceed 100% in some orders, which would be an unfeasible solution for the solvers. This is probably caused by the limitations of the modeling, such as inaccurate measurements, approximation of complex-shaped products as rectangular prisms, and inclusion of soft packages. The summation of these model limitations finally allows the human packer to pack in a manner that is infeasible for the model.

The solvers have now been compared on real orders. Due to there only being 30 real orders available, the next section will evaluate the solvers on the GRO dataset to reduce the stochasticity in the evaluations, and review whether it can provide solutions within a reasonable time for day-to-day distributions of n_p.

5.6.3 Generated Real Orders

To evaluate the solvers' performance in an environment more similar to day-to-day operations, the GRO dataset was used. When using this dataset, the optimal solution is not known. However, since CP-SAT can prove that its solution is optimal, one can see how many optimal solutions are found. Table 5.13 shows the algorithm's performance on a thousand orders from the GRO dataset. DBLF solves almost all the orders to the optimal solution in an instant; CP-SAT finds the optimal solution to all the orders, but with a slower solving time. This aligns with the tests conducted on the previous datasets; CP-SAT performs better than DBLF, but it comes at a cost of longer time. DRL is included in this test to show that it makes no improvement from the heuristics alone for this dataset. The GA has the second highest amount of optimal solutions, but as can be seen, this comes at a large runtime.

Solver	Optimal solutions [%]	Mean solving time [ms]	Mean F_R
CP-SAT	100.0	61.3	37.0
DBLF EMS	94.8	1.1	36.3
GA	98.2	703.2	36.9
DRL	94.8	6.5	36.3

Table 5.13: The best algorithms performance on 1000 orders of the GRO dataset.

5.7 Summary

The experiments conducted with their results have now been presented throughout this chapter with brief discussions on how to interpret them. The next chapter will elaborate on these discussions in a critical manner highlighting the limitations of the model and suggesting improvements to be made. The conclusions to be drawn from these experiments will also be presented and discussed.

6
Discussion

This chapter elaborates on the results presented in Chapter 5. Then, there will be discussions about the entire thesis, including the limitations in the model and potential further improvements. A discussion about the environmental aspects and human interaction will also be included to shed some light on potential ethical effects with implementation.

6.1 Key findings

This section presents general findings concerning the applicability of the tested heuristics and their overall performance. It aims to identify which solver demonstrates the best results, under which conditions it is most applicable, and other performance affecting causes.

6.1.1 Solver Comparison

After the results, it is now possible to conduct a general comparison of different solvers and to discuss the scenarios in which each method is most suitable. It outlines the strengths and weaknesses of various solvers, helping to determine the best approach depending on the specific context. A more detailed and technical analysis regarding improvements can be viewed in Section 6.2.3.

CP-SAT

CP-SAT is shown to outperform other solvers in finding high-quality solutions within reasonable time constraints for all datasets. This makes CP-SAT the most reliable solver for general cases, which contrasts with earlier research that primarily highlights DRL techniques or metaheuristics as state-of-the-art [7][9]. Notably, this performance advantage appears unique to CP-SAT as SCIP, for instance, demonstrates considerably weaker results. Gurobi, as shown in [6], also fails to generate satisfactory solutions within reasonable time limits. This is surprising since the 3D-BPP inherently is a MILP problem, and hence traditional MILP solvers were initially expected to perform better. One reason behind this result could be that the model contains a large number of Boolean integer variables, which resemble SAT-style constraints, an area where CP-SAT can perform well. Moreover, CP-SAT utilizes LCG, a technique that fits well with problems with vast integer search spaces. By rewriting integer constraints into Boolean logic, CP-SAT can exploit SAT-solving techniques

more efficiently, likely contributing to its performance. Another key factor is parallelism. CP-SAT was designed with multi-core execution in mind and was allowed to run on 12 cores during evaluation. While Gurobi also supports parallel computation, SCIP does not, giving CP-SAT a further advantage over it. In summary, CP-SAT's strong performance can likely be attributed to its combination of LCG, effective handling of SAT-style constraints, the ability to integrate LP relaxations, and its support for parallel processing. This thesis also tried to further speed up the CP-SAT solver by providing an initial solution hint from the heuristic; this did not provide any benefits, and this can be due to the heuristic solutions being far from the optimal solutions in the search area.

Interestingly, this thesis has not found any prior application of CP-SAT to the 3D-BPP, suggesting that this use case represents a novel and potentially valuable research direction. It is worth noting that MILP models such as the one used in this thesis are straightforward to extend by adding more equations and variables. This adaptability makes CP-SAT well-suited for added constraints such as weight, but this remains true only if runtimes are not affected gravely. If they are, another solver might be more feasible.

