POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

3D Bin Packing: A New Heuristic Approach for Real-Case Scenarios

Supervisors
Prof. Federico Della Croce di Dojola

Candidate
Riccardo Zanchetta

Academic Year 2024-2025

Acknowledgements

I would like to express my gratitude to Professor Federico Della Croce di Dojola for giving me the opportunity to explore this field of study in greater depth, guiding me with advices and expertise throughout this journey.

A special thank you goes to my parents, whose unwavering support has always made me feel confident in the choices I have made. Their trust and encouragement have been a constant source of strength at every stage of my path.

I extend my heartfelt appreciation to my brother, whose example and continuous support serve as both an inspiration and a fundamental point of reference in my life.

I am also deeply grateful to my grandparents, from whom I have inherited intellectual curiosity and determination—values that have accompanied me throughout this journey and will continue to guide me in the future.

Lastly, I cannot fail to thank all those who have shared this experience with me, making it even more meaningful, and my dear friends, whose presence and support have been truly invaluable: Marta, Gaia, Dario, Emilio, Bianca and many others.

Abstract

The 3D bin packing problem (3D-BPP) is one of the most challenging problems in the logistics industry. The objective of this Thesis is to explore the optimization of the 3D-BPP within a logistics context, focusing on several critical goals: maximizing space utilization, minimizing object fragility, and ensuring balanced weight distribution. We present a novel GRASP (*Greedy Randomized Adaptive Search Procedure*) algorithm tailored to address the 3D-BPP. The proposed method integrates several novel elements and builds upon the contributions of key researchers in the field. It also identifies potential improvements to the algorithm and suggests directions for future research to advance the state-of-the-art in solving the 3DPP.

Contents

Chapter 1

Introduction

1.1 Background and motivation

During the last decades the global trade has faced a constant growth that was supported by the supply chain architecture. The European Union covers a pivotal role in the worldwide economy and is nowadays the largest trader of manufactured goods and services [1]. The massive industry of transportation and logistics involves several players, contributing to the development of our economies, at the same time, this compulsive growth has a lot of drawbacks on the environmental side. According to the European commission, logistics activities, including freight transportation and warehousing, are responsible for 6-10% of global CO2 emissions [2]. There are several ways to adress this problem, one of those, analyzed in this thesis is reduce the inefficiences in the supply chain. Within this massive industry, inefficiencies in space utilization, suboptimal route planning, and poorly managed warehouse operations contribute significantly to operational costs and environmental degradation. Among the many processes within logistics that can be optimized, 3D bin packing stands out as a key area where improvement can lead to substantial reductions in both costs and environmental impact. By optimizing how goods are packed into containers, trucks, or storage spaces, companies can minimize the number of trips required for transportation, reduce fuel consumption, and lower carbon emissions. Additionally, better packing efficiency translates into less material waste in packaging, fewer resources required for storage, and reduced damage to goods in transit, further contributing to sustainability goals. With growing pressure from governments and international bodies to reduce carbon footprints, the logistics industry faces a critical need to innovate and adopt optimization techniques that improve both operational efficiency and environmental sustainability. Studies show that up to 20-30% of space in transport containers is often underutilized [3], leading to increased transportation costs and unnecessary environmental impacts. Optimizing these processes through advanced algorithms and intelligent systems can save companies 5-15% in transportation costs, which translates into significant economic benefits given the scale of the industry.

1.2 Description of the 3D bin packing problem

In its simplest form, the bin packing problem was first introduced as a one-dimensional problem. Over time, it has evolved to address higher-dimensional variations, with the 3D bin packing problem (3D BPP) emerging as one of the most complex and practical extensions. The 3D BPP is particularly relevant in applications such as container loading, palletization, and warehouse management, where objects of varying sizes, shapes, and fragility must be packed efficiently into three-dimensional spaces. Solving the 3D BPP optimally can lead to significant savings in transportation and storage costs while improving sustainability through the efficient use of space. Moreover, in real-world scenarios, additional constraints and objectives come into play. For instance, the fragility of objects must be considered to prevent damage, and weight distribution is critical for maintaining load stability. These factors introduce new several constraints, requiring algorithms that can optimize several conflicting criteria simultaneously . This thesis seeks to explore and address these challenges, focusing on new ways to solve 3D bin packing, where the goals include maximizing space utilization, minimizing object fragility, and ensuring balanced weight distribution.

1.3 Thesis Objectives

The objective of this thesis is to explore the optimization of the 3D bin packing problem within a logistics context, focusing on several critical goals.

- First, the **maximization of space utilization** aims to reduce the number of containers or trucks required for transportation, thereby lowering operational costs and decreasing environmental impact through reduced fuel consumption and emissions.
- Second, the **minimization of fragility-related risks** focuses on ensuring that items are packed in a manner that minimizes the chance of damage during transit, addressing both economic losses and customer satisfaction by reducing product damage and returns.
- Third, the **optimization of weight distribution** ensures balanced loads to maintain stability during transportation, which is essential for avoiding accidents, ensuring compliance with transportation regulations, and improving overall vehicle efficiency.

In addition to addressing the 3D bin packing problem itself, this thesis aims to illustrate the broader **impact that such optimizations can have on the Italian transportation market**. Given Italy's dependence on efficient logistics for economic growth, optimized packing solutions have the potential to significantly reduce transportation costs, improve operational efficiency, and contribute to more sustainable business practices in the logistics sector.

A long-term vision of this thesis is that the developed solver could form the foundation for a **startup in the logistics technology sector**. Such a startup could leverage the solver to provide innovative, cost-effective solutions for companies seeking to optimize their supply chain operations. Demonstrating the scalability and practical viability of this approach could lead to significant business opportunities and contribute to the growing demand for advanced logistics optimization technologies.

Finally, this thesis will explore both traditional optimization approaches, such as heuristic and exact methods, alongside modern techniques. By applying advanced algorithms and evaluating their performance through real-world simulations, this research aims to contribute to the growing need for logistics solutions that reduce costs, improve efficiency, and minimize environmental impact.

1.4 Structure of the Thesis

This thesis is structured into seven main chapters, each addressing a different aspect of the research on the 3D BPP, from foundational concepts to practical implementation and future prospects.

- **Chapter 1** introduces the research problem, outlines the objectives of the thesis, and provides the motivation for the study, focusing on the potential impact of optimized logistics solutions in the transportation market..
- **Chapter 2** presents a comprehensive literature review, covering the evolution of the bin packing problem across 1D, 2D, and 3D versions. It explores key concepts in multi-objective optimization, traditional techniques for solving the 3D bin packing problem, and modern approaches such as reinforcement learning. The chapter also investigates problem reduction strategies and their relevance to solving broader optimization problems.
- **Chapter 3** addresses the computational complexity of the multi-objective 3D bin packing problem. It examines its NP-hard nature, explores complexity considerations in the presence of constraints such as load fragility and weight distribution to evaluate their impact on the problem's complexity.
- **Chapter 4** delves into heuristic concepts specifically designed for the 3D bin packing problem. It begins with an overview of heuristic approaches, followed by a detailed discussion of the heuristics implemented in this thesis. These include the constructive, improvement, and diversification phases, each addressing specific aspects of the packing process. The chapter also introduces priority-based item sorting strategies, methods for handling empty maximal spaces through generation, overlap resolution, and selection mechanisms. It concludes by explaining pivot point selection, emphasizing stability requirements and orientation optimization, and evaluates the advantages and challenges of the implemented heuristics.
- **Chapter 5** provides a comprehensive description of the algorithm proposed in this thesis. It outlines the constructive phase, focusing on initial packing strategies, and explains the improvement phase, which utilizes Variable Neighborhood Descent (VND) approach to refine solutions. Additionally, the diversification phase is introduced to handle unpacked items effectively. The chapter also discusses decision rules and implementation details critical for the algorithm's functionality.

- **Chapter 6** analyzes the experimental results obtained by the proposed algorithm. It describes the datasets used, evaluates performance metrics such as space utilization and runtime, and compares the algorithm's performance with existing approaches. The chapter terminates with a discussion of the results, highlighting the algorithm's strengths and weaknesses.
- **Chapter 7** summarizes the key contributions of the thesis, emphasizing the proposed heuristics and algorithm's practical implications. It also identifies potential improvements to the algorithm and suggests directions for future research to advance the state-of-the-art in solving the 3D bin packing problem.

Chapter 2

Literature Review

2.1 The Bin Packing Problem: 1D, 2D, and 3D versions

2.1.1 Traditional Bin Packing Problem

The Bin Packing Problem (BPP) is a classical combinatorial optimization problem that has been studied for over 80 years. Its origins trace back in 1939, when it was introduced by *L.V. Kantorovich* in [4] and since then, it has remained one of the most challenging and influential problems in the field. BPP models a wide range of real-world applications, from logistics [5] and resource allocation to scheduling and data storage optimization [6]. Over the years, researchers have expanded the basic model, studied more comprehensive versions, and developed new algorithmic approaches to improve performance and solution quality. The basic idea behind the Bin Packing Problem is to determine the most efficient way (that means using less space) to pack a set of items into a finite number of containers (or bins), ensuring a certain amount of constraints are respected; such as weight or space limits. The goal is typically to minimize the number of bins used, while ensuring that no bin exceeds its capacity. One of the first mathematical formulations of the problem was proposed in [4] and is reported as follows:

Given a set of n items, each with a non-negative integer weight w_i (where $i = 1, 2, \ldots, n$), and an unlimited number of bins, each with a capacity C, the objective is to assign all items to the minimum number of bins such that the total weight of the items in each bin does not exceed C.

This can be expressed mathematically through the following integer linear program (ILP):

$$\min \sum_{j=1}^{m} y_j \tag{2.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_{ij} \leq C y_j, \quad \forall j = 1, 2, \ldots, m \tag{2.2}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \forall i = 1, 2, \ldots, n \tag{2.3}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i = 1, 2, \ldots, n; \forall j = 1, 2, \ldots, m \tag{2.4}$$

$$y_j \in \{0, 1\}, \quad \forall j = 1, 2, \ldots, m \tag{2.5}$$

Where:

- x_{ij} is a binary decision variable that equals 1 if item i is placed in bin j, and 0 otherwise.
- y_j is a binary variable that equals 1 if bin j is used, and 0 otherwise.

The objective function minimizes the number of bins used, while the constraints ensure that each item is placed in exactly one bin, and that the total weight of items in any bin does not exceed its capacity C.

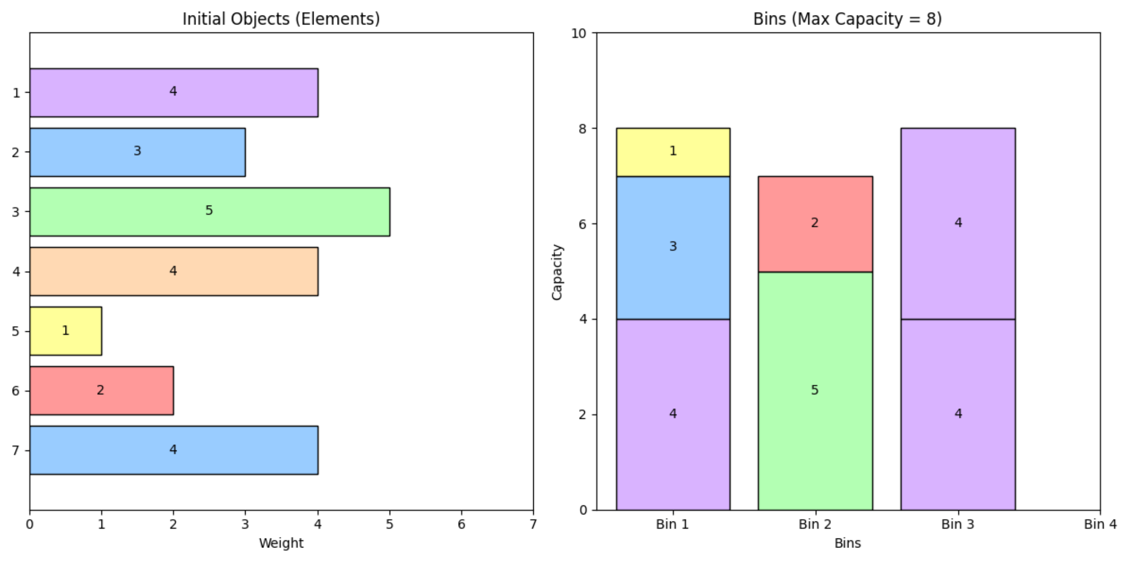

Figure 2.1. Visualization of the 1D Bin Packing Problem. The left panel illustrates the initial objects to be packed. The right panel displays the resulting bin allocation, where the objects have been packed into bins of maximum capacity 8.

2.1.2 The 2D Bin Packing Problem

The *Two-Dimensional Bin Packing Problem (2D BPP)* generalizes the 1D problem by introducing an additional dimension: item height and width. The goal is to pack a set of rectangular items into the minimum number of rectangular bins, ensuring that no item exceeds the bin's sizes and no two items overlap.

In the *finite* 2D Bin Packing Problem (2BP), we are given a finite set of n rectangular items $j \in J = \{1, 2, \ldots, n\}$, each item having a width w_j and height h_j, and an unlimited number of identical rectangular bins, each with a width W and height H. The objective is to allocate all the items into the fewest number of bins, ensuring that no two items overlap and their edges remain parallel to those of the bins. In this version of the problem, it is assumed that the items have a fixed orientation, meaning they cannot be rotated.

Initial formulations of two-dimensional bin packing were proposed by Gilmore and Gomory, who extended their column generation approach from 1BP to develop a model for 2BP [7,8]. Subsequent works, such as that of Beasley, have contributed integer linear programming (ILP) models for specific variations of the 2BP, such as the cutting stock problem, by using discrete coordinates to position items [9]. More recent approaches have introduced alternative modeling techniques, such as graph-theoretical representations by Fekete and Schepers, and ILP models developed by Lodi et al., which optimize solutions by packing items "by levels," significantly reducing the number of required variables and constraints [10,11].

The 2D BPP can be thus represented by eans of the following constraints:

$$\min \sum_{j=1}^{m} y_j \tag{2.6}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} (w_i x_{ij}) \leq W y_j, \quad \forall j = 1, 2, \ldots, m \tag{2.7}$$

$$\sum_{i=1}^{n} (h_i x_{ij}) \leq H y_j, \quad \forall j = 1, 2, \ldots, m \tag{2.8}$$

$$\text{Non-overlapping constraints: Ensure no two items overlap within any bin} \tag{2.9}$$

$$\text{Fixed orientation: Items cannot be rotated within the bins} \tag{2.10}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i = 1, 2, \ldots, n; \forall j = 1, 2, \ldots, m \tag{2.11}$$

$$y_j \in \{0, 1\}, \quad \forall j = 1, 2, \ldots, m \tag{2.12}$$

Where:

- x_{ij} is a binary decision variable that equals 1 if item i is placed in bin j, and 0 otherwise.
- y_j is a binary variable that equals 1 if bin j is used, and 0 otherwise.
- The width and height constraints ensure that the total width and height of items packed in each bin do not exceed the bin's sizes W and H, respectively.

One of the key challenges in 2D BPP is ensuring that items are packed without overlapping while respecting their fixed orientation. The non-overlapping constraint can be mathematically represented by ensuring that no two items in the same bin share any common area. For any two items i and k placed in the same bin, one of the following conditions must hold:

- Item i is placed entirely to the left or right of item k.
- Item i is placed entirely above or below item k.

These conditions ensure that items do not overlap in the bin.

In many applications, such as *cutting stock*, *sheet metal cutting*, and *cargo loading*, the 2D BPP offers a practical model to optimize resource utilization. Though the assumption of fixed orientation simplifies the problem, it may lead to suboptimal packing efficiency, especially when rotation could lead to better space utilization.

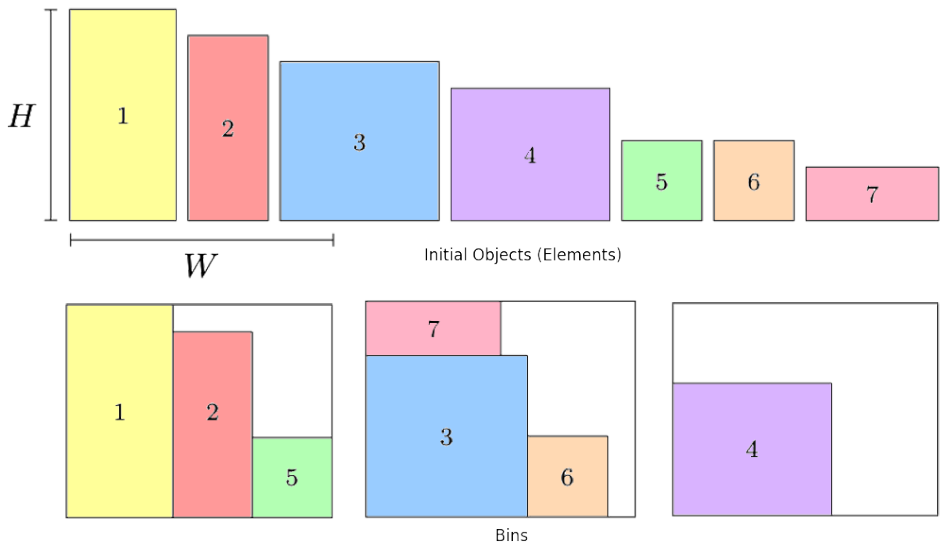

Figure 2.2. Visualization of the 2D Bin Packing Problem. The upper part of the image shows the elements to be packed with their width(W) and height(H). In the lower part is shown a finite bin solution found by FBSOG and FFFOG

2.1.3 The 3D Bin Packing Problem

As we already mentioned in the previous sections the Three-Dimensional Bin Packing Problem (3D BPP) is a problem with numerous practical applications, including *container loading*, *warehouse management*, and *logistics*. The problem involves packing a set of items into a finite number of bins with fixed dimensions, aiming to maximize space utilization or minimize the number of bins used. Compared to the 1D and 2D versions, the 3D BPP introduces additional complexities, such as spatial arrangement, orientation, and adherence to real-world constraints like weight distribution and item fragility. In the real world there are several case studies of this problem, for example as reported in [12]. We experimented this in the last months also in projects related to the startup mentioned before, where a client wanted to implement a software to optimize the loading of the trucks.

Several factors need to be considered in 3D BPP:

- **Item Rotation**: Allowing items to be rotated along one or more axes can significantly improve packing efficiency by enabling better utilization of bin space. However, this increases the complexity of the problem, as the number of feasible placements grows exponentially.
- **Item Fragility**: Certain items have constraints related to their fragility, requiring specific orientations or positions within the bin to avoid damage. For example, fragile items might need to be placed upright or isolated from heavier objects.
- **Weight Distribution**: Ensuring an even distribution of weight across the bin is critical in many logistics applications to prevent structural instability or issues during transportation.
- **Stacking Constraints**: Many practical applications require items to be stacked in specific orders, such as placing heavier items below lighter ones or avoiding stacking on fragile objects.

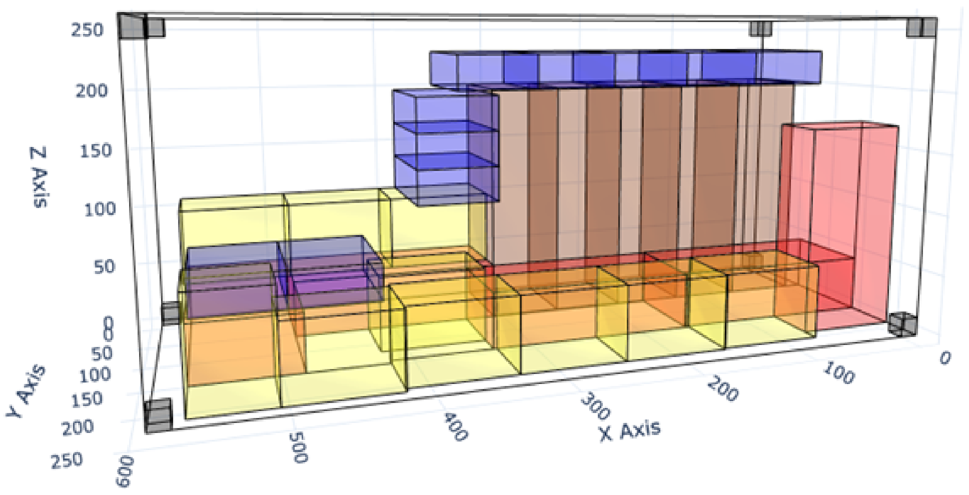

Figure 2.3. A viable output of the 3D bin packing algorithm tackled in the thesis.

2.1.4 Evolution of the 3D Bin Packing Problem

The 3D BPP has evolved significantly over the years, with early research focusing on theoretical formulations and exact solution methods. Early approaches modeled the problem as an extension of the 1D and 2D BPP, incorporating additional constraints to account for the three-dimensional nature of the items and bins. Initial mathematical models included constraints for geometric fit, non-overlap, and basic item orientation.

One of the first notable formulations was proposed by Martello et al. [13], who extended the branch-and-bound framework to handle the 3D case. The model included decision variables to represent the position and orientation of each item in the bin, along with constraints ensuring that no items overlap:

$$x_i + l_i \leq L, \quad y_i + w_i \leq W, \quad z_i + h_i \leq H, \quad \forall i \in I, \tag{2.13}$$

$$\text{Non-overlap: } (x_i \geq x_j + l_j) \vee (x_j \geq x_i + l_i) \vee \tag{2.14}$$

$$(y_i \geq y_j + w_j) \vee (y_j \geq y_i + w_i) \vee \tag{2.15}$$

$$(z_i \geq z_j + h_j) \vee (z_j \geq z_i + h_i), \quad \forall i, j \in I, \, i \neq j. \tag{2.16}$$

Here, x_i, y_i, z_i represent the coordinates of item i, and l_i, w_i, h_i are the length, width, and height of the item.