Heuristics

There exist scenarios in which switching to heuristic methods might be justified, such as when dealing with exceptionally large orders. This is because heuristics tend to converge in terms of filling rate, while CP-SAT exhibits a clear decline in performance. Nevertheless, these large orders are rare in practice, particularly within the context of e-commerce. Moreover, when order sizes approach or exceed 150 items, cumulative measurement errors combined with model limitations can cause the models to become irrelevant, and the task may ultimately fall to a human packer regardless. From a practical standpoint, and in the scope of this thesis, which focuses on e-commerce applications, it is likely unnecessary to account for such extreme cases, as they introduce so many practical and modeling issues that the choice of packing algorithm may become the least of the concerns. Another scenario could be if time were truly critical, then heuristics could also be the more suitable option. For instance, CP-SAT averages around 61 ms on the GRO dataset, fast enough for most picking processes. However, in scenarios where even minimal delays matter, heuristics offer almost instant results. Additionally, since more complex solvers place a heavier load on servers, heuristics can also help reduce computational demand when low server utilization is a priority. As mentioned in the earlier section, heuristics can perform almost equally well as search-based methods when the optimum filling rate is low, even though it might be beneficial to implement a heuristic in these cases, it also limits the potential if a warehouse chooses to improve their bins available. CP-SAT is hence preferable compared to heuristics in most cases, but under certain rare circumstances, sticking to basic heuristics can be preferable.

It is worth restating that one limitation of heuristic algorithms is that they are often only as effective as their underlying strategy. Allowing them to run for longer

does not necessarily yield better results. Furthermore, depending on the strategy employed, a heuristic algorithm may, in some cases, lack the capacity to find a decent solution, as can be seen with the high number of invalid solutions. This strategic limitation extends to other solvers that depend on heuristics, suggesting that relying too heavily on them should be avoided. However, given the vastness of the optimization space, both the GA and DRL approaches struggle to optimize effectively without significant heuristic guidance. Generally, this thesis recommends the DBLF-EMS heuristic since it can provide fast results with a decent filling rate.

Genetic Algorithm

The GA performs well on the PF dataset and improves upon the existing heuristic in terms of packing efficiency, but this comes at the cost of significant computational complexity. Since CP-SAT already delivers strong efficiency, the GA, in its current form, does not offer any clear advantages. It might become viable if heavily tuned to run with fewer generations and smaller populations, narrowing the gap between it and the heuristic approach at the cost of performance. Additionally, other GA-based solutions have demonstrated better complexity than the approach in this thesis, but they rely on a different 3D-BPP formulation, which includes only a single bin size, and thus, the bin selection process will differ. It is believed that this GA can be further optimized, though achieving this would require additional development. In short, the GA do perform well but would need to be either further optimized or trimmed to compete with CP-SAT.

Deep Reinforcement Learning

This thesis has explored DRL extensively, testing various encoders such as transformers and feed-forward networks. It has been tested by implementing different DRL methods and networks. Several reward structures were analyzed, and while the network demonstrates improvement over time, the results have not been as satisfactory as initially anticipated. Although DRL has the potential to compete with other solvers, as shown in the literature, further development is necessary to improve its performance. One of the key advantages of DRL is its ability to offer comparable runtime to heuristics. However, the current heuristic approach may not be ideal for DRL applications, as it might be difficult to find a general order strategy for this heuristic. It could be beneficial to experiment further with different degrees of freedom between the heuristic and DRL. Other possibilities consist of experimenting further with the network architecture and reward structure.

To finalize this thesis's recommendation for which method to use, one must first determine the application's priorities. In the general case, CP-SAT is the most suitable option. However, there are certain niche situations where the heuristic approach may be more appropriate; for example, when speed is prioritized over solution quality, when the algorithm must run on a server with limited computational resources, or when handling unusually large orders. This is illustrated by the decision

tree shown i Figure 6.1.