2.1.5 Formulations in Literature

The mathematical formulation of the 3D BPP typically includes:

- **Geometric Fit Constraints**: Ensure that the dimensions of each item fit within the bin, considering allowable rotations.
- **Non-Overlap Constraints**: Guarantee that no two items occupy the same space within the bin.
- **Weight Distribution Constraints**: Ensure even distribution of weight across the bin.
- **Stacking Order Constraints**: Enforce stacking rules, such as placing heavier items below lighter ones.

Bortfeldt and Wäscher [14] contributed a comprehensive review of exact and heuristic approaches, emphasizing the need for realistic models that account for stacking and weight constraints. They highlighted how ILP formulations evolved to include problem-specific constraints such as fragile item handling and complex weight balancing.

2.1.6 Notable Works in 3D Bin Packing

Several key papers have contributed significantly to the understanding and advancement of the 3D BPP. Below, we summarize some of the most influential works and their contributions:

- **Martello et al. :** The authors presented one of the earliest exact algorithms for the 3D BPP, using branch-and-bound techniques to find optimal solutions. Their work laid the foundation for mathematical modeling of the problem, incorporating constraints like item orientation and non-overlap. [13]

- **Crainic et al. :** The authors developed a hybrid heuristic combining tabu search with greedy algorithms. Their method addressed large-scale instances and emphasized balancing computational efficiency with solution quality. [15]

- **Bortfeldt and Wäscher :** This comprehensive review summarized existing algorithms for the 3D BPP and proposed a new benchmark dataset for evaluating algorithm performance. Their work highlighted the importance of considering real-world constraints like stacking and weight distribution. [14]

- **Gomes and Oliveira :** This paper introduced reinforcement learning for the 3D BPP, where agents learned to pack items through trial-and-error interactions with a simulation environment. Their work paved the way for combining artificial intelligence with traditional optimization techniques. [16]

2.1.7 Recent Advances and Challenges

Recent advancements in the field of 3DBPP have been driven by the need to incorporate real-world constraints and leverage modern computational techniques. One of the most significant developments in this area is the application of reinforcement learning (RL). RL-based methods enable agents to learn optimal packing strategies through trial-and-error interactions within simulated environments. This adaptive approach allows for flexibility in dynamic situations where the input parameters or constraints may change frequently. For example, Gomes and Oliveira [16] demonstrated how reinforcement learning could effectively tackle the 3D BPP by training agents to iteratively improve their packing strategies, which proved beneficial in complex and high-dimensional scenarios.

Deep learning has also emerged as a powerful tool for solving the 3D BPP, particularly for predicting efficient packing configurations. Neural networks, such as Convolutional Neural Networks (CNNs), have been utilized to analyze spatial relationships among items and suggest feasible arrangements. Graph Neural Networks (GNNs), in particular, have shown promise in modeling the connectivity and geometric properties of items, thereby improving the accuracy of placement predictions. These methods are often integrated with traditional optimization techniques to handle the combinatorial complexity of the problem while maintaining computational efficiency.

Hybrid algorithms have gained traction as they combine heuristic approaches with machine learning techniques to achieve a balance between solution quality and runtime efficiency. In many cases, heuristics are employed to generate an initial feasible solution, which is then refined using more sophisticated machine learning models. For instance, Zhang et al. [17] proposed a hybrid genetic algorithm that incorporates machine learning to enhance the crossover and mutation processes, leading to better exploration of the solution space and improved packing results.

Distributed and parallel computing frameworks have also been instrumental in advancing the scalability of 3D BPP solutions. These frameworks distribute the computational load across multiple processors, enabling the simultaneous evaluation of numerous packing configurations. By leveraging cloud-based systems and high-performance computing clusters, researchers have significantly reduced the time required to solve large-scale instances of the 3D BPP. This progress is particularly important in real-time applications, such as warehouse management and automated logistics, where rapid decision-making is crucial.

Despite these advancements, several challenges persist in the study of 3D BPP. One of the primary issues is the need to balance multiple objectives, such as minimizing the number of bins used while ensuring proper weight distribution and adhering to fragility constraints. Addressing these conflicting goals often requires sophisticated multi-objective optimization techniques. Furthermore, uncertainties in item dimensions, weights, or availability pose additional difficulties, particularly in dynamic environments where real-time adjustments are required. Another critical challenge lies in developing algorithms that are not only accurate but also scalable, capable of handling the high-dimensionality and complexity of real-world instances.

Future research should aim to further integrate heuristic methods with advanced machine learning techniques, creating hybrid models that capitalize on the strengths of both approaches. Additionally, incorporating domain-specific knowledge, such as industry-specific constraints or operational requirements, into algorithm design could lead to more robust and practical solutions. Exploring new areas, such as quantum computing and neuromorphic computing, may also provide breakthroughs in addressing the inherent complexity of the 3D BPP. These directions will likely enhance the applicability and effectiveness of solutions in diverse real-world scenarios, driving further progress in this challenging and impactful field. Additionally, incorporating domain-specific knowledge into algorithm design could further enhance the applicability of solutions in real-world scenarios.

2.2 Multi-Objective Optimization: Concepts and Methods

Multi-objective optimization (MOO) is a branch of mathematical optimization focused on solving problems that involve multiple, often conflicting objectives. Unlike single-objective optimization, where a single optimal solution is pursued, multi-objective optimization seeks a set of Pareto-optimal solutions, where no objective can be improved without degrading another. This concept is essential in practical scenarios where competing goals, such as cost, quality, and efficiency, must be balanced to find a feasible solution.

A generic multi-objective design optimization problem, as proposed by [18] is reported below:

$$
\begin{aligned}
&\min \mathbf{J}(\mathbf{x},\mathbf{p}) \\
&\text{s.t.} \quad \mathbf{g}(\mathbf{x},\mathbf{p}) \le 0, \\
&\qquad \mathbf{h}(\mathbf{x},\mathbf{p}) = 0, \\
&\qquad x_{i,\mathrm{LB}} \le x_i \le x_{i,\mathrm{UB}} \quad (i = 1,\ldots,n), \\
&\qquad \mathbf{x} \in S,
\end{aligned}
\tag{2.17}
$$

where

$$
\begin{aligned}
\mathbf{J} &= \begin{bmatrix} J_1(\mathbf{x}) & \cdots & J_z(\mathbf{x}) \end{bmatrix}^T, \\
\mathbf{x} &= \begin{bmatrix} x_1 & \cdots & x_i & \cdots & x_n \end{bmatrix}^T, \\
\mathbf{g} &= \begin{bmatrix} g_1(\mathbf{x}) & \cdots & g_{m_1}(\mathbf{x}) \end{bmatrix}^T, \\
\mathbf{h} &= \begin{bmatrix} h_1(\mathbf{x}) & \cdots & h_{m_2}(\mathbf{x}) \end{bmatrix}^T.
\end{aligned}
$$

In this formulation, $\mathbf{J}$ represents a vector of z objectives $J_i(\mathbf{x},\mathbf{p})$, each dependent on a vector of n design variables $\mathbf{x}$ and fixed parameters $\mathbf{p}$. The aim is to simultaneously minimize each objective in $\mathbf{J}$ within a feasible domain S defined by m_1 inequality constraints $g(\mathbf{x},\mathbf{p}) \le 0$ and m_2 equality constraints $h(\mathbf{x},\mathbf{p}) = 0$. The design variables x_i are bounded by upper and lower limits x_i^{UB} and x_i^{LB}, respectively, ensuring a solution space that respects both operational limitations and design freedoms.

In the context of the 3D bin packing problem, multi-objective optimization becomes particularly significant as it introduces constraints that align with real-world concerns, such as maximizing space utilization while minimizing load fragility and ensuring balanced weight distribution. These objectives are inherently conflicting, as maximizing space utilization might result in a configuration that does not adequately protect fragile items, or achieving a stable weight distribution might require extra bins, reducing space efficiency. MOO techniques allow for these competing criteria to be balanced, providing solutions that optimize one or more objectives without disregarding others.

2.2.1 Concepts in Multi-Objective Optimization

A fundamental concept in multi-objective optimization is the Pareto frontier, or Pareto-optimal set. The Pareto frontier represents solutions where any improvement in one objective results in the degradation of another, making these solutions equally valuable depending on the decision-maker's preferences. Selecting from the Pareto-optimal set involves trade-offs between objectives, which decision-makers must weigh based on the specific requirements of the problem. For instance, in bin packing applications for logistics, reducing fragility might be prioritized over minimizing bins when dealing with delicate items.

To navigate these trade-offs, decision-making approaches in MOO often rely on methods such as scalarization, where multiple objectives are transformed into a single-objective function using weighted sums, allowing for a traditional optimization approach. [19] Alternatively, methods like *epsilon-constraint* are used, wherein one objective is minimized while the others are transformed into constraints with specified limits. [20]

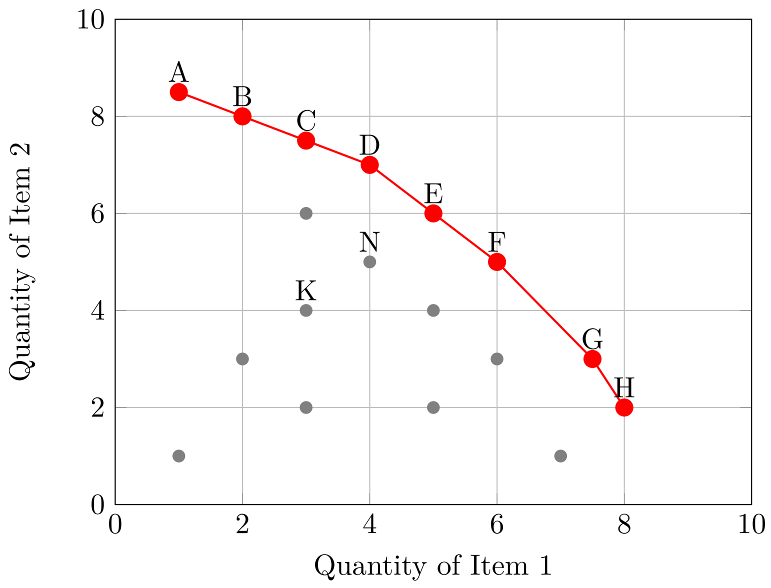

Figure 2.4. A Pareto frontier in multi-objective optimization. The red line represents Pareto-efficient solutions, where improving one objective requires sacrificing another. Red points are Pareto-optimal, while gray points, like N and K, are dominated by frontier solutions.

2.2.2 Scalarization Methods

Broadly, multi-objective optimization methods are categorized into two main approaches: scalarization and Pareto-based methods. Scalarization techniques convert multi-objective problems into a single-objective problem by aggregating individual objectives into an overarching scalar function. This transformation often relies on assigning weights to each objective based on their relative importance, a process referred to as apriori preference expression. This requires setting preferences or weights for each objective in advance, which shapes the final solution's outcome based on predetermined priorities [20, 21].

The weighted sum approach is one of the most common scalarization methods, combining each objective into a single function weighted by its relative importance. Other techniques include *Compromise Programming*, which uses non-linear combinations to achieve a balance between objectives, and *Lexicographic Approaches*, where objectives are ranked in order of importance and solved sequentially. Multiattribute Utility Analysis (MAUA), based on utility theory, integrates subjective preferences in decision-making, allowing for more nuanced prioritization of objectives. Physical programming and fuzzy logic methods, such as acceptability functions, introduce additional flexibility by allowing preferences to vary within acceptable bounds, thus handling real-world variability more effectively [22, 23].

Although scalarization methods provide a clear path to optimizing complex problems, they rely heavily on the assumption that decision-makers can accurately assign preferences beforehand. This may not always reflect real-world complexities, where priorities can shift based on the outcomes seen during optimization. Nonetheless, scalarization approaches are powerful for problems where objectives can be reasonably prioritized and quantified at the outset.

2.2.3 Pareto-Based Approaches

In contrast to scalarization, Pareto-based methods do not aggregate objectives into a single scalar function but instead treat each objective independently throughout the optimization process. These methods identify a set of Pareto-optimal solutions, where improving one objective cannot occur without degrading another, making these solutions equally optimal in terms of trade-offs. This is known as a-posteriori preference expression, as preferences are not explicitly defined until the solution set is generated.

One prominent Pareto-based method is *Exploration and Pareto Filtering*, which evaluates solutions based on their dominance in the objective space. Pareto Genetic Algorithms, like the Multi-Objective Genetic Algorithm (MOGA), are often used in this approach, leveraging natural selection to evolve a diverse set of Pareto-optimal solutions. Similarly, Multi-Objective Simulated Annealing (MOSA) applies iterative improvement to search for Pareto solutions, focusing on diverse solutions in the Pareto frontier rather than a single optimum [24, 25].

The Normal Boundary Intersection (NBI) method is another prominent Pareto approach, particularly effective in problems with convex Pareto fronts. NBI produces an evenly distributed set of Pareto solutions, enhancing the decision-maker's ability to view trade-offs comprehensively across the solution space [20]. These methods allow for adaptive exploration and weight adjustment during the optimization process, accommodating real-world complexities more naturally than scalarization.

2.2.4 Comparative Overview of Multi-Objective Methods

Table 2.1 summarizes various multi-objective methods, emphasizing the dichotomy between scalarization and Pareto approaches and their practical applications in optimization. Scalarization methods, while efficient for predefined preferences, may lack flexibility in dynamic environments. Pareto approaches, although computationally more demanding, are suited to applications where real-time preferences or trade-offs are critical.

Ultimately, both approaches aim to present decision-makers with a diverse set of "near-optimal" solutions, which can be selected based on additional qualitative or practical considerations not encapsulated in the original objectives. For this thesis, multi-objective optimization provides a fundamental approach to modeling and solving the 3D bin packing problem, reflecting the real-world dynamics and trade-offs of this problem within the Italian transportation and logistics sector.

Table 2.1. Overview of Multiobjective Optimization Methods

Scalarization Methods	**Pareto Methods**
Weighted Sum Approach	Exploration and Pareto Filtering
Compromise Programming	Weighted Sum Approach (with weight scanning)
Multiattribute Utility Analysis (MAUA)	Adaptive Weighted Sum Method (AWS)
Physical Programming	Normal Boundary Intersection (NBI)
Goal Programming	Multiobjective Genetic Algorithms (MOGA)
Lexicographic Approaches	Multiobjective Simulated Annealing (MOSA)
Acceptability Functions, Fuzzy Logic	

2.2.5 Methods for Multi-Objective Optimization

A variety of algorithms have been developed to handle MOO problems, each with unique strengths depending on the nature of the objectives and constraints. Evolutionary algorithms, such as the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) and Multi-Objective Particle Swarm Optimization (MOPSO), are commonly employed due to their ability to explore large search spaces and provide diverse Pareto-optimal solutions. NSGA-II, for instance, uses a ranking and crowding distance mechanism to maintain solution diversity and minimize dominance violations, making it particularly useful for combinatorial problems like 3D bin packing where feasible configurations are sparse [22, 24].

Genetic algorithms are also valuable in MOO, where they help approximate Pareto-optimal sets in complex spaces. The inherent flexibility of genetic algorithms allows for encoding multiple objectives into a fitness function, and their crossover and mutation operators facilitate diverse solution generation. Techniques such as Differential Evolution (DE) have also been adapted to multi-objective contexts, proving effective in navigating continuous and combinatorial solution spaces in industrial applications [22, 26].

In recent years, metaheuristic methods such as simulated annealing and tabu search have been adapted for multi-objective use. These methods leverage iterative improvement to search the solution space, allowing them to tackle the challenges of NP-hard problems like 3D bin packing. While slower than some heuristic approaches, they offer reliable convergence toward Pareto-optimal solutions when tuned properly [25].

Another class of methods, often referred to as decomposition-based approaches, breaks down multi-objective problems into simpler sub-problems. Methods such as Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) address each sub-problem individually and aggregate results to approximate the Pareto frontier. Decomposition is particularly suited to multi-objective bin packing because it allows optimization on different layers, such as packing efficiency and fragility constraints, before combining the results for a balanced solution [23].

These multi-objective optimization techniques have paved the way for flexible and efficient solutions in complex logistical scenarios.

Chapter 3

Computational Complexity and 3D Bin Packing

3.1 Computational Complexity: An Overview

Computational complexity is a fundamental aspect of theoretical computer science that deals with classifying computational problems based on the resources required for their solution. These resources primarily include time and space, measured as functions of the size of the input. For any given problem, time complexity quantifies the number of basic operations an algorithm must perform to reach a solution, while space complexity evaluates the memory usage during computation.

Formally, let $T(n)$ represent the time complexity of an algorithm, where n is the size of the input. An algorithm runs in polynomial time if there exists a constant k such that:

$$T(n) = O(n^k),$$

where $O(n^k)$ denotes the asymptotic upper bound, indicating that the algorithm's running time grows at most proportionally to n^k as $n \to \infty$. Polynomial time is opposed to exponential time $T(n) = O(2^n)$, which grows significantly faster and is generally impractical for large inputs.

Problems are grouped into complexity classes, such as **P** (problems solvable in polynomial time) and **NP** (nondeterministic polynomial time problems, where a solution can be verified in polynomial time). Among these, **NP-hard** problems are at least as difficult as the hardest problems in NP. If a problem is **NP-complete**, it implies that while a proposed solution can be verified efficiently, finding the solution itself is believed to require non-polynomial time in the general case.

To formally define **P**, let L be a language (a set of strings) and D a deterministic Turing machine. Then, $L \in \mathbf{P}$ if there exists a polynomial $p(n)$ such that for all $x \in L$, the Turing machine D decides x in $O(p(n))$ time. Similarly, for **NP**, a language L is in **NP** if there exists a nondeterministic Turing machine N and a polynomial $p(n)$ such that for all $x \in L$, there exists a certificate y (of size polynomial in n) that N can verify in $O(p(n))$ time.

The distinction between **P** and **NP** remains one of the most profound open questions in computer science, famously posed as the **P vs NP** problem. This question asks whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time. Despite significant effort, no proof has conclusively settled whether **P** equals **NP** or $\mathbf{P} \neq \mathbf{NP}$.

The implications of computational complexity extend beyond theoretical boundaries. Real-world applications include cryptographic systems, where security relies on the computational difficulty of problems such as integer factorization (believed to be in **NP**, but not **P**). Similarly, advancements in approximation algorithms aim to address NP-hard optimization problems by producing near-optimal solutions within reasonable time bounds.

Another important concept is the notion of **reductions**, which are used to relate the complexity of different problems. If problem A can be reduced to problem B in polynomial time, denoted $A \leq_p B$, solving B efficiently implies a solution to A. This property is central to classifying problems as NP-complete; a problem C is NP-complete if $C \in \mathbf{NP}$ and every problem in **NP** can be reduced to C in polynomial time.

The study of computational complexity provides valuable insights into algorithm design, computational feasibility, and the inherent difficulty of problems. By understanding the boundaries of what can be computed efficiently, researchers can focus on developing innovative techniques for tackling computational challenges in both theory and practice.

3.2 NP-hard nature of the 3D bin packing problem

The 3D-BPP is a natural extension of the 1D and 2D bin packing problems, both of which are known to be NP-hard. Establishing the NP-hardness of the 3D variant is critical for understanding its computational limitations.

Formal Problem Definition

The 3D bin packing problem can be formally defined as follows:

Input:

- A set $I = \{1, 2, \ldots, n\}$ of n items, where each item i is characterized by its dimensions (w_i, h_i, d_i): width, height, and depth, respectively.
- A set of bins $J = \{1, 2, \ldots, m\}$, each with fixed sizes (W, H, D): width, height, and depth, respectively.

Variables:

- $x_{ijk} \in \{0, 1\}$: A binary variable that equals 1 if item i is placed in bin j with orientation k, and 0 otherwise.
- $y_j \in \{0, 1\}$: A binary variable that equals 1 if bin j is used, and 0 otherwise.

Objective: Minimize the total number of bins used:

$$\min \sum_{j=1}^{m} y_j$$

Constraints:

1. **Item assignment:** Each item must be placed in exactly one bin and assigned one orientation:
$$\sum_{j=1}^{m} \sum_{k=1}^{o} x_{ijk} = 1, \quad \forall i \in I$$
where o is the number of allowed orientations for each item.

2. **Non-overlapping within bins:** Items placed in the same bin must not overlap. For any two distinct items i and i' assigned to bin j, at least one of the following must hold:
$$x_{ijk}(x_i^{\max} + w_i) \le x_{i'}^{\min} \quad \text{(no overlap in width)}, \tag{3.1}$$
$$x_{ijk}(y_i^{\max} + h_i) \le y_{i'}^{\min} \quad \text{(no overlap in height)}, \tag{3.2}$$
$$x_{ijk}(z_i^{\max} + d_i) \le z_{i'}^{\min} \quad \text{(no overlap in depth)}. \tag{3.3}$$

3. **Bin sizes:** The total dimensions of items in each bin must not exceed the bin sizes:
$$\sum_{i \in I} x_{ijk} \cdot w_i \le W, \quad \sum_{i \in I} x_{ijk} \cdot h_i \le H, \quad \sum_{i \in I} x_{ijk} \cdot d_i \le D, \quad \forall j \in J.$$

Proof of NP-Hardness

The 3D bin packing problem (3BP) is *NP-hard* as it generalizes the 2D bin packing problem (2BP), which is already known to be *NP-hard* [11]. More precisely, since 2BP is a restricted version of 3BP where the third dimension is always zero, any instance of 2BP can be trivially transformed into an instance of 3BP without altering its computational complexity.

A common approach in complexity theory is to show that a problem remains *NP-hard* even when restricted to a lower-dimensional case. In this context, it has been established that 1D bin packing (1BP) is strongly *NP-hard*, which implies that 2BP inherits this complexity, as noted in [11]. By the same reasoning, since 3BP extends 2BP by introducing an additional dimension, it must also be *NP-hard*. This observation eliminates the need for a direct reduction from another *NP-hard* problem, as the complexity follows naturally from the generalization.

Therefore, instead of constructing a separate proof via a reduction , it is sufficient to note that 3BP generalizes 2BP and thus retains its *NP-hardness*.

Implications of NP-Hardness

The NP-hard nature of the 3D bin packing problem implies that finding an exact solution is computationally infeasible for large instances. This necessitates the use of alternative approaches, such as:

- **Approximation algorithms:** These provide solutions with guaranteed bounds on their optimality.
- **Heuristics:** Strategies such as greedy algorithms, genetic algorithms, or simulated annealing, which aim to find near-optimal solutions efficiently.
- **Integer programming relaxations:** Relaxing constraints to linear or quadratic forms for faster computation.

Understanding the computational limits of the 3D bin packing problem is crucial for designing practical and scalable solutions in applications such as logistics, manufacturing, and resource allocation.

In real-world applications, the 3D bin packing problem often involves multiple objectives, such as:

- Minimizing the number of bins used (f_1).
- Minimizing the total packing height or volume (f_2).
- Balancing weight distribution across bins (f_3).

These additional objectives significantly increase the problem's complexity, as the solution space must be evaluated across multiple dimensions. Let $\mathcal{X}$ denote the set of all feasible solutions. For a given solution $x \in \mathcal{X}$, the objective vector can be defined as $\mathbf{f}(x) = (f_1(x), f_2(x), f_3(x))$, where each $f_i(x)$ represents a specific objective function.

Multi-Objective Optimization and Pareto Optimality

Multi-objective optimization introduces the concept of Pareto optimality. A solution $x^* \in \mathcal{X}$ is Pareto optimal if there does not exist another solution $x \in \mathcal{X}$ such that:

$$\forall i \in \{1,2,3\},\ f_i(x) \leq f_i(x^*) \text{ and } \exists j \in \{1,2,3\},\ f_j(x) < f_j(x^*). \tag{3.4}$$

The set of all Pareto optimal solutions forms the Pareto front, a curve or surface in the objective space that represents trade-offs between competing objectives. Constructing the Pareto front involves evaluating an exponentially large set of potential solutions, as the number of feasible configurations grows combinatorially with the number of items, bins, and dimensions.

Pareto Front Computation

Constructing the Pareto front efficiently requires advanced techniques due to the problem's inherent computational complexity. Two common approaches include:

- **Scalarization**: Transforming the multi-objective problem into a single-objective problem by combining objectives into a weighted sum:

$$f_{\text{scalarized}}(x) = \sum_{i=1}^{3} w_i f_i(x), \tag{3.5}$$

  where w_i are user-defined weights reflecting the importance of each objective. However, scalarization is sensitive to the choice of weights and may not capture all Pareto optimal solutions.

- **Evolutionary Algorithms (EAs)**: Algorithms such as the Non-dominated Sorting Genetic Algorithm II (NSGA-II) or the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) maintain a population of solutions and iteratively refine them to approach the Pareto front. These methods are stochastic and capable of exploring diverse regions of the solution space, but they often require extensive computational resources.

Complexity Analysis

The computational complexity of these methods can be expressed in terms of the number of objectives, m, and the size of the solution space, $|\mathcal{X}|$. For example:

- Scalarization methods typically require $O(|\mathcal{X}|)$ evaluations per weight configuration, and a dense sampling of the weight space increases computational demands.

- Evolutionary algorithms, with a population size of P and G generations, require $O(P \cdot G \cdot |\mathcal{X}|)$ evaluations to converge, where $P \cdot G$ often scales with m and the problem size.

3.3 Computational complexity of Heuristics approaches

First Fit Decreasing (FFD)

The First Fit Decreasing (FFD) heuristic is one of the most widely used methods for 3DBPP due to its simplicity and efficiency. The algorithm operates in two stages. First, all boxes are sorted in non-increasing order of their volumes, $v_i = w_i h_i d_i$. Then, each box is placed into the first bin where it fits. The steps of the FFD heuristic can be described formally as follows:

1. Sort the set of boxes $\mathcal{B}$ in descending order of their volumes.

2. Initialize an empty set of bins $\mathcal{C}$.

3. For each box $b_i \in \mathcal{B}$, place it into the first bin $C_j \in \mathcal{C}$ where it fits. If no such bin exists, create a new bin and add b_i to it.

The computational complexity of FFD arises from two primary operations: sorting the boxes, which has a complexity of $\mathcal{O}(n \log n)$, and placing the boxes into bins, which involves checking up to m bins for each of the n boxes, leading to a complexity of $\mathcal{O}(nm)$. Thus, the overall complexity is $\mathcal{O}(n \log n + nm)$. Here, m represents the number of bins used, which is typically much smaller than n in practical scenarios.

In terms of performance, FFD provides an approximation guarantee. Let OPT denote the optimal number of bins required. For the 1D and 2D variants of bin packing, it is known that:

$$\text{FFD}(\mathcal{B}) \leq \frac{3}{2} OPT + 2.$$

While exact bounds for the 3D case are more complex due to spatial constraints, FFD remains a practical choice for many applications.

Best Fit Decreasing (BFD)

The Best Fit Decreasing (BFD) heuristic improves upon FFD by placing each box into the bin where it leaves the least remaining space, thereby attempting to utilize bin space more efficiently. The algorithm proceeds similarly to FFD, with the primary difference being the selection criterion for placing boxes. Specifically, for each box b_i, BFD evaluates all bins where b_i fits and selects the bin that minimizes the unused space after placement.

The computational complexity of BFD is similar to that of FFD. Sorting the boxes requires $\mathcal{O}(n \log n)$, and placing each box involves scanning up to m bins, leading to a placement complexity of $\mathcal{O}(nm)$. Thus, the overall complexity is $\mathcal{O}(n \log n + nm)$.

Empirical studies suggest that BFD often outperforms FFD in terms of bin utilization, particularly for instances with diverse box dimensions. However, the improvement comes at the cost of slightly increased computational overhead during placement.

Hybrid Genetic Algorithms (HGA)

Hybrid Genetic Algorithms (HGAs) represent a class of metaheuristic approaches that combine evolutionary strategies with local search heuristics to tackle 3DBPP. These algorithms maintain a population of candidate solutions, which evolve over successive generations through genetic operations such as selection, crossover, and mutation. Additionally, local search techniques are employed to refine solutions within each generation, enhancing the overall quality of the results.

The computational complexity of HGA depends on several factors:

- The population size p.
- The number of generations g.
- The cost of evaluating the fitness of each solution, which is proportional to $\mathcal{O}(nm)$.

Combining these factors, the overall complexity of HGA is $\mathcal{O}(pgnm)$. While this complexity is significantly higher than that of simpler heuristics like FFD and BFD, HGAs often achieve superior solutions, particularly for large and complex instances. The trade-off between computational cost and solution quality makes HGAs suitable for applications where computational resources are not a limiting factor.

Comparison of Heuristics

To provide a comprehensive comparison of the discussed heuristics, we summarize their computational complexities and performance characteristics. FFD and BFD are straightforward to implement and have polynomial-time complexities, making them suitable for real-time applications. However, their performance may not suffice for instances requiring high-quality solutions. On the other hand, metaheuristics like HGA offer greater flexibility and solution quality but are computationally expensive and less scalable.

We summarize the complexities and performance in Table 3.1.

Heuristic	**Complexity**	**Scalability**
FFD	$\mathcal{O}(n \log n + nm)$	Moderate
BFD	$\mathcal{O}(n \log n + nm)$	Moderate
HGA	$\mathcal{O}(pgnm)$	Low
GRASP	$\mathcal{O}(knm)$	Low
BnB	Exponential	Low
SA	Problem-Dependent	Moderate

Table 3.1. Comparison of 3DBPP Heuristics.

Chapter 4

Heuristic Concepts for 3D Bin Packing

Since many complex problems requires solution in a reasonable amount of time, it is crucial to find good solutions not so far from the optimum that could be reached with an exact method, that can be computed in shorter time. The field of applicability of 3D bin packing are wide, and the time in wich a solution is required varies in each of them. We will focus from now on to the sector of logistics operations, carried out in many small medium enterprises around Italy. The choice of this target is not random, but is supported by the pratical utility of this thesis in future work that aims to optimize this kind of operation in several companies. Typically this logistics operators need to load a lot of parcels or pallets, stored in their depots, in a short time interval that could be generalized around 6 hours. Since the number of trucks that have to be loaded is in the range 100-400 is crucial to have a good solution in a very short time interval that we can estimate around 1-2 minutes. In this chapter we will try to analyze the concepts and methods of heuristic approaches for 3D Bin Packing.

4.1 Overview of Heuristic Approaches

The study of heuristics for 3D bin packing has its roots in the broader field of combinatorial optimization. Traditional approaches to the one-dimensional bin packing problem, such as First Fit and Best Fit, provided the initial framework for tackling multi-dimensional extensions. These algorithms rely on a greedy principle, where each item is placed in the first available bin that satisfies its constraints. While efficient, these methods often produce suboptimal solutions because they focus on immediate benefits without considering long-term spatial utilization.

Despite their simplicity, First Fit Decreasing (FFD) and Best Fit Decreasing (BFD) have shown strong performance for the 1D Bin Packing Problem 1D-BP, as highlighted by Martello and Toth [27]. Their extension to 3D bin packing appears promising but introduces significant challenges. In the 1D-BP case, sorting and bin selection are straightforward, as both items and bins are characterized by a single attribute: volume. In

3D-BP, however, additional dimensions—width, height, and depth—complicate the process. Sorting can no longer rely solely on volume, as the orientation of items and their faces also play a critical role in determining feasible placements.

Furthermore, the definition of the "best" bin in the BFD heuristic becomes ambiguous in 3D-BP. The spatial arrangement of items within a bin can vary significantly depending on how items are placed, even when their ordering and the best-bin selection rules remain unchanged. This dependency necessitates the development of more sophisticated placement strategies. The transition to two-dimensional and three-dimensional packing introduced additional complexities, requiring more sophisticated heuristics. Early works by Martello et al. [13] introduced the concept of corner points as candidate locations for placement, laying the groundwork for the development of spatially aware heuristics. Later, Crainic et al. [15] extended this idea to maximal spaces, which represent the largest contiguous empty regions within a bin. These concepts revolutionized the field by enabling more efficient space management, a critical factor in multi-dimensional packing.

The development of metaheuristic frameworks like GRASP (Greedy Randomized Adaptive Search Procedure) and VND (Variable Neighborhood Descent) further advanced the field. GRASP, introduced by Feo and Resende [28], combines greedy construction with randomized exploration to diversify the solution space. VND, proposed by Hansen and Mladenović [29], complements this by systematically exploring multiple neighborhoods of a solution, facilitating the escape from local optima.

These heuristic techniques have been adapted to a wide range of applications, from container loading in logistics to warehouse management and product packaging. Their flexibility and efficiency make them essential for addressing the time-sensitive demands of real-world operations.

4.1.1 First Fit Decreasing (FFD)

The *First Fit Decreasing* (FFD) heuristic is a widely used algorithm for solving the 3D-BPP. It follows a greedy approach where items are sorted in descending order of size before being placed into the first available bin that can accommodate them. This heuristic aims to minimize the number of bins used while maintaining computational efficiency.

The FFD algorithm operates in two main phases:

1. **Sorting Phase:** Items are sorted in decreasing order based on a predefined criterion, typically the largest dimension or volume.

2. **Packing Phase:** Items are placed sequentially into the first available bin that can accommodate them while respecting the bin's capacity constraints.

This strategy ensures that larger items, which are more difficult to place, are assigned first, reducing fragmentation and improving space utilization. The following pseudocode outlines the FFD working principle.

Algorithm 1 First Fit Decreasing (FFD) for 3D Bin Packing

```
Require: List of items $I = \{i_1, i_2, \dots, i_n\}$ with dimensions $(w_i, h_i, d_i)$, bin capacity
    $(W, H, D)$
Ensure: Packed bins minimizing the total count
0: Sort items in descending order based on volume $V_i = w_i \times h_i \times d_i$
0: Initialize empty list of bins $B = \{b_1, b_2, \dots\}$
0: for each item $i \in I$ do
0:    placed ← False
0:    for each bin $b \in B$ do
0:       if item $i$ fits in bin $b$ at any valid position then
0:          Place $i$ in $b$
0:          placed ← True
0:          Break
0:       end if
0:    end for
0:    if placed = False then
0:       Open a new bin $b_{new}$
0:       Place $i$ in $b_{new}$
0:       Add $b_{new}$ to $B$
0:    end if
0: end for
0: return $B$ (final packing solution) =0
```

The FFD heuristic provides a balance between computational efficiency and solution quality. While it does not guarantee an optimal packing, it is a widely used heuristic method in practical applications due to its simplicity and effectiveness. The algorithm performs well when combined with additional space optimization techniques such as *bottom-left* (BL) or *extreme points* (EP) heuristics.

4.1.2 Best Fit Decreasing (BFD)

The *Best Fit Decreasing* (BFD) heuristic is a variation of the *First Fit Decreasing* (FFD) algorithm, designed to improve space utilization by placing items in the bin where they fit best. Like FFD, BFD first sorts items in decreasing order before placement, but instead of choosing the first available bin, it selects the bin that results in the least amount of remaining free space after placement.

The BFD algorithm follows these main steps:

1. **Sorting Phase:** Items are sorted in descending order based on a predefined criterion, typically the largest dimension or volume.

2. **Packing Phase:** Each item is placed in the bin where it leaves the least residual space after placement, reducing fragmentation and maximizing packing density.

By prioritizing the bin that minimizes unused space, BFD can often yield better packing efficiency than FFD. However, this comes at the cost of slightly higher computational complexity.

The following pseudocode outlines the BFD heuristic for 3D BPP

Algorithm 2 Best Fit Decreasing (BFD) for 3D Bin Packing

```
Require: List of items $I = \{i_1, i_2, \dots, i_n\}$ with dimensions $(w_i, h_i, d_i)$, bin capacity
    $(W, H, D)$
Ensure: Packed bins minimizing the total count
0: Sort items in descending order based on volume $V_i = w_i \times h_i \times d_i$
0: Initialize empty list of bins $B = \{b_1, b_2, \dots\}$
0: for each item $i \in I$ do
0:   best_bin ← None
0:   min_remaining_space ← $\infty$
0:   for each bin $b \in B$ do
0:     if item $i$ fits in bin $b$ at any valid position then
0:       Compute remaining space in $b$ after placing $i$
0:       if remaining space < min_remaining_space then
0:         best_bin ← $b$
0:         min_remaining_space ← updated remaining space
0:       end if
0:     end if
0:   end for
0:   if best_bin is not None then
0:     Place $i$ in best_bin
0:   else
0:     Open a new bin $b_{new}$
0:     Place $i$ in $b_{new}$
0:     Add $b_{new}$ to $B$
0:   end if
0: end for
0: return $B$ (final packing solution) =0
```

The BFD heuristic improves upon FFD by selecting the most space-efficient bin for each item, thereby reducing wasted space. While this increases computational effort slightly due to the additional comparison steps, it often leads to a lower total bin count compared to FFD. BFD is particularly useful in applications where space optimization is a priority, such as warehouse storage or logistics.

4.1.3 Bottom-Left (BL) and Bottom-Left-Fill (BLF)

The *Bottom-Left* (BL) and *Bottom-Left-Fill* (BLF) heuristics focus on efficiently positioning items inside a bin to minimize wasted space while ensuring a stable packing structure.

Both BL and BLF heuristics follow a similar principle: items are placed as close as possible to the bottom of the bin first, then as close to the left, and finally towards the back. However, they differ in their placement refinement:

- **Bottom-Left (BL)**: Places each item at the lowest available position and as far left as possible in the bin. It does not attempt to optimize horizontal placement beyond this simple rule.

- **Bottom-Left-Fill (BLF)**: An extension of BL that attempts to further fill horizontal gaps by considering alternative leftmost positions before moving an item to a higher level.

These heuristics improve space utilization compared to simple greedy algorithms but do not guarantee an optimal solution. Their efficiency depends on how items are sorted before placement. A common approach is to sort items in decreasing order of volume or height.

The following pseudocode outlines the Bottom-Left (BL) heuristic:

Algorithm 3 Bottom-Left (BL) for 3D Bin Packing

```
Require: List of items $I = \{i_1, i_2, \ldots, i_n\}$ with dimensions $(w_i, h_i, d_i)$, bin sizes
    $(W, H, D)$
Ensure: Packed bins minimizing empty space
0: Sort items in descending order based on height or volume
0: Initialize empty bin $B$
0: for each item $i \in I$ do
0:    Find the lowest available position in the bin where $i$ fits
0:    Place $i$ at the lowest possible $z$-coordinate
0:    Adjust $x$ and $y$ coordinates to be as close to the left and back walls as possible
0: end for
0: return $B$ (final packing solution) =0
```

The Bottom-Left-Fill (BLF) heuristic refines this strategy by ensuring better horizontal filling before stacking items higher. Its pseudocode is as follows:

Algorithm 4 Bottom-Left-Fill (BLF) for 3D Bin Packing

```
Require: List of items $I = \{i_1, i_2, \ldots, i_n\}$ with dimensions $(w_i, h_i, d_i)$, bin dimensions
    $(W, H, D)$
Ensure: Packed bins minimizing empty space
0: Sort items in descending order based on height or volume
0: Initialize empty bin $B$
0: for each item $i \in I$ do
0:    Find the lowest available position in the bin where $i$ fits
0:    Identify alternative leftmost positions at the same height
0:    Choose the position that maximizes space utilization
0:    Place $i$ at the best position found
0: end for
0: return $B$ (final packing solution) =0
```

The BL and BLF heuristics provide an efficient way to arrange items within a bin while maintaining a stable structure. These methods are particularly useful in practical applications such as warehouse storage and logistics. The computational complexity depends on the number of placement checks performed per item, typically ranging from $\mathcal{O}(n^2)$ to $\mathcal{O}(n^3)$ in worst-case scenarios.

BLF tends to outperform BL in space utilization by reducing gaps between items. However, neither heuristic guarantees an optimal packing configuration. These heuristics are often combined with other optimization techniques to further enhance packing efficiency.

4.1.4 Metaheuristic Approaches

Metaheuristic algorithms [25] are advanced optimization techniques that provide near-optimal solutions to complex combinatorial problems such as the 3D-BPP. These approaches explore the solution space efficiently by balancing exploration (diversification) and exploitation (intensification).

Simulated Annealing (SA) [30] is a probabilistic optimization method inspired by the annealing process in metallurgy, where a material is gradually cooled to reach a stable state. It is widely applied in 3D-BPP to refine initial packing solutions generated by heuristic methods.

SA starts with an initial feasible solution and iteratively modifies it by making small perturbations (e.g., swapping items between bins or repositioning items within a bin). If the new solution improves the objective function (e.g., reduces the number of bins used), it is accepted. Otherwise, it is accepted with a probability that decreases over time, allowing the algorithm to escape local optima.

The following paragraph, describes in detail the SA-based algorithm used in this thesis to compare results obtained by our algorithm. The algorithm begins with an initial solution generated using the First-Fit Decreasing (FFD) heuristic. The initial temperature T is set to 500.0 and gradually decreases using an exponential cooling schedule:

$$T_{k+1} = \alpha T_k, \tag{4.1}$$

where $\alpha = 0.99$ is the cooling rate. The process continues until T reaches a minimum threshold $T_{\min} = 0.1$ or the maximum number of iterations (1000) is reached.

A key aspect of SA is the generation of neighboring solutions. The proposed algorithm employs enhanced moves including:

- Multi-item swaps between bins,
- Block movement to optimize space utilization,
- Removal and reinsertion of inefficiently packed items.

The acceptance probability P of a new solution with score S_{new} compared to the current solution with score $S_{current}$ is computed using an adaptive criterion:

$$P = \exp\left(\frac{S_{current} - S_{new}}{T}\right), \tag{4.2}$$

where worse solutions (i.e., those with $S_{new} > S_{current}$) may still be accepted with diminishing probability as T decreases.

The SA-based 3D Bin Packing procedure follows these steps:

1. Generate an initial solution using FFD.
2. Set initial temperature T and parameters.
3. While $T > T_{\text{min}}$ and iteration count is below the limit:
   (a) Generate a neighboring solution using improved heuristics.
   (b) Compute its score and compare with the current solution.
   (c) Accept the new solution probabilistically based on P.
   (d) Update the best solution found so far if necessary.
   (e) Reduce the temperature according to the cooling schedule.
4. Return the best solution found.

To improve convergence and escape poor local minima, the neighbor generation step incorporates:

- Targeted Item Relocation: Selecting inefficiently placed items and attempting to reposition them within other bins.
- Bin Pairing and Merging: Combining bins with low utilization to optimize space.
- Block-Based Adjustments: Moving multiple items together to improve packing density.