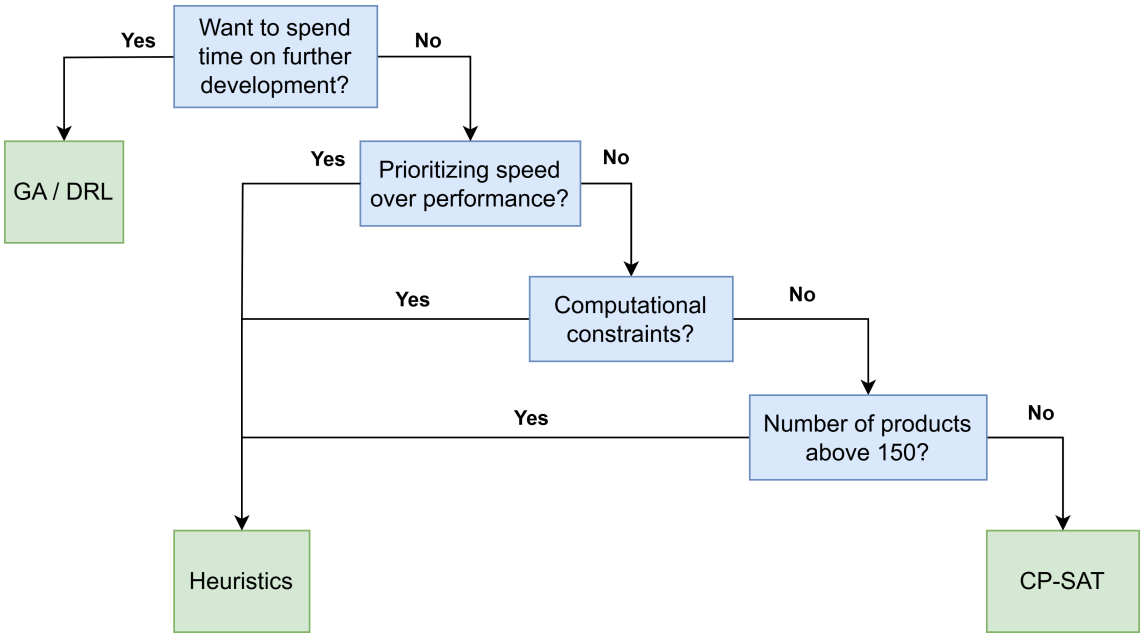


Figure 6.1: Decision Tree for this thesis implementation recommendation.

6.1.2 General Applicability

Based on CP-SAT's results, some conclusions can be drawn. Firstly, this solver is capable of finding optimal solutions in a low amount of time, when the number of products is within reasonable limits. This indicates that it can perform well in real-world scenarios. The fact that a human packer can achieve a fill rate exceeding 100% suggests inaccuracies in the model, such as measurement errors, but does not imply that the solver is infeasible. For CP-SAT to be truly applicable in practice, it must be able to outperform human performance. It is reasonable to believe that this is achievable if the model accurately reflects the warehouse environment, since the solver has demonstrated the ability to find optimal solutions. This thesis has found that CP-SAT can outperform human workers and solve the 3D-BPP within time limits if the following conditions are true:

1. Accurate bin and item measurements are provided.
2. Items are not overly soft or irregularly shaped.
3. The number of items is within normal bounds ($n_p \leq 20$)

If any of these conditions are not met, a decrease in packing efficiency is to be expected. While some degree of compromise may be acceptable, it is important to recognize the cause of any resulting suboptimal performance.

6.1.3 The Carton Selection Problem

Secondly, even when accurate data is available, a lack of appropriately sized packing cartons in the warehouse will inevitably lead to bins containing significant amounts of void space. The PF dataset shows that solvers such as CP-SAT and the GA can consistently find quite good solutions when the warehouse has an ideal set of bins in stock and n_p remains within typical bounds. The tests using the RO and GRO datasets demonstrate that the solvers can still find an optimal solution even though the filling rate is rather low. Furthermore, under the conditions where the bins are not as well matched, the performance gap between heuristics and more advanced solvers diminishes significantly. This suggests that when the available bin types are ill-suited to the items being packed, the benefits of using a more complex algorithm, such as CP-SAT or GAs, are largely negated. In short, the carton selection problem plays a crucial role in overall efficiency. While implementing a bin-packing algorithm in warehouses can be beneficial, the absence of appropriately sized cartons often makes simpler heuristics a more practical choice.

Also, since box selection is not the main cause of box vacancy in these cases, the expected benefits of box selection automation should be lowered. Another takeaway related to the carton selection problem is that the number of available bin types does not significantly impact bin packing solver runtime, but increases packing efficiency. It is more likely that the practical challenges of managing a larger number of packing boxes will become problematic before any computational issues arise. Therefore, it is advisable to utilize as many bin sizes as are practically manageable, since it improves the bin-packing algorithms without increasing runtime significantly.

This section has concluded that CP-SAT has the best performance for the evaluations done in this thesis. The potential of outperforming manual bin packing under specific conditions has been highlighted. The importance of having suitable bins in stock has been noted, as well as the fact that increasing the number of available bins generally leads to improved performance. Moving forward, discussions about how to build upon this work will be presented and provide solver-specific ideas for improvement.

6.2 Further Work

There are several potential directions for extending this project. The following section outlines the most promising next steps and includes brief discussions to describe the suitability of each extension.