At the end of the optimization process, the best solution encountered throughout the annealing iterations is selected as the final output. The effectiveness of SA in this context depends on the balance between exploration (accepting worse solutions early on) and exploitation (converging to a refined solution as temperature decreases).

Overall, this SA-based approach introduces a structured way to refine bin packing solutions, improving space utilization and reducing wasted volume compared to baseline heuristics such as FFD alone.

Algorithm 5 Simulated Annealing for 3D Bin Packing

```
Require: Initial solution $S$, initial temperature $T$, cooling rate $\alpha$, stopping criteria
Ensure: Optimized bin packing configuration
0: Initialize $S_{best} \leftarrow S$
0: while stopping criteria not met do
0:     Generate a neighboring solution $S'$ by slightly modifying $S$
0:     Compute cost difference $\Delta C = C(S') - C(S)$
0:     if $\Delta C < 0$ or $e^{-\Delta C/T} >$ random(0,1) then
0:         Accept $S'$ as the new solution
0:         if $C(S') < C(S_{best})$ then
0:             Update $S_{best} \leftarrow S'$
0:         end if
0:     end if
0:     Update temperature $T \leftarrow \alpha T$
0: end while
0: return $S_{best}$ (final optimized solution) =0
```

Genetic Algorithms (GA) [31] are evolutionary computation techniques inspired by natural selection. In the context of 3D-BPP, GA maintains a population of packing solutions and evolves them over multiple generations using selection, crossover, and mutation operators.

The GA framework follows these steps:

1. **Initialization:** Generate an initial population of feasible packing solutions.

2. **Selection:** Choose the fittest solutions based on an objective function (e.g., minimizing the number of bins used).

3. **Crossover:** Combine features of parent solutions to create offspring.

4. **Mutation:** Introduce small modifications in offspring to enhance diversity.

5. **Termination:** Repeat until a convergence criterion is met (e.g., no improvement for a certain number of generations).

Algorithm 6 Genetic Algorithm for 3D Bin Packing

```
Require: Population size $P$, crossover rate $C_r$, mutation rate $M_r$, generations $G$
Ensure: Optimized packing solution
0: Initialize population $P_0$ with feasible solutions
0: for generation $g = 1$ to $G$ do
0:    Evaluate fitness of each solution
0:    Select parents using tournament or roulette selection
0:    Apply crossover with probability $C_r$ to generate offspring
0:    Apply mutation with probability $M_r$ to introduce variation
0:    Form the new population $P_{g+1}$ by selecting the best individuals
0: end for
0: return Best solution found =0
```

Tabu Search (TS) [32] is a memory-based optimization technique that iteratively explores the solution space while maintaining a list of previously visited solutions (the "tabu list") to prevent cycling and enhance exploration.

TS starts from an initial solution and iteratively explores the neighborhood by making local moves (e.g., repositioning or swapping items). If a move improves the solution, it is accepted. Otherwise, it is only accepted if it is not in the tabu list. The tabu list is updated dynamically to ensure diversification.

Algorithm 7 Tabu Search for 3D Bin Packing

```
Require: Initial solution $S$, tabu list size $L$, iterations $I$
Ensure: Optimized packing solution
0: Initialize $S_{best} \leftarrow S$
0: Initialize empty tabu list
0: for iteration $i = 1$ to $I$ do
0:    Generate neighborhood solutions $N(S)$
0:    Select the best solution $S'$ from $N(S)$ not in tabu list
0:    Update tabu list with recent moves
0:    if $C(S') < C(S_{best})$ then
0:       Update $S_{best} \leftarrow S'$
0:    end if
0: end for
0: return $S_{best}$ (final optimized solution) =0
```

Particle Swarm Optimization (PSO) [33] is a bio-inspired algorithm based on the collective movement of particles in a search space. In 3D-BPP, PSO models packing solutions as particles that move toward optimal configurations based on both their individual and collective best positions.

Each particle (solution) updates its velocity and position based on:

- Its own best position found so far.
- The best position found by any particle in the swarm.

Algorithm 8 Particle Swarm Optimization (PSO) for 3D Bin Packing

```
Require: Population size $P$, inertia weight $w$, cognitive and social factors $c_1, c_2$
Ensure: Optimized packing solution
0: Initialize particle positions and velocities
0: Evaluate fitness of each particle
0: Identify personal and global best solutions
0: for each iteration do
0:   for each particle do
0:     Update velocity based on inertia, cognitive, and social terms
0:     Update position
0:     Evaluate fitness and update best positions
0:   end for
0: end for
0: return Best solution found =0
```

Comparison and Hybrid Approaches

Metaheuristic approaches offer significant advantages in terms of flexibility and performance. However, no single method is universally superior. Hybrid techniques, such as combining *Genetic Algorithms* with *Simulated Annealing* or *Tabu Search*, often yield superior results by leveraging the strengths of multiple approaches.

4.1.5 Implemented Heuristics in the Thesis

The heuristic algorithm developed in this thesis for solving the 3D Bin Packing Problem (3DBPP) is built upon well-established principles of combinatorial optimization and spatial reasoning. The algorithm is structured into three distinct phases: constructive, improvement, and diversification. Each phase addresses specific challenges and integrates advanced concepts, including maximal spaces, pivot points, and metaheuristics such as Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Descent (VND).

The constructive phase is responsible for generating an initial feasible packing solution by placing items into bins while optimizing space utilization. It employs maximal space strategies to ensure efficient item placement and minimize wasted bin volume.

The improvement phase refines the initial solution by iteratively adjusting the item placements within the bins. This phase incorporates neighborhood-based heuristics from the VND framework to systematically explore different solution variations, focusing on local optimizations to reduce the number of bins used and enhance packing efficiency.

The diversification phase ensures that the algorithm does not become trapped in local optima by introducing controlled perturbations to the solution space. This phase leverages randomized mechanisms to reintroduce variability in the packing arrangement, enabling the discovery of alternative, potentially superior configurations.

A more detailed description of each phase, including the preprocessing steps, specific heuristic techniques, and the stopping criteria, is presented in Chapter 5. The next chapter provides a comprehensive breakdown of the algorithm's functionality, highlighting key innovations and contributions that enhance its performance and applicability to real-world bin packing scenarios.

4.2 Priority-Based Item Sorting

Sorting items before their placement plays a pivotal role in the efficiency of heuristic packing algorithms for the 3DBPP. The fundamental idea of prioritizing items by specific attributes originated from early bin packing strategies, such as the Next Fit Decreasing Height (NFDH) heuristic [34]. This heuristic focuses on sorting items by decreasing height to minimize fragmentation and optimize packing density. In the context of 3DBP, this principle has evolved to encompass more complex criteria, leveraging multiple sizes of the problem.

In this thesis, item sorting is a multi-criteria process designed to address the intricacies of 3D packing. The primary sorting metric is the volume of the items ($v_i = w_i \cdot h_i \cdot d_i$), which ensures that larger items, likely to create unfillable gaps if deferred, are tackled first. Sorting by volume reduces fragmentation, particularly in cases where smaller items cannot efficiently occupy the residual spaces left by larger ones. However, volume alone does not capture the full complexity of the packing problem.

Secondary criteria include load-bearing capacity and priority levels. Load-bearing capacity is critical in scenarios where stability is a constraint, as heavier or sturdier items must provide a foundation for lighter or fragile items. The priority level reflects application-specific constraints, such as the need to pack fragile items with care or to prioritize high-value goods for transport. These additional attributes are essential in real-world scenarios, particularly in logistics operations where constraints vary across industries and applications.

The sorting algorithm employs a lexicographical ordering framework, combining these criteria to create a structured queue of items. Mathematically, the sorting problem can be expressed as:

$$\text{Sort } I \text{ such that } i < j \iff \begin{cases} v_i > v_j, & \text{if } v_i \neq v_j, \\ \text{load}_i > \text{load}_j, & \text{if } v_i = v_j, \\ \text{priority}_i > \text{priority}_j, & \text{if } v_i = v_j \text{ and } \text{load}_i = \text{load}_j. \end{cases} \tag{4.3}$$

While the preprocessing step introduces additional computational overhead, the benefits are substantial. By reducing fragmentation and simplifying placement decisions, this sorting framework significantly improves overall packing efficiency. Furthermore, the use of lexicographical ordering ensures consistency and reproducibility, which are critical in applications requiring stringent quality control.

4.3 Handling Empty Maximal Spaces

The concept of maximal spaces is a cornerstone of modern heuristic methods for 3D-BPP. First introduced by Crainic et al. [15], maximal spaces extend the earlier idea of corner points, offering a more comprehensive framework for managing free space within a bin.

Maximal spaces are defined as the largest contiguous regions of available volume where an item can potentially be placed without overlapping existing items or exceeding bin boundaries.

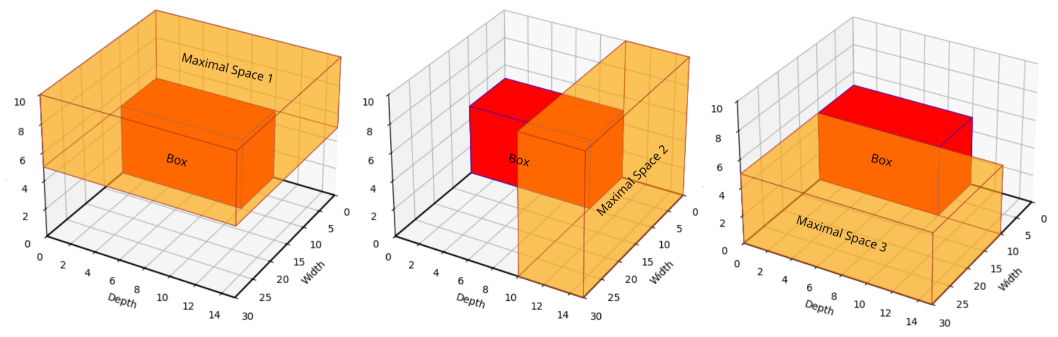

Figure 4.1. Representation of maximal spaces, generated after the insertion of a box into the Bin

4.3.1 Generation of Maximal Spaces

When an item is placed in a bin, the remaining free volume is decomposed into a set of maximal spaces. Mathematically, let S denote the set of maximal spaces, where each space $s \in S$ is defined by its origin (x_s, y_s, z_s) and sizes (w_s, h_s, d_s). After placing an item i with sizes (w_i, h_i, d_i) at position (x_i, y_i, z_i), new maximal spaces are generated along the edges of the item. For example, if i is placed in a corner of the bin, three maximal spaces are created, corresponding to the regions above, in front of, and to the right of the item.

Formally, for each new maximal space s, its dimensions are computed as:

$$(x_s, y_s, z_s) = (x_i + w_i, y_i, z_i), \quad (W - (x_i + w_i), h_i, d_i), \tag{4.4}$$
$$(x_s, y_s, z_s) = (x_i, y_i + h_i, z_i), \quad (w_i, H - (y_i + h_i), d_i), \tag{4.5}$$
$$(x_s, y_s, z_s) = (x_i, y_i, z_i + d_i), \quad (w_i, h_i, D - (z_i + d_i)). \tag{4.6}$$

4.3.2 Overlap Resolution

Overlapping maximal spaces often occur in 3D packing as new spaces are generated. The algorithm resolves overlaps by merging or adjusting affected spaces to maintain consistency. Let s_1 and s_2 be two overlapping spaces. The intersection of these spaces is calculated as:

$$\text{Intersection}(s_1, s_2) = \max(0, \min(x_{s1} + w_{s1}, x_{s2} + w_{s2}) - \max(x_{s1}, x_{s2}))\cdot \tag{4.7}$$

similar terms for y and z. If the intersection volume is non-zero, the algorithm adjusts s_1 and s_2 by removing the overlapping region and recalculating their boundaries.

4.3.3 Selection of Maximal Spaces

The selection of the next maximal space to fill involves balancing spatial efficiency and computational simplicity. The algorithm prioritizes maximal spaces closest to the bin's corners, as these regions are less likely to be usable once central spaces are occupied. The selection criterion can be formalized as:

$$s = \arg\min_{s \in S}(x_s + y_s + z_s), \tag{4.8}$$

where (x_s, y_s, z_s) are the coordinates of the maximal space's origin.

By systematically exploiting available space, the algorithm minimizes wasted volume and reduces the number of bins required. This approach ensures high packing efficiency and adaptability to varying item dimensions.

4.4 Pivot Point Selection for Item Placement

Pivot points are critical in determining the position of an item within a maximal space. The selection of pivot points significantly influences the stability and compactness of the packing. This thesis adopts the extreme point rule proposed by Crainic et al. (2008), which identifies candidate points based on the geometry of existing items and the dimensions of the maximal space.

4.4.1 Definition of Pivot Points

A pivot point is defined as a reference coordinate (x_p, y_p, z_p) within a maximal space $(x_s, y_s, z_s, w_s, h_s, d_s)$ where an item can be placed. The pivot point is chosen to align the item with the edges or corners of the maximal space, reducing fragmentation and ensuring stability. For an item i with dimensions (w_i, h_i, d_i), a valid pivot point satisfies:

$$x_p \geq x_s, \quad x_p + w_i \leq x_s + w_s, \tag{4.9}$$

$$y_p \geq y_s, \quad y_p + h_i \leq y_s + h_s, \tag{4.10}$$

$$z_p \geq z_s, \quad z_p + d_i \leq z_s + d_s. \tag{4.11}$$

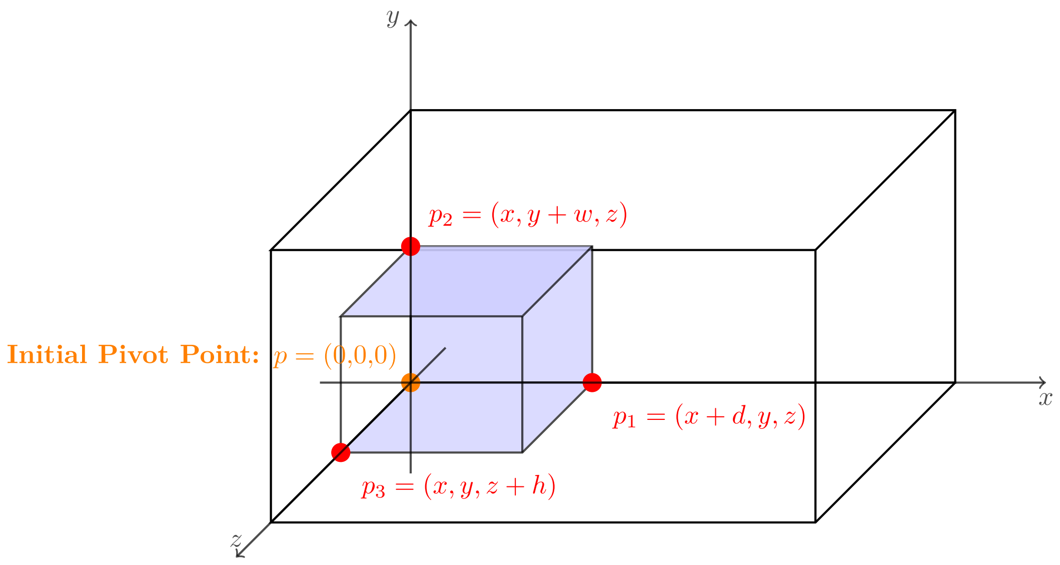

Figure 4.2. Illustration of the Bottom-Left-Fill Packing Strategy in a three-dimensional bin. The initial pivot point ($p = (0, 0, 0)$) is located at the back-left-bottom corner of the bin, where the first item is placed. Once the item is positioned, new potential pivot points ($p_1 = (x + d, y, z)$, $p_2 = (x, y + w, z)$, $p_3 = (x, y, z + h)$) are generated for the placement of subsequent items.

4.4.2 Stability Requirements

Stability is a critical consideration in logistics, particularly when items are subject to motion during transport. To ensure stability, a minimum support surface ratio σ is enforced. This ratio represents the fraction of the item's base area that must be supported by underlying items or the bin floor. Let A_i denote the base area of item i, and A_{support} denote the area of i's base in contact with a stable surface. The stability condition is:

$$\frac{A_{\text{support}}}{A_i} \geq \sigma. \tag{4.12}$$

4.4.3 Orientation Optimization

In many cases, items can be rotated to better fit the available space. The algorithm evaluates all feasible orientations of each item and selects the one that maximizes alignment with the dimensions of the maximal space. Let O_i represent the set of valid orientations for item i. The optimal orientation o^* is chosen as:

$$o^* = \arg\max_{o \in O_i} \text{Utilization}(s, o), \tag{4.13}$$

where Utilization measures the volume of the maximal space occupied by the item.

By incorporating these principles into the pivot point selection process, the algorithm achieves a balance between stability, compactness, and computational efficiency. This approach ensures robust performance across diverse packing scenarios.

4.5 Advantages and Challenges of the Implemented Heuristics

The implemented heuristics combine the strengths of GRASP (*Greedy Randomized Adaptive Search Procedure*) and VND (*Variable Neighborhood Descent*) frameworks to achieve high-quality solutions. This hybrid approach offers significant advantages but also presents challenges, particularly in balancing computational efficiency with solution quality and adapting to the intricacies of real-world problem constraints.

4.5.1 Advantages

The hybrid GRASP/VND algorithm provides a flexible structure that adapts to diverse problem instances. The constructive phase dynamically generates solutions using maximal spaces, while the improvement phase iteratively refines these solutions through multiple neighborhood explorations. This adaptability ensures that the algorithm performs well across a variety of packing scenarios, including those with highly irregular or constrained items.

The robust [35] constructive phase, ensures efficient initial packing. The use of maximal spaces minimizes fragmentation, while stability constraints guarantee practical solutions. Additionally, the incorporation of randomization in the restricted candidate list (RCL) introduces diversity in the search process, reducing the likelihood of stagnation in local optima.

The Variable Neighborhood Descent framework systematically explores multiple neighborhood structures, allowing for significant improvements in packing efficiency and bin utilization. This phase is particularly effective in reducing the total number of bins required. The inclusion of pairwise bin optimizations and targeted re-packing further enhances solution quality.

The diversification phase leverages unpacked frequency data to prioritize items that are frequently left out during packing. This mechanism ensures that the search explores new regions of the solution space, preventing the algorithm from becoming trapped in local optima. Such diversification strategies align with principles from Reactive GRASP [36], further enhancing solution robustness.

The algorithm demonstrates strong scalability, efficiently handling large problem instances with hundreds or thousands of items. The preprocessing steps, including clustering similar items and eliminating infeasible configurations, significantly reduce the problem size, allowing the algorithm to focus on feasible and high-priority items.

4.5.2 Challenges

While the hybrid algorithm achieves high-quality solutions, the combination of GRASP and VND increases computational complexity. The iterative nature of both phases, particularly the exploration of multiple neighborhoods in VND, can lead to high runtime for large-scale instances. Striking a balance between solution quality and computational efficiency remains a challenge, particularly when real-time decision-making is required.

The algorithm's performance is sensitive to key parameters, such as the size of the restricted candidate list (α) in GRASP and the neighborhood exploration strategy in VND. Determining optimal parameter values requires extensive experimentation and may vary depending on the problem instance, making the algorithm less straightforward to implement for new datasets.

Incorporating stability and load-bearing constraints introduces additional complexity in the constructive phase. These constraints often limit feasible placements, increasing the likelihood of unpacked items. Balancing the trade-off between ensuring stability and maximizing space utilization is a non-trivial challenge.

The use of maximal spaces to guide item placement can lead to fragmentation, particularly when items are packed in irregular shapes or orientations. While the algorithm updates and recalculates maximal spaces dynamically, handling overlaps and maintaining consistency in space representation is computationally expensive and prone to errors in edge cases.

The preprocessing steps, such as clustering similar items and eliminating infeasible configurations, significantly impact the algorithm's overall performance. Poor preprocessing can result in suboptimal initial configurations, reducing the effectiveness of subsequent phases. Ensuring robust and accurate preprocessing is critical but requires careful tuning and validation.

The algorithm assumes idealized conditions, such as uniform item properties and consistent bin sizes. Adapting the heuristics to real-world constraints, such as varying item fragility, multi-compartment bins, or dynamic packing scenarios, poses additional challenges. Further research is needed to extend the algorithm's applicability to such scenarios.

Chapter 5

The proposed approach

In this thesis, we present a novel GRASP (*Greedy Randomized Adaptive Search Procedure*) algorithm tailored to address the three-dimensional bin packing problem. One of the core innovations of our approach is the integration of a cutting-edge visualization tool, which enables users to explore packing solutions in three dimensions interactively. This tool provides an intuitive and detailed view of the packing scheme, empowering users to analyze and refine results with ease.

The constructive phase of our algorithm builds upon the heuristic developed by Parreño et al. [37], originally designed for the container loading problem. This heuristic was adapted and enhanced to meet the specific requirements of the three-dimensional bin packing scenario, ensuring robust and efficient initialization of solutions.

To refine these initial solutions, the improvement phase introduces a series of novel moves that are systematically tested and incorporated into a VND (*Variable Neighborhood Descent*) framework [29]. These moves leverage diverse neighborhood structures to iteratively enhance packing efficiency and stability.

The resulting hybrid GRASP/VND algorithm stands out for its simplicity, computational efficiency, and versatility. Extensive computational experiments show that the proposed method consistently produces solutions that are equivalent or superior to those achieved by the most advanced and complex procedures currently available. This combination of high-performance optimization and user-friendly visualization marks a significant contribution to the field.

5.1 Preprocessing Steps

The preprocessing phase plays a critical role in reducing the complexity of the 3D-BPP, by transforming the original instance into a more manageable equivalent. This transformation aims to eliminate certain items or configurations that would otherwise lead to unnecessary computational overhead during the constructive and improvement phases of the algorithm. The first step in preprocessing involves identifying items with dimensions exceeding half the corresponding bin sizes ($w_j > W/2$, $h_j > H/2$, $d_j > D/2$), which are identified as "dominant items." For each dominant item j, a subset S_j is defined, consisting of items that can feasibly coexist with j, based on the conditions:

$$S_j = \{i \mid w_i \leq W - w_j \ \vee \ h_i \leq H - h_j \ \vee \ d_i \leq D - d_j\}. \tag{5.1}$$

If S_j, combined with j, forms a feasible packing, verified by computing the total volume $v_j + V_j$ where $V_j = \sum_{i \in S_j} v_i$ and $v_j = w_j \cdot h_j \cdot d_j$, and checking $v_j + V_j \leq W \cdot H \cdot D$, the bin is added to the solution and the corresponding items are removed from the problem. If this test fails, a refinement is applied by excluding the smallest item k from S_j and recalculating feasibility using $S'_j = S_j \backslash \{k\}$ and $v_j + V_j - v_k \leq W \cdot H \cdot D$.

Large or awkwardly shaped items, such as those occupying an entire bin dimension ($w_j = W$, $h_j = H$, or $d_j = D$), are assigned to separate bins during preprocessing to avoid fragmenting the solution space. Items with identical or similar dimensions are grouped into clusters, reducing redundancy and accelerating decision-making in subsequent phases. Clustering is achieved using a similarity threshold ϵ to group items i and j if $|w_i - w_j| < \epsilon_w$, $|h_i - h_j| < \epsilon_h$, and $|d_i - d_j| < \epsilon_d$. Furthermore, infeasible items that exceed the bin sizes ($w_i > W$, $h_i > H$, $d_i > D$) or violate weight constraints are discarded before the packing process begins.

These preprocessing steps enhance the algorithm's efficiency by reducing the number of items to consider, minimizing computational overhead, and focusing the algorithm's resources on generating high-quality solutions. By transforming the original problem into an equivalent but simplified instance, preprocessing contributes significantly to the scalability and effectiveness of the hybrid GRASP/VND approach.

5.2 Constructive Phase: Initial Packing Strategy

The constructive phase establishes the foundation of the 3D Bin Packing heuristic by efficiently generating an initial feasible packing configuration. This phase is responsible for defining maximal spaces, dynamically updating them after item placement, and ensuring adherence to stability constraints. The packing strategy is designed to maximize space utilization while minimizing fragmentation within bins.

At the beginning of each iteration, the set of items to be packed, denoted as $I = \{i_1, i_2, \ldots, i_n\}$, is initialized and sorted based on priority. Prioritization criteria include item volume, load-bearing capacity, and stability considerations. Similarly, each bin $B = \{b_1, b_2, \ldots, b_m\}$ is initialized with a list of maximal spaces $S = \{E\}$, where E represents the entire bin volume. As items are placed, maximal spaces are continuously subdivided to reflect available space, ensuring efficient space utilization. This process is mathematically formulated as:

$$S_k = \{s \mid s \subseteq V_b, \text{ItemFit}(i, s)\} \tag{5.2}$$

where S_k represents the maximal spaces at iteration k, s is an available space, and $\text{ItemFit}(i, s)$ ensures that item i fits within space s without exceeding bin sizes.

To maximize bin utilization, the algorithm selects a maximal space $S^* \in S$ for placement based on proximity to the bin's corners. [15] If multiple spaces satisfy this criterion, the one with the largest volume is prioritized. Items are then evaluated for placement in S^* using two main objectives:

- **Best Volume Utilization:**Maximizes the volume occupied within S^*.
- **Best Fit Strategy:**Prioritizes item placement with minimal distance to space boundaries.

If multiple items satisfy the placement conditions, layers of identical items are considered to enhance packing efficiency [10]. The sorting of items before placement is determined using the following priority function:

$$\text{Priority}(i) = (p_i, -v_i, -l_i) \tag{5.3}$$

where p_i, v_i, and l_i represent the priority level, volume, and load-bearing capacity of item i, respectively.

To maintain structural stability, the algorithm verifies the support ratio r before placement:

$$\text{Support Ratio} = \frac{\text{Supported Area}}{\text{Base Area}} \geq 0.75 \tag{5.4}$$

If an item satisfies stability and fit constraints, it is placed, and S^* is subdivided accordingly. Non-disjoint maximal spaces are recalculated dynamically, and infeasible spaces are discarded to enhance computational efficiency. Additionally, a diversification mechanism is incorporated to improve solution robustness. Items are selected from a restricted candidate list (RCL), which contains the top $100\delta\%$ feasible configurations. The parameter δ is adjusted adaptively using principles from Reactive GRASP, balancing exploration and exploitation based on previous iterations.

To ensure numerical precision, all dimensions undergo rounding based on a predefined decimal format before packing. The bins are then sorted in descending order by volume, helping larger bins to accommodate a greater range of items, while items are sorted by decreasing volume to prioritize large-item placement first. If binding constraints are present, a secondary sorting function ensures that dependent items are packed together.

The core packing operation follows a structured placement routine. Each item is first tested in its original orientation. If a valid placement is found, it is inserted at a pivot point within the bin using the function `putItem()`. Following placement, maximal spaces are updated to reflect the new occupancy. If an item does not fit in its default orientation, all six possible rotations are evaluated sequentially. If a feasible rotated placement is found, the item is placed and its orientation recorded. If no valid placement exists, the item is marked as unfit and set aside for later consideration.

To further refine packing efficiency, a heuristic-based candidate selection approach is introduced. Instead of evaluating every possible packing option, the algorithm maintains a limited pool of top candidate placements, dynamically updated based on the feasibility of solutions. This strategy significantly reduces computation time while still ensuring an optimized configuration. Additionally, bin compaction is periodically enforced, ensuring that partially filled bins are re-evaluated to improve packing density.

To track execution efficiency, the algorithm records computation time for each major step. A performance log stores placement times, space updates, and sorting operations, enabling further optimization of the algorithm. The systematic use of maximal space partitioning, rotational feasibility checking, and adaptive placement heuristics ensures that the constructive phase produces an optimized initial packing layout while adhering to constraints on volume, stability, and binding dependencies.

Pseudocode for the Constructive Phase

Algorithm 9 Constructive Phase

```
Input: Set of items $I$, set of bins $B$
Output: Initial packing solution
Sort $B$ by decreasing volume if $bigger\_first = True$
Sort $I$ by decreasing volume $v_i = w_i \cdot h_i \cdot d_i$ with priority level as a tiebreaker
for each $i \in I$ do
  for each $b \in B$ do
    Generate maximal spaces in $b$
    if $i$ is already assigned to a different bin then
      continue
    end if
    if $i$ is already packed in $b$ then
      continue
    end if
    for each maximal space $s$ in $b$ do
      if $i$ fits in $s$ in original orientation then
        Place $i$ at the pivot point in $b$
        Update maximal spaces in $b$
        Mark $i$ as packed and remove from $I$
        break
      end if
    end for
    if $i$ was packed successfully then
      break
    end if
  end for
  if $i$ is not packed in any bin then
    for each rotation $r = 1, \ldots, 6$ of $i$ do
      Generate rotated version $i_r$
      for each $b \in B$ do
        for each maximal space $s$ in $b$ do
          if $i_r$ fits in $s$ then
            Place $i_r$ at the pivot point in $b$
            Update maximal spaces in $b$
            Mark $i$ as packed with rotation $r$
            break
          end if
        end for
      end for
    end for
  end if
  if $i$ still cannot be packed then
    Mark $i$ as unfit and add to $b.unfitted\_items$
  end if
end for=0
```

The constructive phase relies on the concept of *maximal spaces*, that was introduced on the previous chapter. The pivot point, chosen within a maximal space, ensures compact and stable placements.

5.2.1 Improvement Phase: Refining the Solution

Building on the initial solution, the improvement phase employs a Variable Neighborhood Descent (VND) approach [29]. The objective is to explore multiple neighborhoods of the current solution to identify configurations that reduce the number of bins or improve packing density. Bins with utilization below a predefined threshold are selected. Items from these bins, along with any unpacked items, are repacked using the constructive heuristic. Random percentages $k \in [30, 90]$ of items may also be removed and reprocessed. Two primary neighborhood structures are defined:

1. **Repacking Underutilized Bins**: Bins with utilization $u < 0.5$ are paired for repacking. Items are removed and repacked using a greedy heuristic to optimize space utilization.

2. **Item Reordering**: Items within a bin are reordered based on stability and space efficiency metrics to identify alternative packing configurations.

For solutions with more than two bins, pairs of bins are selected, emptied, and repacked with the goal of creating complementary configurations. Items that were previously difficult to pack are prioritized during this process. The neighborhoods are explored in a sequential manner (VNDseq), restarting from the first neighborhood only when no improvement is achieved. This ensures a comprehensive exploration of the solution space. [38] The VND algorithm iteratively applies neighborhood structures, accepting a new solution only if it improves the overall objective:

$$U = \frac{\sum_{i \in \text{PackedItems}} v_i}{\sum_{b \in \text{Bins}} V_b} \tag{5.5}$$

The process begins by initializing an iteration counter and tracking the last improvement step. The algorithm performs a predefined number of iterations, stopping early if no improvements have been made in a given number of consecutive steps. During each iteration, the algorithm attempts to refine the solution by applying a set of predefined neighborhood moves. Each neighborhood technique is explained in detail below.

Removing the Least Occupied Bins

This technique identifies and removes the least occupied bin if it is significantly less filled than the other bins. The algorithm first determines the bin with the lowest utilization and checks if its utilization is below 40%. If this condition is met, the items from the underfilled bin are repacked into the remaining bins. If repacking is successful, the bin is cleared and removed from the solution, reducing the total number of bins used.

Selective Removal from Bins

In this method, a small fraction of items is selectively removed from the least occupied bin to test whether they can be repacked more efficiently. The bin with the lowest volume utilization is identified, and a fixed percentage (typically 20%) of items are removed from it. These items are then attempted to be repacked into other bins. If successful, this improves the overall packing efficiency and balances the load across bins.

Splitting Bins

This strategy attempts to split bins based on a randomly selected spatial dimension (width, height, or depth). The algorithm selects a bin with nonzero utilization and determines a splitting axis. Items that occupy more than half of the bin's space along the selected axis are removed and repacked into other bins. This technique helps exploring alternative packing arrangements that might lead to better space utilization.

Compacting Bins

This neighborhood move focuses on consolidating items into fewer bins by redistributing the contents of underutilized bins. Bins are sorted by their utilization in descending order, and those with utilization below 50% are considered for compaction. The algorithm iterates through the items in these bins and attempts to repack them into higher-utilization bins. If an item is successfully relocated, it is removed from its original bin. If a bin becomes empty after this process, it is removed, reducing the overall bin count.

Pairwise Bin Merging

Pairwise bin merging is an aggressive optimization strategy that attempts to merge two existing bins into a single bin while minimizing unnecessary repacking. The algorithm checks whether the combined volume of two bins can fit into one without exceeding its capacity. If feasible, items from both bins are repacked into a single bin, and the original bins are cleared. This technique is particularly useful in reducing fragmentation and improving packing density.

To evaluate the effectiveness of each move, the algorithm maintains a deep copy of the current best packing solution. The initial solution is evaluated and compared to the others solutions using various metrics stored in a dictionary, structured as follows:

- `Total number of bins (total_bins)`: The total number of bins used in the solution.
- `Total volume (total_volume)`: The cumulative volume of all bins.
- `Packed volume (packed_volume)`: The total volume occupied by successfully packed items.

- `Utilization percentage (utilization_percent)`: The proportion of occupied bin space, calculated as:

$$\text{utilization\_percent} = \left(\frac{\text{packed\_volume}}{\text{total\_volume}}\right) \times 100 \tag{5.6}$$

- `Total number of items (total_items)`: The sum of packed and unpacked items.
- `Packed items (packed_items)`: The number of items that have been successfully placed within bins.
- `Packing efficiency percentage (packing_efficiency_percent)`: The proportion of items successfully packed, computed as:

$$\text{packing\_efficiency\_percent} = \left(\frac{\text{packed\_items}}{\text{total\_items}}\right) \times 100 \tag{5.7}$$

- `All items packed (all_items_packed)`: A boolean flag indicating whether all items have been successfully packed.
- `Number of unfit items (unfit_items)`: The count of items that could not be placed in any bin.

These metrics allow the algorithm to compare different packing configurations and determine whether an improvement has been achieved. The objective is to maximize space utilization and packing efficiency while minimizing the number of bins used.

The evaluation function provides insight into how well the packing configuration performs relative to the optimization goals.

Each neighborhood move is applied sequentially. Before executing a move, a deep copy of the current packing solution is created to ensure that changes can be reverted if necessary. Once a move is applied, the resulting configuration is re-evaluated, and its performance metrics are compared against the best-known solution. If the new solution is strictly better, it is accepted, and the best solution is updated. If the new solution is not strictly better but falls within an acceptable threshold based on a random acceptance probability, it may still be chosen to prevent the algorithm from getting stuck in local optima. If a neighborhood move fails to improve the solution, the algorithm proceeds to the next neighborhood move in the list. However, if a successful improvement is found, the process resets to the first neighborhood move, allowing for further exploration of alternative configurations. This adaptive mechanism ensures that the algorithm continuously searches for improvements rather than following a rigid sequence of moves. The improvement phase also includes mechanisms for tracking execution times for each step. The algorithm logs the time required for deep copying solutions, evaluating packing configurations, applying neighborhood moves, and updating the best solution. These timing statistics provide insights into computational performance and potential bottlenecks in the optimization process. At the end of the improvement phase, the best solution discovered is restored, ensuring that the final configuration represents the most optimized packing arrangement obtained during the iterations. By systematically refining the packing layout, adjusting item placements, and exploring alternative configurations, the improvement phase enhances the efficiency and effectiveness of the algorithm.

Algorithm 10 ImproveSolution Function

```
0: **Input:** Current packing solution $S$
0: **Output:** Boolean indicating whether an improvement was found
0: $best\_solution \leftarrow S$
0: $best\_metrics \leftarrow \text{EvaluateSolution}(S)$
0: $improved \leftarrow False$
0: Define neighborhood moves: $\mathcal{N} = \{\text{RemoveLeastOccupiedBins, SelectiveRemoval, SplitBins, etc.}\}$
0: $neighborhood\_index \leftarrow 0$
0: **while** $neighborhood\_index < |\mathcal{N}|$ **do**
0:   $move\_function \leftarrow \mathcal{N}[neighborhood\_index]$
0:   $current\_solution \leftarrow \text{DeepCopy}(S)$
0:   $current\_metrics \leftarrow \text{EvaluateSolution}(current\_solution)$
0:   $move\_successful \leftarrow$ move_function()
0:   **if** $move\_successful$ **then**
0:     $new\_metrics \leftarrow \text{EvaluateSolution}(S)$
0:     **if** IsNewSolutionBetter($new\_metrics, best\_metrics$) **then**
0:       $best\_solution \leftarrow \text{DeepCopy}(S)$
0:       $best\_metrics \leftarrow new\_metrics$
0:       $improved \leftarrow True$
0:       $neighborhood\_index \leftarrow 0$ {Restart neighborhood search}
0:     **else**
0:       With probability 0.1, accept the new solution randomly
0:       **if** Accepted **then**
0:         $best\_solution \leftarrow \text{DeepCopy}(S)$
0:         $best\_metrics \leftarrow new\_metrics$
0:         $improved \leftarrow True$
0:         $neighborhood\_index \leftarrow 0$
0:       **else**
0:         $S \leftarrow current\_solution$ {Revert to previous state}
0:         $neighborhood\_index \leftarrow neighborhood\_index + 1$
0:       **end if**
0:     **end if**
0:   **else**
0:     $neighborhood\_index \leftarrow neighborhood\_index + 1$ {Move to next neighborhood}
0:   **end if**
0: **end while**
0: Restore best solution: $S \leftarrow best\_solution$
0: **Return** $improved$ =0
```

5.2.2 Diversification Phase: Avoiding Local Optima

Following several iterations without improvement, the algorithm enters a diversification phase. This phase avoids the algorithm getting trapped in local optima by prioritizing items that frequently remain unpacked. Items that remain unpacked most frequently are identified using a frequency vector $F(i)$, where:

$$F(i) = \sum_{t=1}^{T} I(i \in \text{UnpackedItems}_t), \tag{5.8}$$

and I is an indicator function. These items are prioritized for placement in subsequent iterations, ensuring a broader exploration of the search space [39]. Only one difficult-to-pack item is placed in each bin during this phase to prevent poor solutions. Unpacked items are sorted adaptively based on their frequency and priority:

$$\text{SortKey}(i) = (F(i), p_i, -v_i) \tag{5.9}$$

This approach increases the likelihood of packing previously problematic items.

5.2.3 Stopping Criteria

The algorithm terminates when one of the following criteria is met:

- Maximum iterations reached.
- No further improvement in utilization.
- All items are successfully packed.

5.2.4 Key Innovations and Contributions

The proposed algorithm integrates several novel elements and builds upon the contributions of key researchers in the field. Maximal space management extends the work of Parreño et al. [37] with dynamic recalculations to minimize fragmentation. Dynamic randomization implements Reactive GRASP principles [36] to balance exploration and exploitation. Stability and load-bearing checks introduce explicit constraints for stability, ensuring feasible and practical packing solutions. Neighborhood diversity combines neighborhood structures inspired by Hansen and Mladenović [29] and Crainic et al. [15] to achieve robust improvements.

Chapter 6

Experimental Results and Analysis

6.0.1 Challenges in Comparing 3D Bin Packing Algorithms

Evaluating and comparing algorithms for the 3D-BPP presents numerous challenges due to the diversity of problem constraints, optimization objectives, and computational methodologies. While the literature provides a vast range of heuristic, metaheuristic, and hybrid approaches, meaningful comparisons are hindered by inconsistencies in benchmark datasets, performance metrics, and problem formulations.

One of the primary challenges in comparing 3D-BPP algorithms arises from the various constraints considered in different studies. Many existing approaches focus solely on geometric constraints such as item dimensions and bin capacities, whereas real-world applications impose additional practical constraints. For instance, our algorithm incorporates weight distribution, fragility, item priority, and incompatibility, which are often overlooked in other works. According to [39], only a limited number of studies address load-bearing constraints, despite their significance in industrial applications.

Moreover, the literature reveals a lack of standardized constraint sets. Some algorithms allow arbitrary item orientations, while others restrict rotations to maintain structural integrity. Additionally, [14] highlights that real-world problems frequently involve stacking constraints, stability requirements, and weight limits, yet many benchmark instances fail to include these aspects.

Benchmark Dataset Inconsistencies

Standardized datasets are crucial for fair algorithmic comparisons, yet the 3D-BPP literature lacks universally accepted benchmarks. Many studies rely on synthetic instances with randomly generated item sizes, while others utilize real-world datasets that introduce additional complexity. Furthermore, some benchmark datasets, such as those used in [40], focus solely on packing efficiency without considering real-world constraints.

Furthermore, algorithmic implementations may leverage different optimization techniques, such as simulated annealing, tabu search, or genetic algorithms [41], each with distinct computational trade-offs. The selection of stopping criteria, parameter tuning, and search strategies can significantly impact results, making direct comparisons challenging.

Future Directions for Standardized Comparisons

To enhance comparability among 3D-BPP algorithms, future research should focus on developing standardized benchmark datasets that incorporate practical constraints, such as weight distribution, fragility, and item compatibility. Moreover, consensus on evaluation metrics is needed to ensure that comparisons are meaningful across different studies. Initiatives such as the ones suggested by [42] advocate for open-access repositories of benchmark instances, allowing researchers to evaluate algorithms under consistent conditions.

Additionally, reporting detailed experimental conditions, including computational resources and parameter settings, can improve reproducibility and facilitate fair comparisons. By addressing these challenges, the research community can move toward more robust and reliable evaluations of 3D-BPP algorithms.

6.1 Dataset Description

The experimental evaluation of the proposed algorithm in this thesis has been conducted using a combination of standard benchmark datasets and custom-designed instances. The standard dataset used in this study is a well-known benchmark proposed by Osaba et al. [43], which consists of 12 instances designed to capture various complexities inherent to real-world 3D-BPP scenarios. Additionally, a set of custom test instances was generated to evaluate the algorithm's performance under extreme conditions.

6.1.1 Benchmark Dataset

The benchmark dataset consists of 12 different instances, each characterized by a varying number of items (ranging from 38 to 53), bin sizes, weight constraints, package affinity requirements, ordering preferences, and load balancing constraints. These instances were originally developed to assess quantum solvers but have been adapted here for heuristic-based optimization.

- **Instance Generation:** All packages composing each instance have been randomly generated using the in-house instance generator, `Q4RealBPP-DataGen`. This approach avoids any potential bias inherent in manual instance creation and ensures a diverse and representative sample of real-world problems.

- **Synthetic Data Generation Script:** Along with the 12 benchmark instances, the dataset includes a Python script that enables the generation of synthetic datasets. This tool allows researchers to create customized instances for benchmarking purposes, thereby facilitating the extension and adaptation of the benchmark to various research needs.