6.2.1 Implementation and Evaluation

The next step for this project should be to integrate CP-SAT and the proposed model into the warehouse management system and see if it provides practical benefits. This thesis considers the algorithm ready for implementation, which would allow for real-world testing and feedback. To evaluate its effectiveness in a practical

warehouse setting, this thesis proposes the following evaluation criteria:

1. Collect operational data to assess whether packing efficiency improves with the algorithm in place.
2. Monitor whether the picking process is affected in terms of time and workflow.
3. Observe to what extent the algorithm's packing suggestions are followed by warehouse staff.

This would offer real-world performance data that could be evaluated, as certain aspects might have been missed during the modeling that are necessary for implementation.

6.2.2 Carton Selection Suggester

Aside from implementing CP-SAT and evaluating it in a practical environment, developing a carton selection suggester is a critical next step. As noted, packing efficiency is ultimately constrained by the types of packing bins available in the warehouse. If the bins do not allow for tight packing, the performance of the BPP solver will be limited. Therefore, it is recommended to develop a bin size suggestion tool that analyzes typical orders in the warehouse and uses this data to recommend optimal bin sizes to use.

6.2.3 Algorithm Refinements

While further development of all presented solvers is possible, it is likely more valuable to focus on improving carton selection and assessing the real-world performance of CP-SAT before revisiting the 3D-BPP. That said, if there is ever a need to revisit alternative algorithms, specific improvement suggestions are outlined below.

Heuristics

There is potential for further optimization regarding the heuristic approach, particularly in refining the sorting algorithm. Currently, the algorithm primarily relies on volume, which yields good results but can struggle with items that are highly non-cubic. From a practical standpoint, humans typically prioritize such items early in the packing process. However, the DRL experiments suggest that the current sorting strategy is the most suitable one, as the DRL model struggles to outperform it. Additionally, the GA tuning appears to favor this strategy over both alternative sorting methods and random sorting. That said, there may still exist some undiscovered formula based on the items' dimensions that provide better results. Additionally, testing different bin prioritization strategies and experimenting with bin selection methods beyond the current trial-and-error approach could further enhance the heuristic's performance.

Genetic Algorithm

Improving the GA would primarily involve enhancing its scalability, as it already delivers strong performance. Since the most time-consuming component is the calculation of fitness scores, the greatest potential for improvement lies in further optimizing the underlying heuristic method. Although significant effort has already gone into this optimization, it is likely not yet fully optimal. Currently, most computationally intensive tasks such as collision checks and EMS handling are performed using vectorized NumPy operations, which are generally efficient. However, there is likely still room for further performance gains.

Currently, due to the choice of a 2-DoF chromosome, the algorithm relies on trial and error across multiple bin sets. This approach is not computationally optimal and could potentially be improved by implementing a more guided bin gene. Biased bin genes were tested, but a major issue emerged as many individuals became invalid due to unfit bin assignments, leading to too high a selection pressure and premature convergence. While this approach shows promise, it would likely require additional biasing or heuristic guidance to be effective.

DRL

While DRL solutions have shown the potential to outperform GAs in the literature, the DRL solver used in this thesis did not yield impressive results. This does not imply that DRL is unfeasible for the problem, but rather that some initial design choices may have limited its performance. Since the reasons behind the early stagnation of the DRL model remain unclear, the following DRL design areas are suggested for further experimentation:

- **Change the heuristic**: Some DRL-based approaches operate without heuristics, while others rely heavily on them. In this case, changing to another heuristic and experimenting with the responsibility balance between the agent and the heuristic might provide benefits.

- **Modify the environment**: The reward structure and observation space are critical design choices. Implementing a voxel grid, or allowing the agent access to the full EMS list could lead to better learning outcomes. Additionally, the reward structure currently used may yield negative rewards in too many stages causing the model to instead terminate early, when the final reward does not outweigh the intermediate rewards. The reward structure is critical to the success of a DRL and this one might benefit from a redesign of the reward structure.

- **Revise the network architecture**: DRL architectures in the literature vary a lot, but many use combinatorial encoders such as pointer networks or transformers. Although transformer encoders were tested in this thesis and yielded only modest improvements during initial trials, a more comprehensive inves-

tigation into network architectures, both with and without combinatorial encoders, may be necessary to find clear benefits.

CP-SAT

There may be some room for improvement, as it has been demonstrated for the one-dimensional case that performance can be enhanced [59]. However, it remains uncertain whether similar benefits can be achieved for the 3D-BPP, and to what extent the filling rate would be compromised by allowing suboptimal formulations. Furthermore, while improving the internal functionality of the CP-SAT solver is possible, it is already a highly optimized system, and modifying its source code may not be a practical option. Unlike the other solutions discussed, experimenting with CP-SAT at a low level is less straightforward, and therefore, it can be difficult to make improvements. It is also worth mentioning that it is not certain that CP-SAT will be the best solver if the model gets altered.

6.2.4 Model Refinements

As with most models, there are some limitations to this one. It is important to understand these before using it to ensure it is fitting for the use case. Understanding the model limitation may also help minimize the efficiency decrease caused by these. This section discusses on the limitations for the model and potential refinements that could be made.

Model limitations and improvements

As seen in Table 5.10 - 5.12, the worker can sometimes fill the bins beyond 100%, which the bin packing solver considers infeasible. This issue arises from the model's limitations. First, there may be differences in the given dimensions. The measurements might have confidence tolerances, or they could simply be incorrect. Secondly, shapes are not always rectangular blocks; they can take irregular forms, which may further explain the differences. Finally, both products and bins can be soft, allowing items to change shape and to be compressed, introducing entirely new packing possibilities.

One way to deal with this could be to improve the model and include more complex shapes if data is available. As seen in the theory chapter, it has been done [15], but it would require the warehouses to start gathering more complex geometric data about the items. This is most likely not worthwhile for most warehouses since it would require some sort of scanning or point cloud generation. Another possibility would be to include softness into the model; something that is possible [65]. It too would require some data gathering, and in this case, needs the deformability of the items, which might not be available for all articles in a warehouse. Similar to rotation, softness is a sort of deformation and should therefore have the possibility to be included in the BPP. Improving the model by considering more aspects will

improve the solver's potential to come up with more and better solutions. The drawbacks are that more data is needed, as well as that the optimization space increases even further, which in turn increases the complexity and difficulty of convergence. The much simpler form of model improvement would consist of improving the measurements and ensuring that they are consistently correct, which is necessary for a model to perform well. This is the first improvement that should be done, and if that still does not provide sufficient results, then it is possible to review whether more advanced models could be applicable.

Model reformulation

It is worthwhile to clarify that the practical problem described in this project is not necessarily a bin packing problem. Rather, it is more accurately framed as a bin suggestion problem, where the primary objective is to suggest bins to use for packing. Bin packing is used to validate that the solvers generate feasible solutions and to be able to provide packing instructions to workers. Ensuring feasibility was a key priority, as deploying a solver that occasionally produces infeasible solutions could lead to workers disregarding its recommendations, as discussed earlier. Even though this thesis decided not to investigate solvers that could risk generating unfeasible solutions, I.E, suggesting bins that do not fit all products, it could be interesting to investigate whether it is suitable to approach this problem as a probabilistic bin suggestion problem. This reformulation could be beneficial in warehouses where model limitations can cause significant issues, such as those handling non-cubic objects (e.g., sporting equipment) or soft materials (e.g., clothing), it may be more practical to abandon the model altogether. Instead of focusing on order, placement, and rotation, a more effective approach might be to learn the direct correlation between direct items and bins to determine whether it improves overall efficiency.

This section has outlined the potential next steps for further development. First, the system can be implemented and tested in a real-world setting to evaluate its practical benefits. Second, implementing a carton suggestion algorithm could improve the overall packing efficiency and the benefits of a bin-packing algorithm. Third, specific improvements to the solving algorithms have been explored. Finally, the section is concluded with a discussion on possible enhancements to the underlying model.

6.3 Sustainability and Human interaction

To ensure that this thesis aligns with ethical and sustainable standards, a brief discussion on these aspects is required.

6.3.1 Sustainability

This project created algorithms for selecting the smallest combination of boxes in warehouse operations. This could potentially optimize the logistics chains from order picking to customers, resulting in not only economic benefits but also environ-

mental. More densely packed boxes provide for better-filled trucks. By maximizing truck and transportation capacity, fewer trips may be required, reducing fuel consumption and greenhouse gas emissions. Maximizing the filling rate of used boxes reduces the amount of material required for their production, thereby decreasing the use of cardboard, plastic, and other packaging materials.

6.3.2 Human interaction

Another important aspect is that this paper does not deal with robot packers. It will be people at the other end of the algorithm that has to look at the suggested bins and their packing instructions. Consider a scenario where a worker, under time pressure, receives an order containing 50 heterogeneous items. While, in theory, these items could fit perfectly into a bin if placed in the optimal order and orientation, the worker is unlikely to value this theoretical optimality as highly as the algorithm designer does. Instead, they may select a larger bin and pack the items in a more practically feasible way. A decision related to the human aspect was prioritizing robustness. It was undesirable to have an algorithm that could risk a solution presenting infeasible solutions. If workers lose trust in the solver, they are unlikely to follow its suggestions, rendering it irrelevant. This issue is particularly significant given that non-compliance with packing instructions has already been documented [66]. Therefore, the human aspect must be carefully considered, regardless of how well the solver performs algorithmically. Its effectiveness also depends on whether workers are willing and able to follow its recommendations.