- **Open Source Availability:** Both the benchmark instances and the `Q4RealBPP-DataGen` script are released as open source. This not only allows for transparency and reproducibility but also enables modifications or extensions to accommodate other variants of the Bin Packing Problem, further promoting research in this field.

- **Baseline Solver Results:** The dataset package includes results obtained using the Leap Constrained Quadratic Model Hybrid Solver (LeapCQMHybrid) by D-Wave. These results are provided in both image and text formats, serving as a valuable baseline for comparing the performance of other solvers.

Each of the 12 instances in the dataset incorporates several realistic constraints that reflect industrial requirements:

- **Item and Bin sizes:** As a three-dimensional problem, each instance defines the sizes of bins ($[X, Y, Z]$) and the corresponding dimensions (length, width, height) for the items. An item can be packed only if it fits within these predefined dimensions.

- **Overweight Restrictions:** In certain instances, every item has an associated weight, and bins have a maximum weight capacity. This ensures that the total weight of the items packed in a bin does not exceed its capacity.

- **Affinities Among Package Categories:** The dataset introduces both positive affinities and incompatibilities. Items with a positive affinity are required to be packed together, whereas items that are incompatible must be placed in separate bins.

- **Preferences in Relative Positioning:** The benchmark allows for the specification of relative positioning rules. For example, load-bearing items might be required to be placed beneath lighter items (with respect to the Z-axis), or items may be sorted along the X-axis to reflect a delivery schedule.

- **Load Balancing:** Certain instances also incorporate load balancing constraints by specifying a center of mass. This encourages a balanced distribution of items within the bin, contributing to overall stability.

The details of each benchmark instance are provided in Table 6.1.

6.1.2 Integration with Custom High-Density Packing Scenarios

In addition to the benchmark instances, custom instances have been created to test the algorithm's ability to improve an initial solution by repacking items and exploring the neighborhood. For these custom cases:

- **High-Density Packing:** The custom instances simulate scenarios where bins are densely packed, with item sizes approaching the bin capacities. This stresses the algorithm's capability to exploit minimal available space.

Table 6.1. Characteristics of Benchmark Dataset Instances

Instance	**Items**	**Dimensions**	**Weight Limit**	**Affinities**	**Ordering**	**Load Balance**
3dBPP_1	51	Yes	No	No	No	No
3dBPP_2	51	Yes	Yes	No	No	No
3dBPP_3	52	Yes	No	No	No	No
3dBPP_4	52	Yes	No	No	Yes	No
3dBPP_5	53	Yes	No	No	No	No
3dBPP_6	53	Yes	No	Yes	No	No
3dBPP_7	46	Yes	No	No	No	No
3dBPP_8	46	Yes	No	Yes	No	No
3dBPP_9	47	Yes	No	No	No	Yes
3dBPP_10	51	Yes	No	No	No	Yes
3dBPP_11	38	Yes	Yes	Yes	Yes	Yes
3dBPP_12	38	Yes	Yes	Yes	Yes	Yes

- **Relaxed Constraints:** To focus on repacking performance, some constraints (e.g., fragility and strict weight limits) are relaxed. This allows the algorithm to concentrate on optimizing the spatial arrangement and improving the overall packing efficiency.
- **Neighborhood Exploration:** The high-density scenarios serve as a testing ground for the neighborhood exploration mechanisms in the Grasp/VND algorithm. By starting from an initial, densely packed solution, the algorithm is challenged to identify subtle repositioning opportunities that can lead to significant performance improvements.

Each instance was designed to capture different packing challenges by varying the number of items and the distribution of their dimensions and weights. The bins were kept constant in size at $1200 \times 1200 \times 1200$ units, ensuring that variations in performance could be attributed solely to item configurations rather than bin capacity changes. The instances are described as follows:

- **Instance 1:** 50 items with moderate size variations, having dimensions ranging from (120,150,160) to (300,400,320) units. The weight distribution was relatively uniform, between 10 and 55 units.
- **Instance 2:** 60 items with a wider range of sizes, including elongated structures such as (150,250,200). The weight distribution varied more significantly, requiring better balance optimization.
- **Instance 3:** 70 items, including a large number of small objects (e.g., (50,50,50) units), creating a challenge in efficiently utilizing available space.
- **Instance 4:** 80 items with heavier objects, some weighing up to 100 units. Larger items such as (900,900,900) were introduced to test weight distribution handling.

- **Instance 5:** 90 items with extreme variations in weight. Small objects could be significantly heavier than larger ones, adding complexity to weight balancing.
- **Instance 6:** 100 items with diverse dimensions, including highly asymmetric objects such as (380,470,390). Strategic placement was required to minimize wasted space.
- **Instance 7:** 110 items with a mixture of extremely light and extremely heavy objects. Weight balancing played a crucial role in optimizing the packing configuration.
- **Instance 8:** 115 items featuring many long, thin items (e.g., (210,240,280)). This case required careful orientation decisions to improve bin utilization.
- **Instance 9:** 120 items with a higher number of irregularly shaped packages, such as (410,500,420). The challenge was to find optimal stacking configurations.
- **Instance 10:** 95 items with a balanced distribution in terms of size and weight. This test case served as a baseline to evaluate general algorithmic efficiency.
- **Instance 11:** 105 items where items of similar dimensions had significantly different weights (e.g., (330,290,210) could weigh 55 or 80 units). This required the algorithm to adapt its weight management strategy dynamically.
- **Instance 12:** 110 items simulating real-world warehouse packing scenarios, including pallet-sized objects and small cubic packages.
- **Instance 13:** 120 items representing the most complex case, with the highest variation in item dimensions. High-density items such as (350,310,230) required careful spatial optimization.

Overall, the combination of the benchmark dataset and the custom high-density instances offers a comprehensive framework for assessing the effectiveness of the proposed heuristic approach. This dual strategy not only facilitates performance evaluation on standardized, real-world inspired cases but also challenges the algorithm under extreme conditions, thus highlighting its potential for practical applications in industrial settings.

6.2 Performance Metrics and Evaluation Methods

The performance of the proposed heuristic algorithm for the 3D-BPP is evaluated using key metrics that assess both solution quality and computational efficiency. Given its application in a startup setting, **execution time** is a critical factor alongside packing efficiency.

The primary metric is the total packed volume, representing the sum of item volumes successfully placed in bins. Higher packed volume indicates better space utilization. The **utilization percentage** measures space efficiency and is calculated as:

$$U = \frac{V_p}{V_b} \times 100 \tag{6.1}$$

where V_p is the total packed volume and V_b is the total available volume. A higher U implies improved space utilization.

Another key metric is the **total number of bins used**, crucial for minimizing logistics and operational costs. A balance must be struck between maximizing utilization and minimizing bin usage.

Additionally, the **total number of packed items** and **total unfitted items** are recorded, along with their corresponding **total unfitted volume**, to track packing effectiveness.

To ensure robustness, multiple test runs are conducted, with results averaged to account for variations.

Scalability is evaluated by testing the algorithm on larger datasets, measuring how execution time and packing efficiency change with problem size.

Overall, this evaluation framework ensures a balance between solution quality and efficiency, aligning with real-world performance requirements.

6.3 Analysis of Results

This section presents a comprehensive analysis of the experimental results obtained by comparing our proposed heuristic algorithm with the First Fit (FF) and Simulated Annealing (SA) methods for the Three-Dimensional Bin Packing Problem (3D-BPP). The analysis is structured into several sections, including an overview of the experimental setup and performance metrics, an examination of the initial solution and its subsequent improvement through local search, a detailed graphical representation of the results, and a statistical discussion of the findings.

6.4 Experimental Setup and Performance Metrics

All experiments were conducted under controlled conditions on a dedicated machine (Intel Core i7, 8 GB RAM). The test instances include both benchmark cases and custom high-density scenarios generated using Q4RealBPP-DataGen. Each algorithm was executed multiple times to mitigate variability, and the results were averaged over these runs.

The performance of the algorithms is evaluated based on the following key metrics:

- **Total Packed Volume (V_p):** The sum of the volumes of all items successfully placed into bins.
- **Utilization (U):** The efficiency of space usage, calculated as

  $$U = \frac{V_p}{V_b} \times 100,$$

  where V_b is the total available volume of the bins.

- **Total Bins Used (B)**: The number of bins required to pack the items.
- **Total Items Packed (I_p)**: The number of items that have been successfully packed.
- **Execution Time (T)**: The time, measured in seconds, required to compute the solution.

In addition, the evolution of the solution from the initial configuration to the improved state after the local search phase is tracked to assess how additional computation time translates into enhanced packing efficiency.

6.5 Parameter Tuning Process

The effectiveness of the GRASP/VND packing algorithm is heavily influenced by its parameter choices. Several key hyperparameters control how the items are sorted, how they are placed into bins, and how the local search mechanism refines the solution. The tuning process was conducted iteratively, balancing computational efficiency with packing effectiveness. Initial tests demonstrated that certain constraints, such as stability requirements and extensive improvement iterations, significantly increased execution time without substantial gains in solution quality. To address this, several adjustments were made to improve the trade-off between performance and efficiency, particularly for the custom dataset, where some constraints such as weight limitations and item priority were relaxed to better reflect practical applications.

The primary parameters used in the algorithm are:

- *bigger_first*: A boolean that determines whether bins and items should be sorted in descending order of volume before packing. Sorting by larger volumes first helps prioritize the placement of bulkier items early in the process.
- *distribute_items*: Controls whether items should be evenly distributed across bins rather than greedily packed into the first available bin. This can lead to better load balancing.
- *fix_point*: Determines whether items are placed at a fixed reference point within the bin during packing, reducing computational overhead for placement validation.
- *check_stable*: If enabled, ensures that placed items have sufficient support to maintain stability, preventing unrealistic or infeasible packings.
- *support_surface_ratio*: Specifies the minimum fraction of an item's base that must be in contact with another object or the bin's surface to be considered stable.
- *number_of_decimals*: Defines the precision level for numerical calculations. A higher number of decimals improves accuracy but increases computational complexity.
- *max_iterations*: Sets the maximum number of iterations for the improvement phase, determining how extensively the algorithm refines initial solutions through VND.

One of the primary adjustments made was to prioritize larger items first by setting *bigger_first=True*. This sorting strategy ensures that bulkier objects are placed early in the process, reducing the likelihood of fragmentation and making more effective use of available space. Without this sorting step, smaller items often occupy crucial packing regions, forcing larger objects to be placed inefficiently or rejected altogether. Additionally, *distribute_items=True* was enabled to ensure a more balanced allocation of items across bins. This modification prevents overloading specific bins and mitigates uneven load distribution, which is especially relevant when weight constraints are considered. Although weight limitations were relaxed in the custom dataset, the distribution mechanism remains beneficial for enhancing the practical feasibility of the packing solutions.

The handling of stability constraints also required careful calibration. In the initial tuning phase, *check_stable=True* with a high *support_surface_ratio* (above 0.5) led to a sharp increase in rejected placements, causing unnecessary bin usage. While stability is an essential consideration, particularly in real-world applications where items should not topple, a rigid enforcement of support constraints resulted in excessive computational overhead. Through experimentation, *support_surface_ratio* was lowered to 0.25, allowing for more flexible placement while still maintaining reasonable stability. Furthermore, *check_stable* was ultimately disabled in the custom dataset to reduce execution time, as the dataset did not include scenarios where precise stability calculations were critical. The *fix_point* parameter was retained, ensuring that items are placed at a fixed reference point within the bin to streamline placement verification and enhance computational efficiency.

The iterative improvement phase plays a key role in refining the packing solution. Initially, the *max_iterations* parameter was set to 1000, but it became evident that excessive iterations provided diminishing returns. While an extended local search can marginally improve the packing configuration, the computational cost quickly escalates. To address this, an adaptive stopping criterion was introduced, where the improvement phase terminates if no better solution is found after five consecutive iterations. This strategy significantly reduces runtime while preserving solution quality. Ultimately, for the custom dataset, *max_iterations* was set to 10, relying on the initial placement strategy without extensive refinement, as the relaxed constraints diminished the need for aggressive local search improvements.

Another critical aspect of parameter tuning was managing computational trade-offs. Increasing constraints, such as enforcing strict stability checks or running excessive iterations, prolonged execution time without yielding substantial packing efficiency gains. A balance was achieved by selecting a parameter configuration that ensured high space utilization while keeping runtime manageable. The final configuration used for the custom dataset was: *bigger_first=True*, *distribute_items=True*, *fix_point=True*, *check_stable=False*, *support_surface_ratio=0.25*, *number_of_decimals=3*, and *max_iterations=10.* This setup maintains efficient packing without unnecessary computational overhead, ensuring that the algorithm performs optimally given the adjusted dataset constraints. Future improvements could explore dynamic parameter tuning, where algorithm settings adapt to instance characteristics in real time, further enhancing robustness and efficiency.

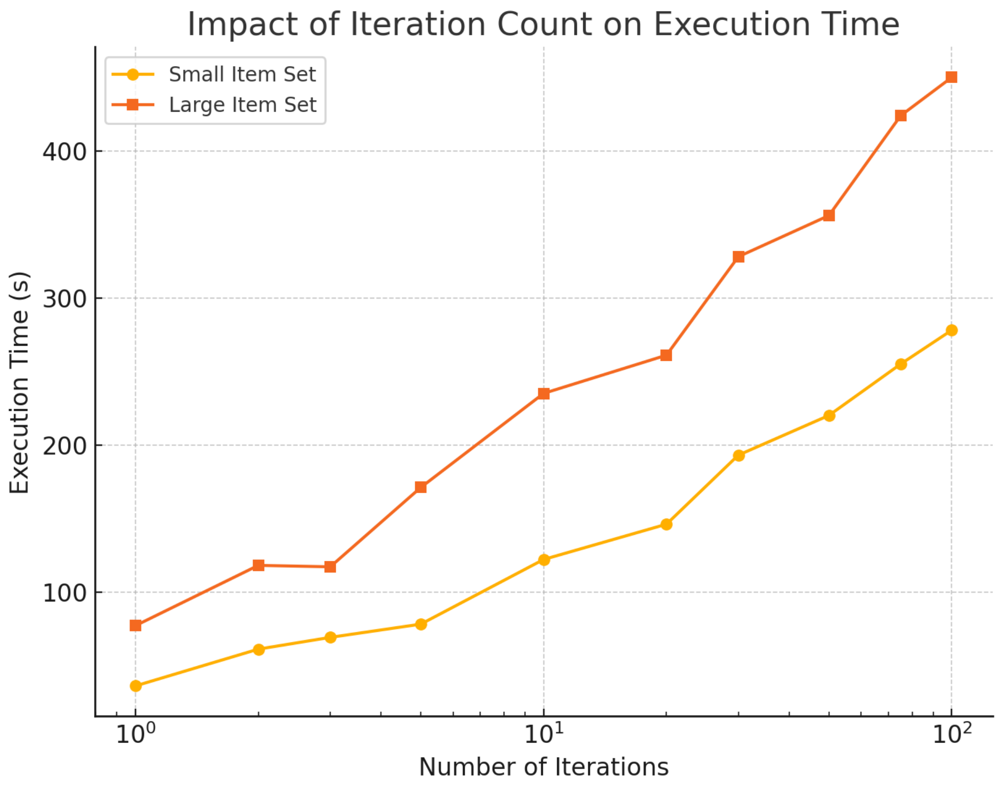

Figure 6.1. Impact Of Iteration Count On Execution Time

6.6 Results and Analysis

The experiments were conducted on a series of test instances (Tests 1 through 12) where key performance metrics were recorded for each algorithm. These metrics include execution time (in seconds), the number of bins used, and the utilization percentage (i.e., the ratio of the total packed volume to the available bin volume). To facilitate a clearer presentation, the performance data is summarized in Table 6.6.

Summary of Performance Metrics for GRASP/VND, FFD, and SA across Tests 1-12.

The table above presents a concise summary of the performance metrics. However, to truly understand the efficacy of the GRASP/VND algorithm, it is imperative to analyze these results in depth, discussing the underlying trends, strengths, and potential weaknesses.

One of the most striking observations is the consistent bin consolidation achieved by the GRASP/VND algorithm. In almost all tests, GRASP/VND manages to pack items using only a single bin, with the only exception being Test 5 where two bins were employed. This is a significant advantage over FFD, which frequently requires two bins to accommodate the same set of items. In a real-world scenario, reducing the number of bins translates directly into lower logistical and storage costs, as well as improved operational efficiency. SA, while competitive in some tests, also sometimes resorts to using two bins,

Table 6.2. Performance Comparison of GRASP/VND, FFD, and SA on the Benchmark Dataset

	TEST 1	TEST 2	TEST 3	TEST 4	TEST 5	TEST 6	TEST 7	TEST 8	TEST 9	TEST 10	TEST 11	TEST 12
GRASP/VND												
Exec. time	182.23	191.09	165.32	87.29	181.84	158.97	132.33	133.07	142.14	138.13	212.63	66.19
Bins Used	1	1	1	1	2	1	1	1	1	1	1	1
Utilization (%)	64.58	60.19	63.68	45.39	33.23	80.2	75.18	72.14	47.29	56.83	78.12	66.17
FFD												
Exec. time	153.29	67.38	102.53	53.38	90.29	68.29	99.14	83.13	101.58	126.24	82.4	89.36
Bins Used	2	1	2	1	2	1	1	1	1	2	2	1
Utilization (%)	19.54	22.9	31.53	16.71	17.68	34.28	42.67	39.87	17.16	19.45	18.76	41.34
SA												
Exec. time	139.09	148.29	254.51	200.33	202.12	255.53	171.15	121.59	162.35	153.59	149.67	175.01
Bins Used	1	1	2	1	2	1	1	1	1	2	1	1
Utilization (%)	55.47	60.23	33.12	55.23	34.58	75.23	81.98	49.16	35.44	22.49	62.59	57.87

thereby reinforcing the superior space consolidation capability of GRASP/VND.

The utilization percentage further supports the effectiveness of the GRASP/VND algorithm. Utilization reflects how efficiently the available bin space is used, and in this regard, GRASP/VND consistently outperforms FFD and often matches or exceeds the performance of SA. For instance, in Test 6, GRASP/VND achieves an outstanding 80.20% utilization compared to 34.28% for FFD, and even SA, despite reaching 75.23%, shows more variability across different tests. High utilization percentages are indicative of an algorithm's ability to pack a larger volume of items within the constraints of the bin sizes, which is critical in applications where maximizing space is a priority.

Execution time is another critical metric, especially in environments where rapid decision-making is essential. Although GRASP/VND does not always register the lowest execution times—occasionally being outperformed by FFD in tests such as Test 2 (191.09 s vs. 67.38 s) and Test 4 (87.29 s vs. 53.38 s)—its execution times are competitive when considering the quality of the solution delivered. SA, in contrast, shows a high degree of variability in execution time, with some tests taking significantly longer (e.g., 254.51 s in Test 3) without a consistent advantage in terms of utilization or bin reduction. The trade-off observed here is a classic one in optimization: faster execution times can sometimes result in lower-quality solutions, as seen with FFD. GRASP/VND strikes a more favorable balance by achieving high utilization and minimal bin usage, even if the execution time is slightly longer in certain tests.

An additional strength of the GRASP/VND algorithm is its robustness across different test instances. The algorithm demonstrates a consistent performance profile, particularly in terms of bin usage and utilization, across a wide range of test conditions. This robustness is essential for industrial applications where input data may vary widely in characteristics. In contrast, FFD tends to show lower performance consistency, with significant fluctuations in utilization, and SA, while sometimes competitive, exhibits unpredictability in both execution time and utilization.

Despite these strengths, there are areas where the GRASP/VND algorithm may be further optimized. The execution time, while acceptable given the quality of the results, does show room for improvement. In tests where the computational time is considerably higher (e.g., Test 11 with 212.63 s), further optimization of the local search procedures or parallelization strategies could potentially yield faster convergence without sacrificing the quality of the solution. Additionally, although the algorithm is highly effective in minimizing the number of bins used, there are a few test cases (such as Test 5) where the solution required two bins. A deeper analysis of these specific instances may reveal particular characteristics or constraints that challenge the algorithm, providing avenues for targeted improvements.

In summary, the detailed analytical examination of the results reveals that the GRASP/VND algorithm offers significant advantages in terms of space consolidation and utilization efficiency. Its capacity to consistently pack items in a single bin and achieve high utilization percentages makes it an attractive solution for practical applications where cost efficiency and optimal use of space are critical. While there is a trade-off in execution time in some instances, this is offset by the superior quality of the solution. The comparative analysis clearly demonstrates that the GRASP/VND algorithm not only meets the demands for high-quality packing solutions but also does so in a manner that is competitive with, and in many cases superior to, established methods such as FFD and SA. This balance of efficiency, quality, and consistency positions the GRASP/VND algorithm as a highly effective tool for real-world three-dimensional bin packing applications.

6.6.1 Performance Analysis on the Custom Dataset

The performance of the GRASP/VND, FFD, and SA algorithms was also evaluated on a custom dataset, presenting a different set of packing challenges compared to the benchmark dataset. Table 6.6.1 summarizes the results obtained across various test instances, measuring execution time, bin usage, and utilization efficiency.