For the workers involved with picking orders, an effective algorithm could alleviate the stress and frustration caused by choosing the wrong box size. However, if the algorithm makes mistakes due to model imperfections or insufficient testing, it can irritate and force workers to do extra work, something that may add stress if they are pressed for time. Also, the algorithm may not take the bins' ergonomic aspects into consideration, which could result in worse conditions for the worker if the boxes are poorly designed from the beginning.

6.4 Summary

This chapter discussed the entire thesis, highlighting the limitations of the solvers, suggesting improvements to be made and discussing the conclusions to be drawn. The human interpretation and possible environmental impacts has also been discussed. The next chapter summarises the conclusions to be drawn from this thesis.

7
Conclusion

This chapter revisits the research question but also summarizes the conclusions already drawn in the discussion chapter to give the reader a clear view of what this thesis concludes.

Is it possible to implement a solving method for the multiple bin-size 3D-BPP that outperforms human workers in terms of packing efficiency while maintaining a feasible runtime?

This thesis concludes that it is possible to accomplish this if the model accurately reflects the environment. Warehouses need to have accurate measurements of their bins and items, which should be able to be well approximated by rectangular shapes. The number of products in an order should preferably be rather low, as CP-SAT manages to find every packing optimum when the number of products is below 24. Although it can still find an optimum, performance is compromised as we gradually increase this number.

Multiple different solvers, such as CP-SAT, GA, heuristics, and DRL, have shown satisfactory results on real-world data and simulated real-world data, meaning that it should be possible to improve upon human bin packing through a variety of strategies. However, as the warehouses use better bins, CP-SAT's performance is unmatched compared to these other algorithms and should hence be the preferred one for implementation. In cases where the number of products is above 150 or heavy computational constraints exist, heuristic methods are preferred.

If the warehouse's packing boxes aren't optimized for their usual orders at first, the benefits of optimizing bin packing will not be as clear. Therefore, it is recommended to also try and optimize the packing bin stock in connection with a bin packing algorithm implementation.

Bibliography

[1] Soma Amol Dhaigude and Bijuna C. Mohan. Logistics service quality in online shopping: A bibliometric analysis. *Journal of Internet Commerce*, 22(1):1–32, Dec 2021.

[2] SphereWMS. Picking and packing in warehouse operations: What it is and how to do it right, 2024. Accessed: 2024-12-20.

[3] Yu-Chung Tsao, Jo-Ying Tai, Thuy-Linh Vu, and Tsung-Hui Chen. Multiple bin-size bin packing problem considering incompatible product categories. *Expert Systems with Applications*, 247:123340, Aug 2024.

[4] Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Operations research*, 48(2):256–267, 2000.

[5] Jianglong Yang, Kaibo Liang, Huwei Liu, Man Shan, Li Zhou, Lingjie Kong, and Xiaolan Li. Optimizing e-commerce warehousing through open dimension management in a three-dimensional bin packing system. *PeerJ Computer Science*, 9:e1613, Oct 2023.

[6] Sara Wäpling. Optimizing box sizes for shipping in e-commerce - a mixed-integer linear programming and a genetic algorithm approach, 2022.

[7] Anan Ashrabi Ananno and Luis Ribeiro. A multi-heuristic algorithm for multi-container 3-d bin packing problem optimization using real world constraints. *IEEE Access*, 12:42105–42130, Jan 2024.

[8] Xueping Li, Zhaoxia Zhao, and Kaike Zhang. A genetic algorithm for the three-dimensional bin packing problem with heterogeneous bins. In *IIE Annual Conference. Proceedings*, page 2039. Institute of Industrial and Systems Engineers (IISE), 2014.

[9] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*, 2017.

[10] Thanh-Hung Nguyen, Viet-Thang Tran, Phan-Quan Doan, and Thi-Thoa Mac. A novel heuristic algorithm for online 3d bin packing. *2021 21st International Conference on Control, Automation and Systems (ICCAS)*, Oct 2021.

[11] Hang Zhao, Qijin She, Chenyang Zhu, Yin Yang, and Kai Xu. Online 3d bin packing with constrained deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):741–749, May 2021.

[12] Shuo Yang, Shuai Song, Shilei Chu, Ran Song, Jiyu Cheng, Yibin Li, and Wei Zhang. Heuristics integrated deep reinforcement learning for online 3d bin packing. *IEEE Transactions on Automation Science and Engineering*, 21(1):1–12, Jan 2023.