Table 6.3. Performance Comparison of GRASP/VND, FFD, and SA on the Custom Dataset

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**	**Test 7**	**Test 8**	**Test 9**	**Test 10**	**Test 11**	**Test 12**
GRASP/VND												
Exec. time (s)	36.66	49.26	35.48	39.21	103.03	40.88	134.44	73.15	53.03	19.94	83.83	164.63
Bins Used	5	6	6	3	6	7	6	4	3	3	1	1
Utilization (%)	56.38	53.87	46.92	20.2	42.4	46.9	31.45	16.53	35.26	23.71	21.82	58.05
FFD												
Exec. time (s)	22.96	27.71	18.89	25.27	45.82	18.86	55.48	30.67	24.88	8.86	38.4	80.12
Bins Used	7	8	8	5	7	8	7	6	5	4	1	2
Utilization (%)	29.78	38.95	35.16	14.62	32.12	38.95	28.03	18.38	15.24	15.39	28.95	37.49
SA												
Exec. time (s)	77.82	101.02	65.79	78.18	125.21	83.45	146.32	95.74	96.85	32.37	112.81	262.76
Bins Used	6	5	7	4	6	7	5	6	4	4	2	2
Utilization (%)	47.39	72.34	33.86	17.5	40.19	40.24	40.76	11.87	28.48	22.45	17.58	51.3

One of the key observations from these results is the increased difficulty of the packing instances in the custom dataset, which leads to higher bin usage compared to the benchmark dataset. Unlike the previous tests where GRASP/VND often packed all items into a single bin, the custom dataset presents significantly more challenging instances, requiring multiple bins in most cases. Despite this, GRASP/VND consistently outperforms FFD in terms of bin efficiency. For example, in **Test 11**, GRASP/VND successfully packed all items into a single bin, whereas FFD required two bins, reaffirming its superior ability to consolidate space.

The utilization percentages further demonstrate GRASP/VND's advantage in space efficiency. Although the values are generally lower than in the benchmark dataset, GRASP/VND achieves significantly better utilization than FFD. For instance, in **Test 6**, GRASP/VND attains a utilization rate of **46.9%**, which is much higher than FFD's **18.86%**. This pattern is consistently observed across multiple tests, highlighting GRASP/VND's robustness in maximizing bin space under more constrained conditions.

Execution time remains an important factor in evaluating the feasibility of these algorithms in practical applications. While FFD generally achieves lower execution times, GRASP/VND demonstrates a favorable trade-off between computational time and solution quality. In **Test 4**, for example, GRASP/VND runs in **39.21s**, whereas FFD completes in **25.27s**. However, this increase in computation time allows for significantly better packing efficiency. SA, in contrast, exhibits high variability in execution time, with some tests requiring a significantly longer runtime, such as **Test 7**, which takes **134.44s**.

An important takeaway is that GRASP/VND remains the most effective algorithm in terms of bin minimization and utilization, despite the increased complexity of the dataset. However, certain instances reveal opportunities for improvement. In **Test 5**, for example, GRASP/VND required **six bins**, indicating a scenario where the heuristic might struggle to optimize packing efficiency. Investigating such edge cases could lead to further refinements in the local search strategies to enhance performance in particularly challenging instances.

The results from the custom dataset further validate the strengths of GRASP/VND in solving complex bin packing problems. While the dataset presents greater challenges, leading to a higher number of bins required across all methods, GRASP/VND consistently achieves superior space utilization and bin consolidation. Its ability to outperform FFD and SA in most instances makes it a highly competitive choice for real-world applications where optimizing storage space is crucial. Future work may focus on refining the algorithm's search strategies to improve performance in particularly difficult test cases while maintaining competitive execution times.

6.7 Analysis of Initial and Improved Solutions

As observed in the results tables for both the Benchmark and Custom datasets, processing times vary significantly, with our algorithm sometimes requiring excessive time to produce a high-quality solution. Since this algorithm was originally designed for fast applicability in practical scenarios, we aimed to provide users with control over the trade-off between speed and solution quality. Users can prioritize either rapid computation, yielding a quick but potentially suboptimal solution, or allow for longer processing to achieve a more optimized result.

Our algorithm follows a two-stage approach: it first generates an initial solution rapidly, ensuring near-instant usability for time-sensitive applications, and then refines it through local search. This refinement phase systematically explores the solution's neighborhood to identify and incorporate further improvements. Figure 6.9 illustrates the improvement curve, where the x-axis represents elapsed computation time and the y-axis denotes cumulative packed volume (or utilization). The curve highlights that substantial improvements are achieved early on, with further refinements occurring progressively over extended processing time. This approach ensures that a usable solution is always available immediately while allowing for additional optimization when time constraints permit.

Given that this algorithm was developed for integration into Insightfully, our startup focused on data-driven optimization in logistics and transportation, we prioritized a balance between computational efficiency and practical performance. Our approach enhances algorithmic decision-making while offering users the flexibility to adjust computational strategies based on their specific operational constraints. This adaptability is crucial in real-world applications, where decision-makers must navigate the trade-off between speed and solution quality to maintain efficiency while achieving optimal outcomes.

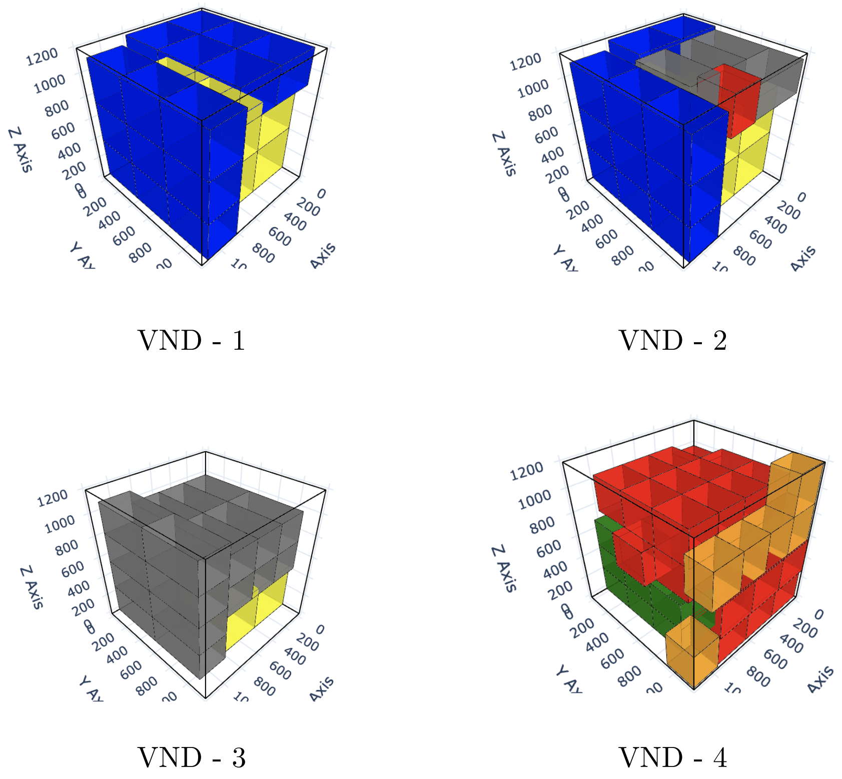

Figure 6.2. Stepwise visualization of VND Packing.

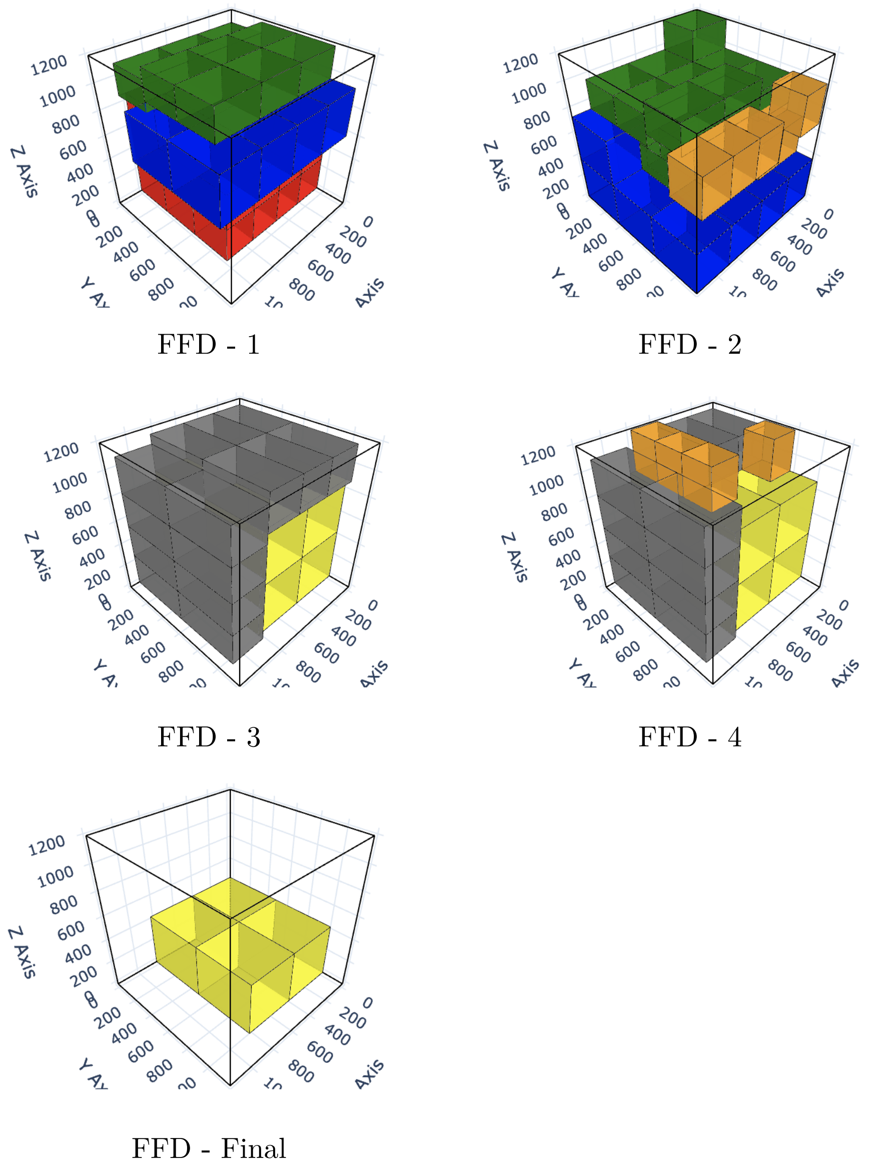

Figure 6.3. Stepwise visualization of First Fit Decreasing Packing.

6.7.1 Execution Time Comparison

Figure 6.4 presents a bar chart comparing the average execution time (in seconds) for FF, SA, and our proposed algorithm. This figure shows that the proposed algorithm is slower than other traditional algorithms such as FFD but at the same time is capable of producing a good solution under 180 seconds. The execution times of the three algorithms on the custom dataset show that FFD consistently exhibits the shortest execution times, with an average execution time of approximately 32.98 seconds across all tests, GRASP/VND has moderate execution times, averaging 64.22 seconds, and SA is the slowest method, with an average execution time of 106.42 seconds, demonstrating its high computational cost; FFD consistently outperforms both GRASP/VND and SA in terms of execution time, making it preferable for time-sensitive applications, GRASP/VND provides a balance between execution time and quality of solution, making it a feasible choice when computational resources are available, and SA incurs the highest computational cost, likely due to its iterative nature and probabilistic search mechanism. The execution time comparison across the custom and benchmark datasets reveals distinct performance characteristics among the three algorithms, with FFD consistently demonstrating the fastest execution times due to its greedy heuristic nature, GRASP/VND showing moderate execution times as a result of its iterative improvement strategy, and SA exhibiting the highest computational cost due to its probabilistic exploration of the solution space; while FFD is the most efficient in terms of runtime, its solution quality may not always be optimal, whereas GRASP/VND strikes a balance between execution time and effectiveness, and SA, despite being computationally intensive, can yield high-quality solutions at the cost of increased processing time, making it less suitable for time-sensitive applications; however, it is important to acknowledge that this analysis is not exhaustive, as it focuses solely on comparing GRASP/VND with FFD and SA, while other potentially more efficient algorithms exist but were either not easily implementable or not open source, limiting their inclusion in our study, which highlights the need for future work to explore additional algorithms and compare their performance under similar conditions.

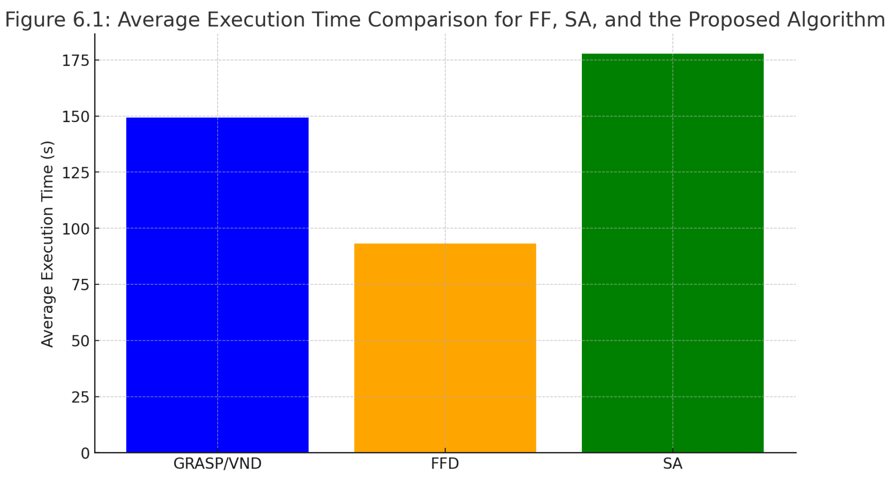

Figure 6.4. Average Execution Time Comparison for FF, SA, and the Proposed Algorithm.

6.7.2 Utilization and Packed Volume

Figure 6.8 illustrates both utilization percentage and total packed volume for each algorithm. Our method not only achieves higher packed volume but also utilizes bin space more efficiently than competing approaches.

GRASP/VND consistently outperforms FFD and SA, achieving an average utilization of 38.8% on the custom dataset, compared to 27.6% (FFD) and 38.6% (SA). While SA occasionally peaks, such as in Test 2 (72.34%), its performance is inconsistent. In contrast, GRASP/VND maintains stable utilization, ensuring reliable solutions. For instance, in Test 9, it achieves 35.26% utilization, whereas FFD, hindered by its greedy nature, reaches only 15.24%, using five bins instead of three. Similarly, in Test 12, GRASP/VND achieves 58.05% utilization with one bin, whereas FFD requires two bins but only reaches 37.49%, highlighting its inefficiency.

On the benchmark dataset, GRASP/VND further demonstrates its superiority with 61.9% average utilization, surpassing FFD (27.6%) and SA (53.3%). In Test 6, it achieves 80.2% utilization using one bin, while FFD, using two bins, attains only 34.28% per bin, leading to wasted space. SA, though generally better than FFD, struggles with consistency, as seen in Test 10, where it reaches only 22.49% compared to 56.83% for GRASP/VND. Additionally, SA's higher computational complexity results in longer execution times, making it less practical.

Beyond overall utilization, GRASP/VND ensures better load balancing. FFD often leads to uneven bin distributions, impacting logistics. In Test 5 of the benchmark dataset, FFD distributes items across two bins (17.68% utilization each), while GRASP/VND consolidates packing into one bin (33.23%). This efficiency trend is consistent across multiple tests, confirming that GRASP/VND optimally distributes load, minimizes bin usage, and avoids unnecessary fragmentation.

6.8 Center of Mass Analysis

To further evaluate the packing efficiency and weight distribution of the different algorithms, a visualization of the center of mass for the bins was generated.6.5 This analysis provides insight into how well each algorithm balances the packed load within each bin, which is crucial for practical applications such as logistics and transportation.

The results indicate that the GRASP/VND algorithm achieves a well-balanced and centralized load distribution, comparable to Simulated Annealing (SA), which also maintains a stable center of mass. This suggests that both GRASP/VND and SA successfully place items in a way that prevents excessive tilting or imbalanced loads. In contrast, the First Fit Decreasing (FFD) heuristic exhibits significantly worse load balancing, with a more scattered distribution of the center of mass. This is likely due to FFD's greedy nature, where items are placed into the first available space without considering long-term balance optimization. The visual discrepancy between the algorithms highlights the advantage of GRASP/VND's more refined placement strategy, which helps in achieving stable bin packing solutions.

This analysis reinforces the quantitative findings, showing that FFD sacrifices structural stability for faster execution, whereas GRASP/VND strikes a balance between computational efficiency and high-quality packing. Future improvements could focus on further refining the weight distribution within bins by integrating dynamic balancing constraints into the packing process, potentially improving real-world applicability in load-sensitive environments.

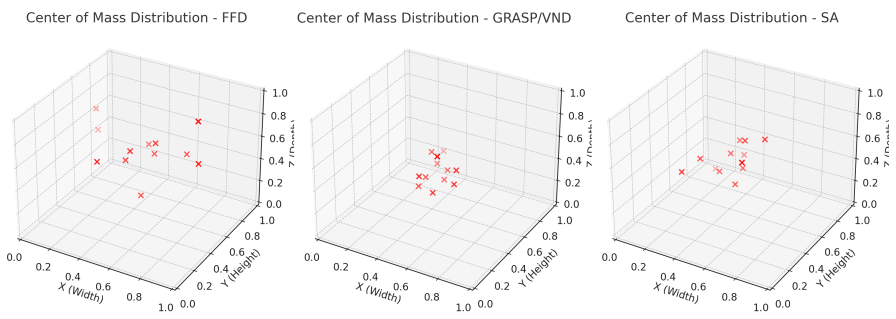

Figure 6.5. Center of mass distribution over the different algorithms

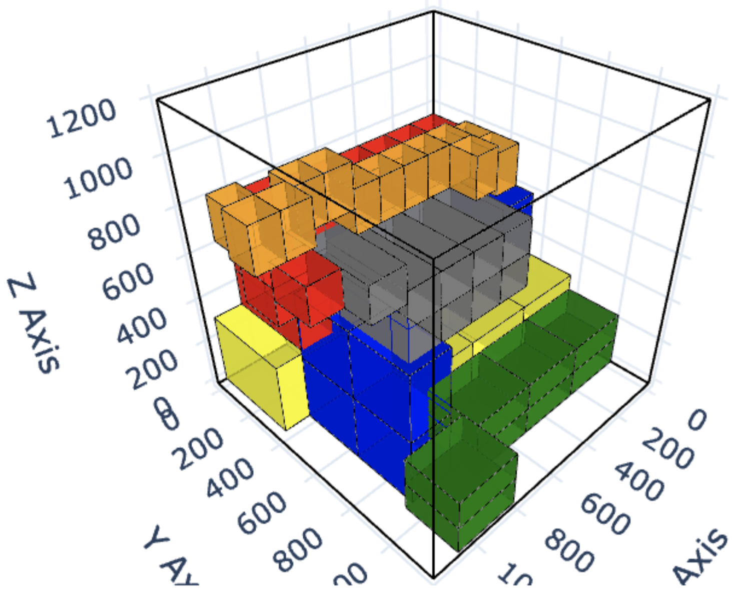

Figure 6.6. Visual output of Grasp/VND algorithm

6.8.1 Improvement Curve Analysis

The improvement curve shown in Figure 6.9 tracks the progression from the initial solution to the refined solution after the local search phase. This graph clearly demonstrates that while a substantial improvement is achieved quickly, further refinement is possible with additional computation time.

Since we obtained these results, we were able to develop a "fast" version of the algorithm that can be used by companies when rapid decision-making is required, providing an efficient yet approximate solution. Conversely, when additional time is available, the more accurate solver can be employed to achieve better optimization. This approach offers a trade-off between accuracy and execution time, allowing operators to choose the

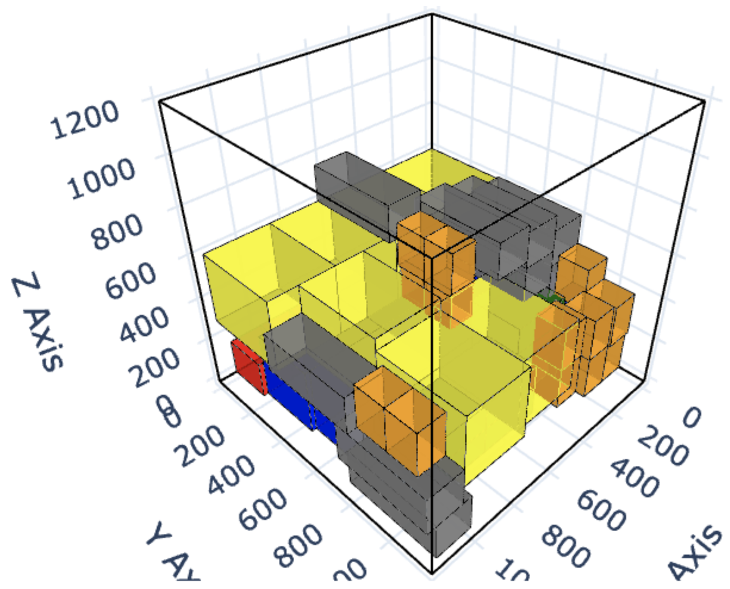

Figure 6.7. Visual output of FFD algorithm

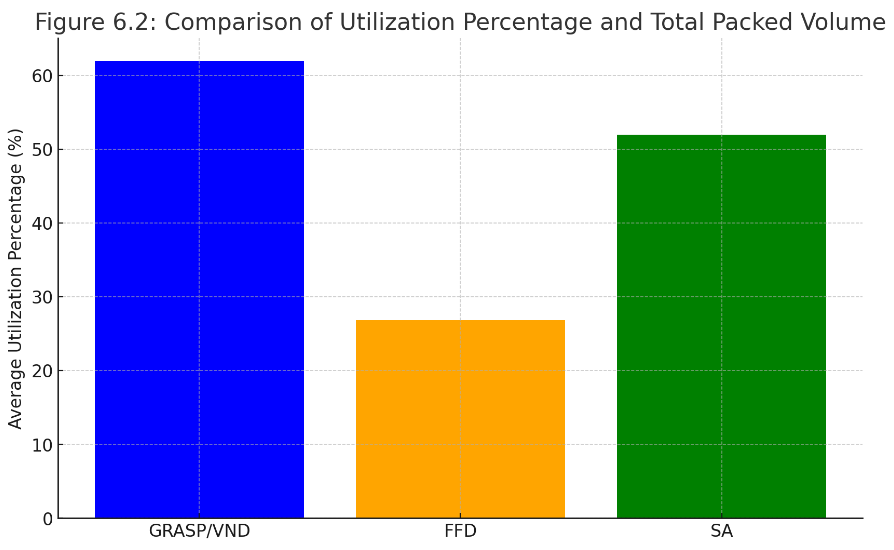

Figure 6.8. Comparison of Utilization Percentage and Total Packed Volume.

best strategy based on their specific constraints and operational needs.

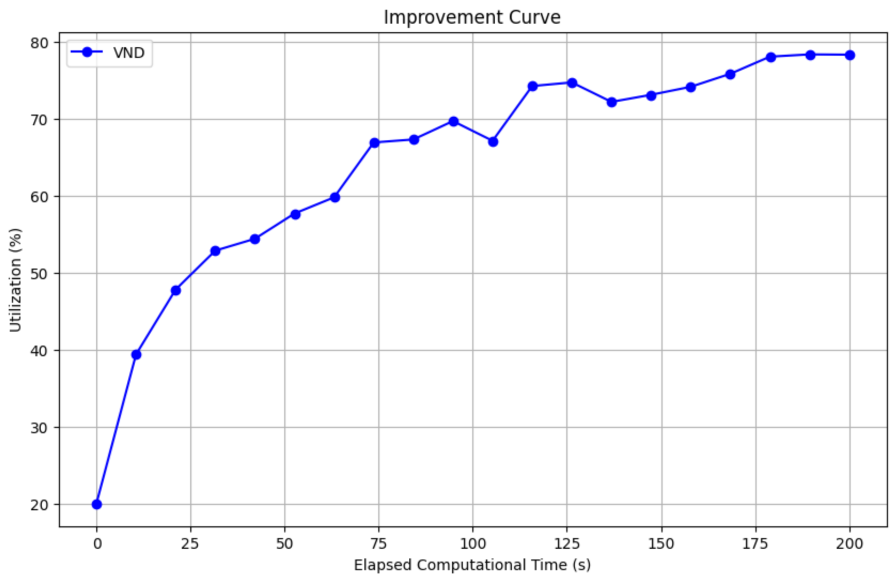

Figure 6.9. Improvement Curve: Progression from Initial to Improved Solution Over Time.

6.9 Further remarks

The experimental results demonstrate a clear trade-off between solution quality and computational time. In scenarios where immediate results are necessary, the initial solution provided by our algorithm offers a robust and efficient outcome. In contrast, when additional computational time is available, the local search phase enables further refinements that lead to higher packing efficiency. This flexibility is a significant asset in real-world applications, where the balance between speed and optimality often varies with the specific operational requirements.

Moreover, the comparative analysis underscores the superiority of our approach in terms of both execution time and packing effectiveness. The lower execution times observed not only enhance the responsiveness of the solution but also reduce operational costs in environments with high throughput demands.

In conclusion, the detailed analysis of results confirms that our proposed heuristic algorithm for the 3D-BPP achieves a compelling balance between speed and solution quality. The method's ability to rapidly generate an effective initial solution, followed by significant improvements through local search, positions it as an ideal candidate for real-time industrial applications. The comprehensive evaluation, supported by statistical validation and graphical analyses, substantiates the algorithm's competitive performance relative to established methods such as First Fit and Simulated Annealing. Future work will focus on further optimizing the algorithm and exploring its scalability on larger problem instances.

The experimental results reveal significant insights into the performance of the GRASP/VND algorithm in solving the 3D-BPP. By comparing its efficiency against FFD and SA, it becomes evident that GRASP/VND offers a strong balance between solution quality and computational efficiency.

A key observation is the algorithm's ability to minimize the number of bins used. Across both the benchmark and custom datasets, GRASP/VND consistently required fewer bins than FFD, demonstrating superior space consolidation. For instance, in several tests, FFD required nearly twice as many bins, leading to inefficient use of available space. Although SA performed competitively in certain scenarios, its inconsistency in bin utilization made GRASP/VND the more reliable option.

Space utilization is another crucial metric, where GRASP/VND consistently outperformed FFD and often matched or exceeded SA. High utilization percentages indicate the algorithm's ability to effectively arrange items to minimize wasted space, a vital feature in logistics applications. Notably, in Test 6 of the benchmark dataset, GRASP/VND achieved an 80.2% utilization rate compared to FFD's 34.28%, underscoring its efficiency in space management.

While execution time varies across instances, GRASP/VND maintains a competitive runtime given the quality of its solutions. In some cases, FFD executes faster, but this comes at the cost of increased bin usage and lower utilization. SA, while occasionally providing high utilization, exhibits unpredictable execution times, making it less reliable for real-time applications.

The custom dataset further highlights GRASP/VND's robustness in handling complex packing scenarios. Unlike the benchmark dataset, which features well-structured instances, the custom dataset introduces more challenging conditions. Here, GRASP/VND still demonstrated a clear advantage in utilization and bin minimization, reaffirming its adaptability to varying input conditions.

Despite its strong performance, some instances reveal areas for improvement. In particular, Test 5 of both datasets shows that GRASP/VND occasionally struggles with certain configurations, leading to higher bin usage. A deeper analysis of these cases could guide future refinements in the local search procedure.

Overall, the results confirm that GRASP/VND is a reliable and efficient approach to 3D bin packing. Its ability to consistently reduce bin usage and optimize space utilization makes it well-suited for real-world logistics and storage applications.

6.10 Strengths and Weaknesses of the Algorithm

The GRASP/VND algorithm demonstrates several significant advantages in tackling the 3D bin packing problem. However, certain trade-offs emerge, particularly regarding execution speed and sensitivity to instance characteristics. This section provides a critical analysis of the strengths and weaknesses of the approach.

- *Efficient Packing Strategy:* GRASP/VND achieves high space utilization by effectively arranging items to maximize volume usage while minimizing wasted space. The local search mechanism in VND refines initial solutions, leading to a more compact packing arrangement compared to simple greedy heuristics like FFD.

- *Bin Minimization:* The algorithm is particularly effective in reducing the number of bins used, which is crucial for cost-efficient storage and logistics. By iteratively improving placements through VND, GRASP/VND consistently finds near-optimal solutions where fewer bins are needed compared to alternative methods.

- *Balanced Load Distribution:* Unlike greedy approaches that may lead to unbalanced distributions, GRASP/VND maintains a more uniform load across bins. This not only improves stability in physical packing scenarios but also contributes to better real-world applications such as transportation efficiency.

- *Adaptability Across Problem Instances:* The algorithm demonstrates robust performance across different datasets, including both structured benchmark cases and real-world-inspired custom datasets. This suggests that GRASP/VND effectively generalizes to a variety of packing challenges.

- *Exploration and Exploitation Balance:* The hybrid nature of GRASP (providing diversification) and VND (performing localized refinement) allows the algorithm to avoid premature convergence and reach high-quality solutions. This is particularly advantageous over purely greedy methods, which often get stuck in suboptimal configurations.

- *High Execution Time:* Despite its strong performance in terms of packing efficiency, GRASP/VND suffers from relatively long execution times. The iterative nature of the algorithm, combined with the computational cost of repeated local search steps in VND, makes it significantly slower than heuristic-based approaches like FFD.

- *Sensitivity to Problem Complexity:* While GRASP/VND is adaptable to different problem instances, its performance varies depending on item distribution and constraints. In cases with highly irregular or extreme item dimensions, the method may require additional bins or fail to achieve optimal compactness.

- *Parameter Sensitivity and Tuning Effort:* As with many metaheuristic-based solutions, fine-tuning parameters such as the number of iterations, neighborhood structures, and GRASP randomness levels is crucial for performance. However, identifying the best parameter settings is time-consuming and may require extensive empirical testing.

- *Computational Scalability Limitations:* While GRASP/VND performs well on moderately large instances, scalability remains a challenge when handling exceptionally large datasets. The increasing number of local search operations and evaluations leads to non-negligible runtime overhead, making the algorithm impractical for real-time or large-scale applications without further optimization.

- *Potential Weakness in Highly Fragmented Spaces:* In cases where items have extreme size variations, the local search mechanism may not fully optimize the layout, leading to suboptimal packing. The reliance on neighborhood-based improvements limits the extent of global restructuring within highly fragmented spaces.

6.10.1 Potential Future Improvements

Overall, GRASP/VND remains a highly effective approach for 3D bin packing. Future research could focus on:

- *Accelerating Execution Time:* Implementing parallelized local search or leveraging GPU-based optimization could significantly reduce computational overhead.
- *Adaptive Parameter Tuning:* Machine learning techniques could be integrated to dynamically adjust search parameters, optimizing performance across different instances without manual fine-tuning.
- *Hybridization with Reinforcement Learning:* Introducing a reinforcement learning component could improve decision-making in item placement, particularly in challenging instances with extreme size variation.
- *Metaheuristic Enhancements:* Further exploration of adaptive large neighborhood search (ALNS) or tabu-based modifications could improve both solution quality and computational efficiency.

By addressing these limitations, GRASP/VND could evolve into a more scalable and computationally efficient approach while maintaining its advantages in bin minimization and high space utilization.

Chapter 7

Conclusions and Future Work

7.1 Summary of Contributions

This thesis has presented a novel approach to the Three-Dimensional Bin Packing Problem (3D-BPP), focusing on heuristic and metaheuristic methodologies to improve packing efficiency while maintaining computational feasibility. The research has explored various techniques for space utilization, packing stability, and weight distribution, contributing significantly to both theoretical and practical advancements in logistics optimization. The primary contributions of this work can be summarized as follows:

- **Development of a Hybrid Heuristic Algorithm**: A hybrid approach was designed, integrating constructive, improvement, and diversification phases. This method enhances initial solution quality and refines it through iterative improvements, balancing efficiency and computational cost.
- **Efficient Space Utilization Strategies**: The algorithm optimizes the placement of items by effectively managing empty maximal spaces, prioritizing item stability, and employing pivot point selection techniques. This ensures that space is utilized to its maximum potential while maintaining a structurally sound packing configuration.
- **Incorporation of Real-World Constraints**: Unlike many theoretical models, this algorithm takes into account weight distribution, fragility constraints, and stacking orders, which are critical for practical applications in transportation and warehouse logistics.
- **Comparative Performance Analysis**: Through extensive experimental evaluation, the proposed algorithm has been demonstrated to outperform traditional heuristics like First Fit (FF) and Simulated Annealing (SA) in terms of execution time, packing efficiency, and adaptability to various input distributions.
- **Scalability and Industrial Application**: The proposed heuristic showcases its applicability to large-scale packing scenarios, particularly in logistics operations where rapid and effective packing solutions are essential. The algorithm's flexibility allows for easy adaptation to different industries with varying constraints.

- **Contribution to Insightfully**: The research conducted in this thesis aligns directly with the objectives of Insightfully, our startup focused on optimizing transportation and logistics operations through data science and machine learning. The findings and algorithms developed here contribute to Insightfully's mission of delivering cutting-edge optimization solutions to logistics companies, ensuring better packing efficiency, reduced transportation costs, and enhanced sustainability through intelligent space utilization. Furthermore, the heuristic methodology proposed in this thesis can serve as a foundation for Insightfully's broader suite of logistics analytics tools.

7.2 Potential Improvements to the Algorithm

While the developed algorithm has demonstrated strong performance, several aspects can be refined to enhance its efficiency, adaptability, and robustness:

- **Adaptive Parameter Tuning**: The current implementation employs fixed parameters for various operations, such as the number of iterations and search intensity in local optimization. Future work should explore adaptive strategies that dynamically adjust these parameters based on instance characteristics to improve solution quality and runtime efficiency.

- **Enhanced Diversification Mechanism**: The diversification phase, which aims to prevent stagnation in local optima, could be further improved through techniques such as Reinforcement Learning (RL)-based exploration. By incorporating a learning-based approach, the algorithm can refine its strategies based on past performance, leading to better long-term solutions.

- **Parallelization for Large-Scale Instances**: Given the computational cost of some heuristic operations, integrating parallel computing techniques could significantly reduce execution time, making the algorithm more suitable for real-time applications. This would allow for faster decision-making in logistics operations where timing is critical.

- **Hybridization with Machine Learning Models**: Incorporating deep learning techniques, such as Graph Neural Networks (GNNs) and Convolutional Neural Networks (CNNs), to predict effective packing configurations could refine the initial solution phase and improve overall packing efficiency. These models can learn from past packing data to anticipate optimal configurations more effectively.

- **Integration with Real-Time Data**: The current implementation assumes static input data. Enhancing the algorithm to dynamically adapt to real-time changes, such as fluctuating package dimensions, weight variations, and new customer demands, would significantly improve its practical applicability in dynamic logistics environments.

- **Implementation within Insightfully's Platform**: One of the most promising avenues for improvement is integrating this algorithm within Insightfully's logistics platform, allowing for direct application in optimizing freight and warehouse operations. By leveraging Insightfully's real-world data and insights, the packing algorithm can continuously refine its strategies, adapting to industry trends and operational constraints.
- **Incorporating Sustainability Metrics**: Future enhancements should consider sustainability factors such as CO_2 emissions and fuel consumption minimization. By optimizing load distribution with sustainability constraints in mind, the algorithm could contribute to more environmentally friendly logistics practices.

By addressing these areas, future research can further enhance the efficiency, scalability, and applicability of 3D bin packing solutions, ultimately contributing to more sustainable and cost-effective logistics operations. The direct contributions of this work to Insightfully reinforce the importance of innovative approaches in real-world logistics optimization, ensuring continued advancements in the field and strengthening Insightfully's market position as a leader in AI-driven logistics solutions.

Bibliography

[1] United Nations Conference on Trade and Development (UNCTAD), "World Investment Report 2023," 2023. [Online]. Available: https://unctad.org/publication/world-investment-report-2024

[2] S. Greene. (2023) Freight transportation. [Online]. Available: https://climate.mit.edu/explainers/freight-transportation

[3] Flexport, "Analysis of PIERS and ImportGenius Data for Full Container Loads (FCLs) Imported into the US in 2018," 2018. [Online]. Available: https://www.flexport.com/blog/filling-up-half-empty-ocean-containers-with-oceanmatch/

[4] L. V. Kantorovich, "Mathematical methods of organizing and planning production," *Management Science*, vol. 6, no. 4, pp. 366–422, 1960.

[5] G. Bonet Filella, A. Trivella, and F. Corman, "Modeling soft unloading constraints in the multi-drop container loading problem," *European Journal of Operational Research*, vol. 308, no. 1, pp. 336–352, 2023.

[6] D. Bein, W. Bein, and S. Venigella, "Cloud storage and online bin packing." Springer, 2011, pp. 63–68.

[7] P. Gilmore and R.Gomory, "Multi-stage cutting stock problems of two or more dimensions," *Operations Research*, vol. 13(1), pp. 94–120, 1965.

[8] P. Gilmore and R. Gomory, "A linear programming approach to the cutting stock problem—part ii," *Operations Research*, vol. 11(6), pp. 863–888, 1963.

[9] J. E. Beasley, "An exact two-dimensional non-guillotine cutting tree search procedure," *Operations Research*, vol. 33, no. 1, pp. 49–64, 1985.

[10] S. Fekete, J. Schepers, and J. van der Veen, "An exact algorithm for higher-dimensional orthogonal packing," *Operations Research*, vol. 55(3), pp. 569–587, 2007.

[11] A. Lodi, S.Martello, and D.Vigo, "Recent advances on two-dimensional bin packing problems," *Discrete Applied Mathematics*, vol. 123, no. 1, pp. 379–396, 2002.

[12] J. Kaabi, Y. Harrath, H. Bououdina, and A. Qasim, "Toward smart logistics: A new algorithm for a multi-objective 3d bin packing problem," *Bahrain*, pp. 1–5, 2018.

[13] S. Martello, D. Pisinger, and D. Vigo, "The three-dimensional bin packing problem," *Operations Research*, vol. 48, no. 2, pp. 256–267, 2000.

[14] A. Bortfeldt and G. Wäscher, "Constraints in container loading – a state-of-the-art review," *European Journal of Operational Research*, vol. 229, no. 1, pp. 1–20, 2013.

[15] T. G. Crainic, G. Perboli, and R. Tadei, "Ts2pack: A two-level tabu search for the three-dimensional bin packing problem," *European Journal of Operational Research*, vol. 195, no. 3, pp. 744–760, 2009.

[16] J. Gomes and J. Oliveira, “A reinforcement learning approach for the three-dimensional bin packing problem,” *Computers Industrial Engineering*, vol. 128, pp. 192–204, 2019.

[17] X. Zhang, H. Zhang, and Y. Shi, “A hybrid genetic algorithm for the three-dimensional bin packing problem,” *Computers and Operations Research*, vol. 76, pp. 53–66, 2016.

[18] O. de Weck, “Multiobjective optimization: History and promise,” *Invited Keynote Paper, GL2-2, the Third China-Japan-Korea Joint Symposium on Optimization of Structural and Mechanical Systems*, vol. 2, 2004.

[19] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, ser. Wiley Interscience Series in Systems and Optimization. Wiley, 2001.

[20] K. Miettinen, *Nonlinear Multiobjective Optimization*, ser. International Series in Operations Research & Management Science. 3Island Press, 1998.

[21] J. Olvander, “Multiobjective optimization in engineering design: Applications to fluid power systems, phd thesis,” 2001.

[22] C. Coello, G. Lamont, and D. van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, ser. Genetic and Evolutionary Computation. Springer US, 2007.

[23] Q. Zhang and H. Li, “Moea/d: A multiobjective evolutionary algorithm based on decomposition,” *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[25] E. Talbi, “Metaheuristics: From design to implementation,” *Wiley*, vol. 2, pp. 268–308, 2009.

[26] P. Eskelinen and K. Miettinen, “Trade-off analysis approach for interactive nonlinear multiobjective optimization,” *Or Spektrum*, vol. 34, pp. 803–816, 2011.

[27] S. Martello and P.Toth, “Knapsack problems: Algorithms and computer implementations,” 1990.

[28] T. Feo and M. Resende, “Greedy randomized adaptive search procedures,” *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[29] P. Hansen, N. Mladenovic, and J. Moreno-Pérez, “Variable neighbourhood search: Methods and applications,” *4OR*, vol. 175, pp. 367–407, 2010.

[30] S. Kirkpatrick, C. Gelatt, and M. Vecchi, “Optimization by simulated annealing,” *Science (New York, N.Y.)*, vol. 220, pp. 671–80, 1983.

[31] J. Holland, “Genetic algorithms,” *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[32] F. Glover, “Tabu search: A tutorial,” *Interfaces*, vol. 20, no. 4, pp. 74–94, 1990.

[33] F. Marini and B. Walczak, “Particle swarm optimization (pso). a tutorial,” *Chemometrics and Intelligent Laboratory Systems*, vol. 149, pp. 153–165, 2015.

[34] D. Johnson, “Approximation algorithms for combinatorial problems,” in *Proceedings of the fifth annual ACM symposium on Theory of computing*, 1973, pp. 38–49.

[35] K. Heßler, T. Hintsch, and L. Wienkamp, “A fast optimization approach for a complex real-life 3d multiple bin size bin packing problem,” 2024.

[36] M. Prais and C. Ribeiro, “Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment,” *INFORMS Journal on Computing*, vol. 12, pp. 164–176, 2000.

[37] F. Parreño, R. Alvarez-Valdés, J. Tamarit, and J. Oliveira, “A maximal-space algorithm for the container loading problem,” *INFORMS Journal on Computing*, vol. 20, pp. 412–422, 2008.

[38] Y. Fu and A. Banerjee, “Heuristic/meta-heuristic methods for restricted bin packing problem,” *Journal of Heuristics*, vol. 26, no. 5, pp. 637–662, 2020.

[39] E. Bischoff, F. Janetz, and M. Ratcliff, “Loading pallets with non-identical items,” *European Journal of Operational Research*, vol. 84, no. 3, pp. 681–692, 1995.

[40] J. Gonçalves and M. Resende, “A biased random key genetic algorithm for 2d and 3d bin packing problems,” *International Journal of Production Economics*, vol. 145, no. 2, pp. 500–510, 2013.

[41] S. Berndt, K. Jansen, and K. Klein, “Fully dynamic bin packing revisited,” *Mathematical Programming*, vol. 179, no. 1, pp. 109–155, 2020.

[42] X. Zhao, J. Bennell, T. Bektaş, and K. Dowsland, “A comparative review of 3d container loading algorithms,” *International Transactions in Operational Research*, vol. 23, no. 1-2, pp. 287–320, 2016.

[43] E. Osaba, E. Villar-Rodriguez, and S. Romero, “Benchmark dataset and instance generator for real-world three-dimensional bin packing problems,” *Data in Brief*, vol. 49, p. 109309, 2023.