[13] Weiwang Zhu, Sheng Chen, Min Dai, and Jiping Tao. Solving a 3d bin packing problem with stacking constraints. *Computers Industrial Engineering*, 188:109814, 2024.

[14] Qianwen Zhu, Xihan Li, Zihan Zhang, Zhixing Luo, Xialiang Tong, Mingxuan Yuan, and Jia Zeng. Learning to pack: A data-driven tree search algorithm for large-scale 3d bin packing problem. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 4393–4402, New York, NY, USA, 2021. Association for Computing Machinery.

[15] Hongteng Wu, Stephen C.H. Leung, Yain whar Si, Defu Zhang, and Adi Lin. Three-stage heuristic algorithm for three-dimensional irregular packing problem. *Applied Mathematical Modelling*, 41:431–444, 2017.

[16] Huwei Liu, Li Zhou, Jianglong Yang, and Junhui Zhao. The 3d bin packing problem for multiple boxes and irregular items based on deep q-network. *Applied Intelligence*, 53(20):23398–23425, Jul 2023.

[17] Fan Wang and Kris Hauser. Stable bin packing of non-convex 3d objects with a robot manipulator. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8698–8704, 2019.

[18] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Ts2pack: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, 195(3):744–760, 2009.

[19] Anjali Awasthi Batoul Mahvash and Satyaveer Chauhan. A column generation-based heuristic for the three-dimensional bin packing problem with rotation. *Journal of the Operational Research Society*, 69(1):78–90, 2018.

[20] Heiner Müller-Merbach. Heuristics and their design: a survey. *European Journal of Operational Research*, 8(1):1–23, 1981.

[21] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Extreme point-based heuristics for three-dimensional bin packing. *INFORMS Journal on Computing*, 20(3):368–384, Aug 2008.

[22] Chi Nguyen, Trung Thanh Nguyen, Lam Thu Bui, and Ran Wang. An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the physical internet. page 140–155, Jan 2017.

[23] Rommel D. Saraiva, Napoleão Nepomuceno, and Plácido R. Pinheiro. A layer-building algorithm for the three-dimensional multiple bin packing problem: a case study in an automotive company. *IFAC-PapersOnLine*, 48(3):490–495, 2015. 15th IFAC Symposium onInformation Control Problems inManufacturing.

[24] Dequan Liu and Hongfei Teng. An improved bl-algorithm for genetic algorithm of the orthogonal packing of rectangles1this work is supported by the national natural science foundation of china (grant no. 69573004).1. *European Journal of Operational Research*, 112(2):413–420, 1999.

[25] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm- a literature review. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 380–384, 2019.

[26] Oliver Kramer and Oliver Kramer. *Genetic algorithms*. Springer, 2017.

[27] Seyedali Mirjalili. *Genetic Algorithm*, pages 43–55. Springer International Publishing, Cham, 2019.

[28] Stanley Gotshall and Bart Rylander. Optimal population size and the genetic algorithm. *Population*, 100(400):900, 2002.

[29] Edmund K Burke, Matthew R Hyde, and Graham Kendall. Evolving bin packing heuristics with genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pages 860–869. Springer, 2006.

[30] Anupriya Shukla, Hari Mohan Pandey, and Deepti Mehrotra. Comparative review of selection techniques in genetic algorithm. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pages 515–519, 2015.

[31] Jingqing Zhang, Hang Yuan, and Hao Dong. *Introduction to Deep Learning*, pages 3–46. Springer Singapore, Singapore, 2020.

[32] Maxim Lapan. *Deep reinforcement learning hands-on*. Packt Publishing Ltd, 2024.

[33] Aristotelis Lazaridis, Anestis Fachantidis, and Ioannis Vlahavas. Deep reinforcement learning: A state-of-the-art walkthrough. *Journal of Artificial Intelligence Research*, 69:1421–1471, 2020.

[34] Subana Shanmuganathan. *Artificial neural network modelling: An introduction*. Springer, 2016.

[35] Chaity Banerjee, Tathagata Mukherjee, and Eduardo Pasiliao Jr. An empirical study on generalizations of the relu activation function. In *Proceedings of the 2019 ACM Southeast Conference*, pages 164–167, 2019.

[36] Jiawei Zhang. Gradient descent based optimization algorithms for deep learning models training. *arXiv preprint arXiv:1903.03614*, 2019.

[37] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers Operations Research*, 134:105400, 2021.

[38] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[39] Quanqing Que, Fang Yang, and Defu Zhang. Solving 3d packing problem using transformer network and reinforcement learning. *Expert Systems with Applications*, 214:119153, 2023.

[40] Shengyi Huang, Anssi Kanervisto, Antonin Raffin, Weixun Wang, Santiago Ontañón, and Rousslan Fernand Julien Dossa. A2c is a special case of ppo. *arXiv preprint arXiv:2205.09123*, 2022.

[41] Yanhua Huang. *Deep Q-Networks*, pages 135–160. Springer Singapore, Singapore, 2020.

[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[43] Jonathon Shlens. Notes on kullback-leibler divergence and likelihood, 2014.

[44] François Clautiaux and Ivana Ljubić. Last fifty years of integer linear programming: A focus on recent practical advances. *European Journal of Operational Research*, 2024.

[45] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.

[46] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, Jul 2009.

[47] Laurent Perron and Frédéric Didier. Cp-sat.

[48] Jens Clausen. Branch and bound algorithms-principles and examples. *Department of computer science, University of Copenhagen*, pages 1–30, 1999.

[49] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, page 530–535, New York, NY, USA, 2001. Association for Computing Machinery.

[50] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[51] Peter J. Stuckey. Lazy clause generation: Combining the power of sat and cp (and mip?) solving. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 5–9, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[52] Bengt Lennartson. Optimization of timed petri nets using cp-sat. *IFAC-PapersOnLine*, 58(1):90–95, 2024. 17th IFAC Workshop on discrete Event Systems WODES 2024.

[53] José Fernando Gonçalves and Mauricio G.C. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2):500–510, 2013.

[54] Boliang Zhang, Yu Yao, H K Kan, and Wuman Luo. A gan-based genetic algorithm for solving the 3d bin packing problem. *Scientific Reports*, 14(1), Apr 2024.

[55] Kyungdaw Kang, Ilkyeong Moon, and Hongfeng Wang. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied Mathematics and Computation*, 219(3):1287–1299, 2012.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[57] Richa Verma, Aniruddha Singhal, Harshad Khadilkar, Ansuma Basumatary, Siddharth Nayak, Harsh Vardhan Singh, Swagat Kumar, and Rajesh Sinha. A generalized reinforcement learning algorithm for online 3d bin-packing, 2020.

[58] Mhand Hifi, Stéphane Negre, and Lei Wu. Hybrid greedy heuristics based on linear programming for the three-dimensional single bin-size bin packing problem. *International Transactions in Operational Research*, 21(1):59–79, 2014.

[59] Sabino Roselli, Fredrik Hagebring, Sarmad Riazi, Martin Fabian, and Knut Åkesson. On the use of equivalence classes for optimal and suboptimal bin packing and bin covering. *IEEE Transactions on Automation Science and Engineering*, 18(1):369–381, 2021.

[60] Manjeet Singh and Ehsan Ardjmand. Carton set optimization in e-commerce warehouses: A case study. *Journal of Business Logistics*, 41(3):222–235, 2020.

[61] Packoplock. Kartonger lådor. `https://www.packoplock.se/kartonger-lador`. Accessed: 2025-04-11.

[62] A. Galrão Ramos, José F. Oliveira, José F. Gonçalves, and Manuel P. Lopes. A container loading algorithm with static mechanical equilibrium stability constraints. *Transportation Research Part B: Methodological*, 91:565–581, 2016.

[63] K.K. Lai and Jimmy W.M. Chan. Developing a simulated annealing algorithm for the cutting stock problem. *Computers Industrial Engineering*, 32(1):115–127, 1997.

[64] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024.

[65] Ana-Maria Cretu, Pierre Payeur, and Emil M. Petriu. Soft object deformation monitoring and learning for model-based robotic hand manipulation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(3):740–753, 2012.

[66] Jiankun Sun, Dennis J Zhang, Haoyuan Hu, and Jan A Van Mieghem. Predicting human discretion to adjust algorithmic prescription: A large-scale field experiment in warehouse operations. *Management Science*, 68(2):846–865, 2022.

A

Bin Dimensions

The bin sizes used in the PF and GRO datasets are listed in Table A.1, sorted in ascending order by volume. These bin dimensions are taken from [61].

Bin	Length [mm]	Width [mm]	Height [mm]	Volume [dm^3]
7	120	120	120	1.728
12	262	165	50	2.162
13	229	164	115	4.324
8	200	150	150	4.500
6	330	230	61	4.628
15	304	216	220	14.467
1	320	225	250	18.000
11	1000	150	150	22.500
9	290	290	290	24.389
2	440	320	252	35.251
14	445	315	300	42.053
3	570	380	380	82.332
10	1160	300	300	104.400
4	580	380	540	118.872
5	780	580	475	214.395

Table A.1: Bin dimensions and volumes sorted by volume in ascending order

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY