POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Hierarchical Option-Based Reinforcement Learning for Efficient Three-Dimensional Bin Packing

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Fausto Allegrini**

Student ID: 252938
Advisor: Prof. Marcello Restelli
Co-advisors: Davide Tenedini
Academic Year: 2024-25

"Entia non sunt semotus praeter necessitatem."

— Occam's Rasor

Abstract

The 3D Bin Packing Problem (3D-BPP) is a well-known combinatorial optimization challenge with significant implications for operational efficiency in the logistics and shipping industries. Effective palletization requires rapid configuration, high volumetric utilization, and strict adherence to structural constraints. While Deep Reinforcement Learning (DRL) has emerged as a viable approach to solving such tasks, integrating it with rigorous industrial and physical constraints remains a persistent challenge. This thesis addresses this gap by first providing a systematic review and taxonomy of existing DRL applications within the 3D-BPP domain. The literature is categorized based on state and action space representations, reward shaping techniques, algorithmic frameworks, stability assessments, and the overall scope of applicability regarding grid resolution.

Building on this analysis, we propose a novel architecture employing Hierarchical Reinforcement Learning (HRL) to decompose the 3D problem into layer-building subtasks. To ensure practical applicability, we introduce the Cornered Anchored Regions (CAR) method, which efficiently reduces the action space while maintaining sufficient exploration capabilities and ensuring the static stability of each sequential loading position.

The model is trained exclusively to construct a single layer of items. When tested on a real-world industrial dataset, the model successfully generates a complete layer in 79.22% of the instances. Furthermore, the model is tested in a full-bin packing scenario, where it achieves physical stability in 99.9% of the cases, with an average volumetric utilization of 64.21%, surpassing traditional heuristic methods by more than 10%, although a minor deficit is observed relative to an established Genetic Algorithm (GA) baseline deployed in real industrial palletization. As a competitive advantage, the model generates a loading sequence in 2.5 seconds per order, an acceleration of nearly 300 compared to the GA.

These results demonstrate that the proposed approach offers a robust alternative for industrial automated palletization capable of achieving comparable performance at a fraction of the inference computational cost.

Keywords: Industrial Palletization, Three-dimensional Packing Problem, Hierarchical Deep Reinforcement Learning, Packing Constraints.

Abstract in lingua italiana

Il problema della pallettizzazione tridimensionale (3D-BPP) rappresenta una sfida di ottimizzazione combinatoria di rilievo per l'efficienza operativa nei settori della logistica e delle spedizioni. Un'efficace pallettizzazione richiede una configurazione rapida, un'elevata utilizzazione volumetrica e una rigorosa aderenza ai vincoli strutturali.
Deep Reinforcement Learning (DRL) è emerso come un approccio valido per tali compiti; tuttavia, la sua integrazione con i vincoli industriali resta una sfida persistente. Tale lacuna viene affrontata nella presente tesi fornendo, in primo luogo, una revisione sistematica e una tassonomia delle applicazioni DRL esistenti nel dominio del 3D-BPP. La letteratura è categorizzata in base alle rappresentazioni degli spazi di stato e di azione, alle tecniche di *reward shaping*, ai framework algoritmici, alle valutazioni di stabilità e all'ambito generale di applicabilità relativo alla risoluzione della griglia.
Sulla base di tale analisi, viene proposta una nuova architettura che impiega Hierarchical Reinforcement Learning (HRL) per scomporre il problema 3D in sottocompiti di costruzione di strati (*layer-building*). Al fine di garantire l'applicabilità pratica, si introduce il metodo delle *Cornered Anchored Regions* (CAR), mediante il quale lo spazio delle azioni viene ridotto, mantenendo capacità di esplorazione e garantendo la stabilità statica del pallet.
L'addestramento è eseguito esclusivamente per la costruzione di un singolo strato di colli. A seguito dei test su un dataset industriale reale, uno strato completo viene generato con successo nel 79,22% dei casi. Inoltre, la valutazione viene estesa a uno scenario di imballaggio completo, in cui la stabilità fisica è mantenuta nel 99,9% dei casi, con un'utilizzazione volumetrica media del 64,21%, superando l'efficacia di un metodo euristico di confronto di oltre il 10%. Sebbene sia osservato un lieve deficit rispetto a una consolidata baseline basata su Algoritmo Genetico (GA) impiegata in ambito industriale reale, il modello HRL mostra un forte vantaggio competitivo nella velocità di generazione delle sequenze di carico (circa 2,5 secondi per ordine), con un'accelerazione di quasi un fattore 300 rispetto al GA. Tali risultati dimostrano che l'approccio proposto è una solida alternativa per la pallettizzazione automatizzata industriale.
Parole chiave: Pallettizzazione Industriale, Problema di Pallettizzazione Tridimensionale, Apprendimento per Rinforzo Gerarchico, Vincoli di Pallettizzazione.

Contents

1 | Introduction

The 3D Bin Packing Problem (3D-BPP) is a fundamental combinatorial optimization challenge, formally classified as NP-hard [36]. Its applicability within the logistics and supply chain industries is extensive, where the efficient and compact packing of goods directly impacts operational costs and throughput. While numerous variations of this problem exist, the core objective remains consistent: packing a set of smaller rectangular items into one or more larger rectangular containers. This study specifically addresses a formulation known as the Single Knapsack Problem [1, 11], where it is required to pack an inventory of items in a single bin, with the primary objective of maximizing the bin's volume utilization.

Historically, a wide range of heuristics, meta-heuristics, exact methods, and search algorithms have been developed to compute approximate solutions for this problem [1, 88]. Pure heuristic algorithms operate with high computational efficiency but often yield suboptimal packing densities, as they rely heavily on rigidly handcrafted rules designed by human experts. Conversely, meta-heuristic approaches, such as genetic algorithms, simulated annealing, and tree search methodologies, are capable of discovering effective packing configurations. However, these methods typically incur high computational costs, making them prohibitively time-consuming for large-scale instances, and their performance remains highly sensitive to empirically designed search rules.

To address the limitations of traditional approaches, recent research has increasingly shifted toward learning-based methods, specifically applying Deep Reinforcement Learning (DRL) to the 3D-BPP [49, 61, 88, 105, 119, 123, 124]. While DRL requires training deep neural networks across a vast distribution of problem instances, it offers a significant advantage during inference, consistently generating solutions of reasonably high quality in a much shorter time without relying on exhaustive search.

Building upon these recent advancements, this thesis focuses on solving the offline 3D-BPP by integrating DRL with deterministic geometric heuristics. The primary objective is to develop a robust framework suitable for real-world industrial applications. Consequently, the proposed solution strictly prioritizes the physical feasibility of the packing sequence,

ensuring both the structural stability of the loaded items and the kinematic reliability of the robotic execution.

2 | Preliminaries

This chapter establishes the theoretical framework for the research by defining the Three-Dimensional Packing Problem (3D-BPP) and the main tools used in this thesis.

2.1. Problem Formalization

The Three-Dimensional Bin Packing Problem (3D-BPP) is defined as the optimal arrangement of a set of cuboid items, commonly referred to as "boxes" or "cargo", within one or more larger rectangular enclosures, designated as "bins" or "containers". The primary objective is typically the minimization of the total number of bins required or the maximization of the volume occupied within each bin. This problem is NP-hard, implying that no exact polynomial-time algorithm is known to find an optimal solution for large-scale instances [36, 71].

The 3D-BPP represents a spatial generalization of two foundational combinatorial optimization problems:

- **1D Bin Packing (1D-BPP):** The assignment of items with specific weights or lengths into bins of fixed capacity [77].
- **2D Packing Problem (2D-BPP):** The arrangement of rectangular shapes within a larger two-dimensional area, often encountered in cutting-stock applications [35].

Industrial relevance is particularly pronounced in the logistics and manufacturing sectors, where the efficiency of palletization and container loading directly impacts operational costs and carbon footprints.

2.1.1. Taxonomy of 3D-BPP

A standardized typology for packing problems was originally proposed by Dyckhoff [28] and later on refined by Wäscher et al. [13, 113]. This classification is predicated on the relationship between the items to be packed and the available containers.

The entities involved are categorized as follows:

- **Small Items (Cargo):** The individual units to be allocated.
- **Large Objects (Bins):** The containers designated to host the cargo.

Based on the heterogeneity of these objects, the problem is further distinguished into:

1. **Identical:** All objects share the same dimensions.
2. **Weakly Heterogeneous:** Objects can be clustered into a limited number of classes, each containing a significant quantity of identical items.
3. **Strongly Heterogeneous:** Most or all items possess unique dimensions.

Fundamental to this taxonomy is the distinction between two primary assignment logic types:

- **Input (value) Minimization:** All items must be accommodated into a finite number of bins, and the objective is to minimize the resources (bins) utilized.
- **Output (value) Maximization:** The number of containers is fixed, and the goal is to maximize the value or volume of the cargos packed.

Different packing problems can be classified under these two classes [1], as illustrated in Table 2.1.

A critical distinction in 3D-BPP literature concerns the temporal availability of input data, which dictates the decision-making strategy of the packing agent.

Off-line and On-line Paradigms. In *off-line* problems, all input data is available from the outset, allowing decisions to be made with complete information. Although global optimality is theoretically reachable given infinite computational resources, the NP-hard nature of the 3D-BPP necessitates the use of sub-optimal solvers in practical applications. Off-line solutions typically represent the upper bound for packing quality due to the full knowledge of the sequence [1].

Conversely, in *on-line* problems, the sequence of the cargos is predefined and must be loaded one at a time following the exact order. Berndt et al. [11] proposed a taxonomy for on-line 3D-BPPs by categorizing algorithms according to their *deletion* and *repacking* capabilities, as summarized in Table 2.1 by [1].

Semi-on-line and Look-ahead Strategies. Between these two extremes lies the *semi-on-line* environment. This class relaxes pure on-line constraints by introducing pre-processing operations or temporary storage. Common implementations include:

Characteristics of large objects		Assortment of small items		
		Identical	Weakly heterogeneous	Strongly heterogeneous
All dimensions fixed	One large object	Identical Item Packing Problem (IIPP)	Single Large Object Placement Problem (SLOPP)	Single Knapsack Problem (SKP)
	Identical		Multiple Identical Large Object Placement Problem (MILOPP)	Multiple Identical Knapsack Problem (MIKP)
	Heterogeneous		Multiple Heterogeneous Large Object Placement Problem (MHLOPP)	Multiple Heterogeneous Knapsack Problem (MHKP)

(a) Problems of Input (value) Minimization type

Characteristics of large objects		Assortment of small items	
		Weakly heterogeneous	Strongly heterogeneous
All dimensions fixed	Identical	Single Stock-Size Cutting Stock Problems (SSSCSP)	Single Bin-Size Bin Packing Problems (SBSBPP)
	Weakly heterogeneous	Multiple Stock-Size Cutting Stock Problems (MSSCSP)	Multiple Bin-Size Bin Packing Problems (MBSBPP)
	Strongly heterogeneous	Residual Cutting Stock Problems (RCSP)	Residual Bin Packing Problems (RBPP)
One large object variable dimensions		Open Dimension Problems (ODP)	
		(ODP/W)	(ODP/S)

(b) Problems of Output (value) Maximization type

Figure 2.1: Schema of the possible packing problems defined by [113]. Table taken from [1].

Name	Deletion	Repacking
On-line bin packing	✗	✗
Dynamic bin packing	✓	✗
Relaxed on-line bin packing	✗	✓
Fully dynamic bin packing	✓	✓

Table 2.1: Overview of on-line models presented by [11].

- **Buffering:** Items are temporarily held in a physical or virtual buffer before final placement [30].
- **Window Visibility:** A fixed-size window of upcoming items is visible to the agent, allowing for local optimization of the packing sequence.

Spatial Constraints: Bounded and Unbounded Environments. Parallel to cargos visibility, a crucial parameter is the container availability. Packing problems are thus divided into:

1. **Unbounded Environments:** An unlimited supply of bins is available. The objective is focused on global minimization of used resources.
2. **Bounded Space Model:** Only a finite and restricted number of bins can be "active" or open simultaneously.

The bounded model is particularly relevant in industrial contexts where physical space at loading docks or conveyor throughput limits the number of pallets that can be processed concurrently.

2.1.2. Constraints for 3D-BPP

Solving the 3D-BPP requires satisfying a multitude of restrictions that extend beyond simple volumetric fitting. The literature classifies these restrictions into three fundamental categories: *Geometric Constraints*, which define the basic feasibility of the packing; *Safety Constraints*, which ensure the physical integrity of the cargo and the container; and *Logistic Constraints*, which address operational and delivery requirements [1]. While geometric constraints are inherent to the definition of the problem, the classification of practical constraints into Safety and Logistic groups was formally structured by Ramos et al. [91] to provide a clearer relationship between physical limitations and operational decisions. In the following paragraphs, a brief overview of all constraints based on [1]

Geometric Constraints

Geometric constraints constitute the necessary conditions for any valid solution. They are universally present in all 3D-BPP variants:

- **Containment:** All packed items must lie entirely within the boundaries of the container dimensions.
- **Non-overlapping:** No two items can occupy the same spatial volume; the intersection of their occupied coordinates must be empty.
- **Orthogonality:** Items must be placed with their edges parallel to the container's axes.

Safety Constraints

Safety constraints are critical in industrial applications to prevent damage to the cargo and to ensure the safety of workers and the transportation unit [91].

Weight Limit Constraint. The total weight of the packed items must not exceed the maximum payload capacity of the container or vehicle. This is typically treated as a hard constraint in the literature.

Weight Distribution Constraint. The weight of the loaded items must be distributed evenly across the container. While simplified approaches attempt to align the center of gravity (CoG) of the load with the geometric center of the container, more rigorous approaches utilize *Load Distribution Diagrams* (LDD) to respect specific axle load limits required by transportation regulations [91].

Orientation Constraint. Items may be restricted in their rotation due to their contents (e.g., liquids, fragile goods). While a cuboid theoretically allows for six orthogonal orientations, practical instances can limit rotation to the vertical axis (z-axis).

Stacking Constraint. Also referred to as the load-bearing constraint, this ensures that items are not damaged by the weight of other items placed on top of them. Constraints are often modeled by defining a maximum weight per unit area a box can support or by prohibiting the placement of heavier boxes on top of lighter or fragile ones.

Physical Positioning Constraint This constraint restricts the placement of specific items to certain areas within the container based on their physical properties (e.g., size, weight, or content). For instance, heavy items may be required to be placed on the floor to lower the center of gravity.

Stability Constraint. Stability ensures the cargo remains stationary during loading and transport. It is generally categorized into:

- *Static (Vertical) Stability*: Ensures items do not fall under gravity during loading. It typically requires that the bottom face of an item is supported by the floor or other items, either fully or by a minimum percentage.
- *Dynamic (Horizontal) Stability*: Prevents items from shifting longitudinally or laterally due to inertia [91].

Logistic Constraints

Logistic constraints address operational decisions unrelated to the physical properties of the cargo, focusing instead on the efficiency of the delivery and handling process [91].

Loading Priorities Constraint. In scenarios where not all items can be packed (Output Maximization) or where delivery deadlines differ, items are assigned a priority level. This ensures that high-priority goods are included in the shipment or placed in accessible positions.

Complete Shipments Constraint. This constraint mandates that for a specific order, either all items are loaded into the same shipment, or none are. This prevents the fragmentation of orders, relevant for ensuring customer satisfaction.

Allocation Constraint. Applicable in multiple container scenarios, this constraint dictates whether specific sets of items must be packed together or separated. A common application is the segregation of incompatible goods (e.g., food and chemicals) into different containers.

Customer Positioning Constraint. Also known as the grouping constraint, this requires items belonging to the same customer or destination to be packed together or, under lenient conditions, in close proximity. This facilitates the unloading process and is essential for multi-drop delivery routes.

Pattern Complexity Constraint. This constraint limits the complexity of the packing arrangement to ensure it can be implemented by human operators or robotic systems. Complex patterns may increase loading time or exceed robotic capabilities.

2.2. Terminology

This section introduces the fundamental terms and concepts used throughout the thesis. A precise terminology is essential to avoid ambiguity in the formal definition of the Three-Dimensional Bin Packing Problem (3D-BPP) and in the description of the Reinforcement Learning environment developed for this study.

- **Item** — A three-dimensional cuboid defined by its geometric attributes: *length*, *width*, and *height*. Each item represents a single physical instance that must be placed within a container according to its spatial dimensions and allowed orientations.
- **Article** — A class of items that share identical dimensional properties. Each article is associated with a unique identifier and represents the abstract type to which multiple identical items may belong. While an item is a specific instance, an article defines its standard size category.

- **Order** — A collection of N items potentially distributed across M distinct articles, with $M \leq N$. An order specifies both the total quantity of items and their classification according to article types.

2.3. Reinforcement Learning

This section outlines the mathematical foundations of Reinforcement Learning (RL) [99] as the main tool utilized in this research. First, we introduce Markov Decision Processes (MDPs), which model decision-making in discrete, atomic time steps. MDP provides a formal structure for the definition of RL. Then, the concept of temporal abstraction is introduced, extending MDPs to Semi-Markov Decision Processes (SMDPs) to account for variable-duration actions. Finally, the Options Framework is detailed as the mathematical bridge connecting MDPs and SMDPs, providing the structural basis upon which the proposed Hierarchical Reinforcement Learning (HRL) strategies are built.

2.3.1. Markov Decision Processes (MDP)

A Markov Decision Process (MDP) is a discrete-time stochastic control process that serves as the fundamental mathematical framework for modeling decision-making under uncertainty; first formalized by Puterman [87].
The principles of sequential optimization had already been established by Bellman [10]; the novelty of MDPs is the integration of these principles with the "Markov Property": the dynamics of the system are memoryless, i.e., it does not depend on the system's history but only on its present state. Formally,

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s_{t+1} \mid s_t, a_t).$$

In this paradigm, the agent interacts with the environment in a continuous feedback loop: at each time step, the agent observes the current state of the system and selects an action. This action acts as a control input that influences not only the immediate utility (reward), but also the probabilistic transition to the next state, affecting all future decision opportunities [87, 99].

Formally, an MDP is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the set of states representing the environment's configuration.
- $\mathcal{A}$ is the set of actions available to the agent.

- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state-transition probability function, denoted as $P(s'|s, a)$, representing the probability of transitioning to state s' given action a in state s.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $R(s, a)$ denotes the immediate reward received after executing action a in state s.
- $\gamma \in [0, 1]$ is the discount factor, balancing the importance of immediate versus future rewards.

At each time step, an action is taken based on a policy $a \sim \pi(\cdot \mid s)$, with $\pi(a \mid s)$ denoting the probability of selecting action a under state s.

The primary objective within an MDP framework is to compute an optimal policy π^* that maximizes the expected cumulative discounted reward. Formally, this goal is expressed as finding the policy that maximizes the functional $J(\pi)$:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right],$$

where $\tau = (s_0, a_0, s_1, \ldots)$ represents a trajectory generated by the interaction between the agent following policy π and the environment dynamics [99].

To navigate the search space efficiently, the quality of a policy is evaluated using two fundamental functions:

- The **State-Value Function** $V_\pi(s)$, which estimates the expected return starting from state s following policy π.
- The **Action-Value Function** $Q_\pi(s, a)$, which estimates the expected return starting from state s, taking action a, and following policy π thereafter.

The closed-form solution to an MDP relies on the recursive relationship known as the *Bellman Optimality Equation*, which asserts that the value of a state under an optimal policy must equal the expected return of the best action from that state:

$$Q^*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid S_t = s, A_t = a \right]. \tag{2.1}$$

This recursive structure is significant because, in scenarios where the environment's dynamics (transition probabilities and reward function) are known, and the state space is finite, these equations form a system that can be solved exactly using **Dynamic Programming** algorithms, such as Value Iteration or Policy Iteration [99]. Ideally, these

methods converge to the global optimum; however, their applicability is limited in practice by the "curse of dimensionality" when the state space $\mathcal{S}$ or action space $\mathbb{A}$ becomes computationally intractable.

2.3.2. Reinforcement Learning – Taxonomy

Reinforcement Learning relaxes the assumptions of Dynamic Programming. In the RL framework, the agent does not have prior knowledge of the transition dynamics $p(s'|s,a)$ or the reward function $\mathcal{R}$. Instead, it must discover optimal strategies through direct interaction with the environment, learning from the consequences of its actions via trial-and-error [99].

RL is typically categorized according to three primary dichotomies: the reliance on an environmental model, the optimization objective, and the data usage strategy.

Model-Based vs. Model-Free

The most fundamental distinction concerns the agent's internal representation of the environment [99, 125]:

- **Model-Based RL:** The agent learns an approximation of the environment's dynamics and uses this learned model to plan actions (e.g., via look-ahead search). While sample-efficient, these methods suffer from "model bias", where errors in the learned model compound to produce suboptimal policies [70, 73].
- **Model-Free RL:** The agent directly learns the value functions or the policy from experience. This approach, widely adopted in complex combinatorial problems, is generally less sample-efficient but asymptotically unbiased and easier to implement in high-dimensional spaces.

Value-Based, Policy-Based, and Actor-Critic

Algorithms are further classified based on the function they aim to optimize [99, 125]:

- **Value-Based:** The agent approximates the action-value function $Q(s,a)$ (e.g., Q-Learning). The policy is implicit, typically derived by selecting the action with the highest estimated value ($\arg\max_a Q(s,a)$).
- **Policy-Based:** The agent explicitly parametrizes the policy $\pi_\theta(a \mid s)$ and optimizes the parameters θ usually via gradient ascent on the expected return (e.g., REINFORCE). This is particularly effective for high-dimensional or continuous action spaces [109].

- **Actor-Critic:** This hybrid approach combines the advantages of both. An "Actor" learns the parameterized policy, while a "Critic" learns a value function to evaluate the actions taken by the actor. The critic reduces the variance of the policy gradient updates, stabilizing the learning process [45].

On-Policy vs. Off-Policy and Learning Modality

The final distinction is based on the use of the generated data [99]:

- **On-Policy:** The agent learns the value of the policy it is currently executing. This requires that the data used for updates must be generated by the current policy π, limiting sample efficiency, as past experiences must be discarded.
- **Off-Policy:** The agent learns a target policy π (usually the optimal one) while acting according to a different behavior policy μ (often more exploratory). This allows for the use of *Experience Replay*, where past transitions are stored and reused, improving sample efficiency.

It is also necessary to distinguish the learning modality in terms of interaction. In **Online RL**,[1] the agent updates its parameters concurrently with its interaction with the environment. In contrast, in **Offline (or Batch) RL**, the agent learns from a dataset of previously collected transitions without further interaction with the environment.

2.3.3. Mathematical Formulation of Policy-based methods

The previous section offers a broad overview of Reinforcement Learning; this research specifically employs a **Model-Free, Policy-Based** framework. This choice is motivated by the high-dimensional state space in the 3D-BPP.
The mathematical derivation presented herein focuses on the trajectory from standard Policy Gradient methods to the specific algorithm utilized in this work: *Proximal Policy Optimization* (PPO) [93].

Policy Approximation via Deep Neural Networks

In policy-based Reinforcement Learning, the goal is to directly optimize the policy $\pi(a \mid s)$, which defines the agent's behavior. Instead of storing a lookup table for every state-action pair, a common strategy is to approximate the policy using a function approximator,

[1] It is crucial to distinguish this terminology from the "On-line" and "Off-line" classifications of the Bin Packing Problem discussed in [Par. 2.1.1]. In the context of RL algorithms, "Online/Offline" refers to the source of training data (dynamic interaction vs. static dataset). In the context of 3D-BPP, these terms refer to the *temporal availability of the item sequence* (real-time stream vs. full prior knowledge).

specifically, a Deep Neural Network parametrized by a d-dimensional weight vector $\theta \in \mathbb{R}^d$ [99].

Formally, the parameterized stochastic policy is denoted as $\pi_\theta(a \mid s)$:

$$\pi_\theta(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s, \theta). \tag{2.2}$$

This representation allows the agent to map the complex features of the State $\mathcal{S}$ to a probability distribution over a set of valid actions $\mathcal{A}$. The objective is to find the optimal parameters θ^* that maximize the expected scalar performance measure $J(\theta)$, defined as the expected cumulative discounted reward:

$$J(\theta) \doteq \mathbb{E}_{\substack{s_t \sim d_{\pi_\theta} \\ a_t \sim \pi_\theta(\cdot|s_t)}} \left[R(s_t, a_t) \right]. \tag{2.3}$$

Where $d_\pi(s)$ is the γ-*discounted occupancy distribution* [100]. Formally, assuming the starting state S_0 is drawn from an initial distribution μ_0, $d_\pi(s)$ is defined as:

$$d_\pi(s) = (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(S_t = s \mid S_0 \sim \mu_0, \pi),$$

generated by following the policy π_θ. The use of a stochastic policy is particularly advantageous in combinatorial optimization as it naturally allows for exploration of the solution space, preventing the agent from getting stuck in local optima during the early stages of training [93].
The distribution $d_\pi(s)$ ensures that the policy updates are proportional to how frequently (and how early, due to the discount factor γ) a state is visited under the current policy [99]. This means the agent prioritizes optimizing actions in states that are part of its likely trajectories, rather than in states that are theoretically reachable but practically irrelevant under the current behavior.

Policy Gradient Theorem

To maximize the objective function $J(\theta)$, the agent employs Gradient Ascent, updating the parameters θ in the direction of the gradient $\nabla_\theta J(\theta)$. A significant challenge in this derivation arises from the fact that $J(\theta)$ depends on both the policy π_θ and the environment's dynamics $p(s' \mid s, a)$.

The **Policy Gradient Theorem** establishes that $\nabla_\theta J(\theta)$ does not depend on the derivative of the environment dynamics $p(s'|s, a)$ [99]. Mathematically, the gradient is proportional

to the expected sum of the gradients of the policy weighted by the action-value function:

$$\nabla_\theta J(\theta) \propto \sum_{s \in \mathcal{S}} d_\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi(a \mid s, \theta). \tag{2.4}$$

By applying the log-derivative trick, this can be expressed as an expectation over the trajectory distribution:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\substack{s_t \sim d_{\pi_\theta} \\ a_t \sim \pi_\theta(\cdot|s_t)}} \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) A_{\pi_\theta}(s_t, a_t) \right], \tag{2.5}$$

where $A_{\pi_\theta}(s_t, a_t)$ represents the *Advantage Function*, defined as:

$$A_{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t).$$

This function quantifies how much better a specific action a_t is compared to the average performance in state s_t. Utilizing the advantage instead of the raw return significantly reduces the variance of the gradient estimate, stabilizing the learning process [93].

Actor-Critic Extension Actor-Critic algorithms implement an approximate policy gradient based on this theorem. They maintain two distinct sets of parameters:

- The **Actor** (parameterized by θ) updates the policy $\pi_\theta(a \mid s)$ in the direction suggested by the Advantage.
- The **Critic** (parameterized by w) estimates the value function (e.g., $V_w(s) \approx V^\pi(s)$) by minimizing the temporal difference (TD) error: $\delta_t = R_{t+1} + \gamma V^\pi(s_{t+1} - V^\pi(s_t)$. The TD error represents the discrepancy between the reward an agent predicts it will receive (via value estimation) and the actual reward it receives after taking an action in the environment. [40]

The Critic's estimate is then used to compute the Advantage $A(s,a)$, providing a lower-variance baseline for the Actor's updates.

Proximal Policy Optimization (PPO)

The Actor-Critic algorithm used in this thesis is PPO.
PPO has become the de facto standard in Deep Reinforcement Learning due to its balance between ease of implementation, sample efficiency, and reliable performance.

The core innovation of PPO is the **Clipped Surrogate Objective**. Instead of allowing

the policy to change arbitrarily during an update step, PPO imposes a "soft" constraint that prevents the new policy π_θ from deviating excessively from the old policy $\pi_{\theta_{old}}$ that generated the data.

The Probability Ratio. First, we define the probability ratio $r_t(\theta)$ between the new and the old policy:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}. \tag{2.6}$$

When $r_t(\theta) = 1$, the policies are identical. If $r_t(\theta) > 1$, the action a_t has become more likely than before; if $r_t(\theta) < 1$, it has become less likely.

The Clipped Objective Function. The objective of PPO is to maximize the expected return while ensuring this ratio stays within a small interval around 1 (typically $[1-\epsilon, 1+\epsilon]$). The objective function is defined as:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right], \tag{2.7}$$

where:

- ϵ is a hyperparameter determining the tolerance for policy change.
- A_t is the Advantage function computed using the old policy.
- The min operator takes the pessimistic lower bound of the unclipped and clipped objectives.

This mechanism works as a safety brake:

- If the advantage A_t is positive (the action was good), the algorithm tries to increase the probability of that action (increasing r_t). However, if r_t grows beyond $1+\epsilon$, the term is clipped, and the gradient becomes zero. This prevents the policy from becoming "too confident" too quickly based on a single batch of data.
- Conversely, if A_t is negative (the action was bad), the algorithm decreases the probability. If r_t drops below $1-\epsilon$, the clipping again stops the update to prevent destroying the policy distribution entirely.

This "pessimistic" approach ensures that updates are always safe, allowing the algorithm to reuse the same batch of data for multiple optimization epochs without diverging.

Total Loss Function. The complete loss function to be minimized includes the value function error and an entropy bonus to encourage exploration:

$$L_t^{PPO}(\theta) = \mathbb{E}_t \left[-L_t^{CLIP}(\theta) + c_1 L_t^{VF}(\theta) - c_2 S[\pi_\theta](s_t) \right], \tag{2.8}$$

where c_1, c_2 are coefficients, $L_t^{VF} = (V_\theta(s_t) - V_t^{target})^2$ is the squared error of the value function, and S denotes the entropy of the policy distribution [93].

2.3.4. Temporal Abstraction and Hierarchical RL

Standard MDPs operate on a strict discrete-time basis where every action takes exactly one time step. However, complex decision-making problems, such as the 3D-BPP, may require reasoning at multiple time scales. For example, placing a single item is an atomic action, whereas filling a complete layer of items constitutes a "macro-action" that persists over multiple time steps.

To model such behaviors, the formalism is extended to Semi-Markov Decision Processes (SMDPs). In an SMDP, a "macro-action" initiated at time t takes a variable duration $\bar{t}$ to complete. The system models the joint probability $P(s', \bar{t} \mid s, a)$ of landing in state s' after $\bar{t}$ steps.
The Bellman equation is modified to account for the cumulative discounted reward accumulated over the duration $\bar{t}$:

$$Q(s,a) = \mathbb{E}\left[\sum_{k=0}^{\bar{t}-1} \gamma^k R_{t+k+1} + \gamma^{\bar{t}} \max_{a' \in \mathcal{A}} Q(s', a') \right]. \tag{2.9}$$

This formulation allows the system to treat temporally extended sequences of actions as single transitions at a higher level of abstraction, effectively reducing the horizon of the decision problem [87, 101].

However, in the standard SMDP formulation, these macro-actions are treated as "black boxes": once initiated, the control is committed to the macro-action until it concludes naturally after $\bar{t}$ steps, without the ability to inspect or alter the behavior during execution [87].

The Options Framework

To overcome the rigidity of standard SMDP actions, Sutton, Precup, and Singh introduced the Options Framework [101]. An **Option** ω formalizes the concept of a macro-action but retains a "white box" structure. It is defined by the tuple $\omega = \langle \mathcal{I}_\omega, \pi_\omega, \beta_\omega \rangle$:

- $\mathcal{I}_\omega \subseteq \mathcal{S}$: The *Initiation Set*, defining where the option can start.
- $\pi_\omega : \mathcal{S} \times \mathcal{A} \to [0, 1]$: The *Intra-option Policy*, which selects primitive actions at every time step while the option is active.
- $\beta_\omega : \mathcal{S} \to [0, 1]$: The *Termination Condition*, which determines the probability of terminating the option at the current state.

The distinction from standard SMDPs lies in the components π_ω and β_ω. Unlike an SMDP action that runs blindly for $\bar{t}$ steps, an Option is a closed-loop control policy. The agent can react to state changes *during* the execution of the option (via π_ω) and can decide to terminate early (via β_ω) if the goal is achieved or if the context changes, offering significantly greater flexibility than fixed macro-actions [8, 101].

Hierarchical Reinforcement Learning (HRL)

Hierarchical Reinforcement Learning (HRL) [51] utilizes this decomposition to solve tasks with long horizons. A typical two-level HRL architecture, as adopted in this research, consists of:

- **Manager (High-Level Controller):** Operates on the induced SMDP, learning a policy $\pi_M(s)$ to select which Option ω to execute to maximize the global long-term return.
- **Worker (Low-Level Controller):** Operates on the underlying MDP, executing the specific primitive actions required by the active option's policy π_ω.

Parametric Hierarchical Reinforcement Learning (HRL-P)

Standard HRL architectures typically assume a finite, discrete set of options $\Omega = \{\omega_1, \ldots, \omega_n\}$. This limitation becomes problematic in environments requiring fine-grained control over continuous variables. To address this, Klimek et al. proposed *Hierarchical Reinforcement Learning with Parameters* (HRL-P) [59].

In the HRL-P framework, the action space of the Manager is extended to include not only the discrete choice of the option index k but also a vector of continuous parameters x_k. The Manager's policy π_M outputs a tuple $\langle \omega_k, x_k \rangle$. The parameter x_k is then passed to the lower-level controller (Worker), effectively conditioning the intra-option policy $\pi_{\omega_k}(a \mid s, x_k)$ on this specific goal.

This enables the agent to instantiate a specific behavior from an infinite family of options while maintaining a compact discrete action space for the high-level planner.

2.4. Deep Learning

Deep Learning serves as a foundational pillar for modern Reinforcement Learning. As discussed in Section 2.3.3, Deep Neural Networks (DNNs) are employed as powerful function approximators to estimate policies π_θ and value functions V_ϕ in high-dimensional state spaces. This section provides a brief overview of the neural architectures utilized to process sequential data, specifically focusing on the Attention Mechanism and Pointer Networks, which are essential tools for the development of this work.

2.4.1. Deep Neural Networks (DNN)

At its core, a Deep Neural Network is a composite function parameterized by weights and biases. The fundamental building block is the Multi-Layer Perceptron (MLP), where the output $h^{(l)}$ of layer l is computed as:

$$h^{(l)} = \sigma(W^{(l)}h^{(l-1)} + b^{(l)}). \tag{2.10}$$

Here, W and b represent the learnable parameters, and $\sigma(\cdot)$ is a non-linear activation function (e.g., ReLU or Tanh). By stacking multiple layers, DNNs can learn hierarchical representations of complex input data [44].

2.4.2. Sequence-to-Sequence (Seq2Seq) Modeling

Traditional feed-forward neural networks, such as Multi-Layer Perceptrons (MLPs), require fixed-size input and output vectors, which makes them unsuitable for handling sequential data or sequences of variable length. To overcome this limitation, the Sequence-to-Sequence (Seq2Seq) architecture was introduced [98]. Seq2Seq follows an encoder –decoder paradigm: the encoder compresses an input sequence into a compact latent representation, and the decoder generates an output sequence from that representation. This architecture is widely used in deep learning, with mostly known applications in image encoding and reconstruction.

This framework decomposes the learning process into two distinct phases:

1. **Encoding:** An Encoder network processes the variable-length input sequence $X = (x_1, \ldots, x_T)$ and compresses the information into a fixed-length context vector c.

2. **Decoding:** A Decoder network takes this context vector c and generates the output sequence $Y = (y_1, \ldots, y_M)$ step-by-step, conditioned on the context and the

previously generated outputs.

While traditionally implemented using Long Short-Term Memory (LSTM) networks to mitigate the vanishing gradient problem, this architecture suffers from an information bottleneck: forcing the entire history of a long sequence into a single fixed-length vector c inevitably leads to the loss of information and the inability to capture long-range dependencies effectively [22].

2.4.3. The Attention Mechanism

To overcome the bottleneck of fixed-context vectors, Bahdanau et al. [9] introduced the *Attention Mechanism*. The core idea is to relieve the encoder from the burden of compressing all information into a single state. Instead, attention allows the decoder to "look back" at the entire source sequence and dynamically focus on different parts of the input at each decoding step.

Mathematically, at each time step t, the model computes a distinct context vector c_t as a weighted sum of all encoder hidden states $(h_1, \ldots, h_T)$:

$$c_t = \sum_{i=1}^{T} \alpha_{ti} h_i. \tag{2.11}$$

The attention weights α_{ti} represent the relevance or "alignment" between the current decoding state and the i-th input element. By explicitly modeling these dependencies, the attention mechanism enables the network to retrieve specific details from the input regardless of the sequence length.

2.4.4. The Transformer Architecture

Vaswani et al. demonstrated that recurrence is not strictly necessary for sequence modeling. In their seminal work "Attention Is All You Need" [104], they proposed the **Transformer**, an architecture based entirely on attention mechanisms both on encoding and decoding (see [Fig. 2.2]).
By discarding recurrence, Transformers allow for massive parallelization during training and, more importantly, enable the modeling of dependencies between input elements regardless of their distance in the sequence.

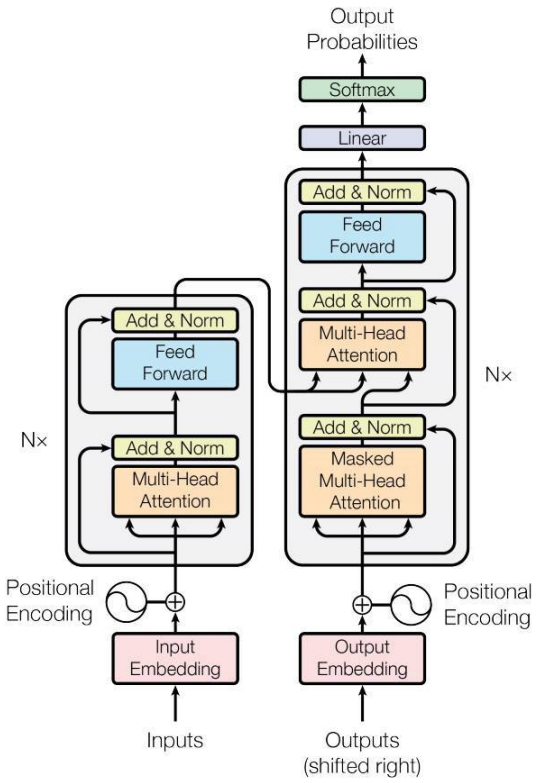


Figure 2.2: Transformer Architecture - image from [104].

Self-Attention

The core component of the Transformer is **Self-Attention**. Unlike the standard attention described in Section 2.4.3, where the queries come from the decoder and keys/values from the encoder, in Self-Attention, the queries, keys, and values are all derived from the same input sequence.

Given an input matrix X, the layer learns three weight matrices W^Q, W^K, W^V to project the input into Queries (Q), Keys (K), and Values (V). The output is computed using the Scaled Dot-Product Attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.12}$$

The dot product QK^T computes a similarity score between every pair of items in the sequence. The scaling factor $\frac{1}{\sqrt{d_k}}$ is applied to counteract the effect of large dot products pushing the softmax function into regions with extremely small gradients [104].

Multi-Head Attention

To allow the model to focus on different types of relationships simultaneously, the Transformer employs *Multi-Head Attention*. Instead of performing a single attention function, the model projects the queries, keys, and values h times with different, learned linear projections. These h "heads" run in parallel, and their outputs are concatenated and projected again:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O, \tag{2.13}$$

where each $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

Positional Encoding

Since the Transformer contains no recurrence and no convolution, it is invariant to the order of the sequence. To inject information about the relative or absolute position of the items in the list, *Positional Encodings* are added to the input embeddings at the bottom of the encoder and decoder stacks. These encodings use sine and cosine functions of different frequencies:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right). \end{aligned} \tag{2.14}$$

This allows the model to distinguish between identical items appearing at different positions in the input stream.

2.4.5. Pointer Networks

Standard neural network architectures, including the original Transformer, generally predict outputs from a fixed-size vocabulary (e.g., the dictionary of a language). However, in combinatorial optimization problems, the output is a permutation or a subset of the input sequence itself. Since the number of items varies across instances, a fixed output layer is inapplicable.

To solve this, Vinyals et al. introduced *Pointer Networks* (Ptr-Nets) [107]. Instead of using attention to blend hidden states into a context vector (as in standard translation tasks), Ptr-Nets use the attention weights directly as the output probability distribution.

Mathematically, let $(e_1, \dots, e_n)$ be the encoder states representing the n available items. At decoding step t, the network compares the current decoder state d_t with each encoder

state to compute a score u_i^t:

$$u_i^t = v^T \tanh(W_{ref} e_i + W_q d_t) \quad \text{for } i \in \{1, \dots, n\}, \tag{2.15}$$

where v, W_{ref}, W_q are learnable parameters. The probability of selecting the i-th item in the input sequence is then obtained via a softmax over the input length:

$$p(C_t = i \mid C_1, \dots, C_{t-1}, X) = \text{softmax}(u^t)_i. \tag{2.16}$$

In modern Reinforcement Learning, this mechanism is often combined with Transformer encoders (an architecture referred to as *Pointerformer*), where the Transformer captures the complex inter-item relationships, and the Pointer head selects the optimal item to pack based on the learned representation [54].

3 | Related Works

This chapter reviews the literature concerning solution strategies for the 3D-BPP. The discussion begins with traditional non-learning methods, followed by a comprehensive overview of recent work using DRL.
As explained in [Par. 2.1.1], offline and online packing represent inherently different problems characterized by distinct constraints and information availability. Consequently, this review separates the analysis of solutions for the offline and online paradigms.

3.1. Non-Learning Solutions for Offline 3D-BPP

The 3D-BPP has been a subject of intensive research for over five decades, beginning with early algorithmic explorations [23, 55]. Given the problem's NP-hard nature, researchers have developed various strategies to balance computational efficiency with solution quality.

Zhao et al. [126] categorize early algorithmic approaches into three primary groups:

- **Placement Heuristics:** Fast, rule-based methods that provide immediate feasible solutions through specific loading instructions.
- **Meta-heuristics and Hybrid Genetic Algorithms:** Optimization frameworks designed to navigate the search space for local optima, often improving upon basic heuristic results.
- **Exact Solvers:** Methods such as linear programming [48] that guarantee optimality but typically require significant problem simplification to remain computationally tractable.

A more recent holistic review [1] analyzes 3D-BPP variants by their constraints and their operational paradigms. Following the structure proposed in [1], this section classifies 3D-BPP solutions into: exact algorithms, constructive heuristics, meta-heuristics, tree-search, and hyper-heuristics.

3.1.1. Exact Methods for Offline 3D-BPP

Exact methods are designed to find the mathematically optimal packing configuration. Albeit powerful for small-scale instances, these methods often fail to converge within reasonable time frames as the number of boxes increases.

The most prominent exact approaches include:

- **Mathematical Programming.** These formulations, such as Mixed-Integer Linear Programming (MILP), model the packing task as a set of linear constraints and objectives [33, 48, 83, 102].
- **Combinatorial Algorithms.** These include specialized branch-and-bound techniques that prune the search space to find optimal arrangements more efficiently than exhaustive search [72].

3.1.2. Constructive Heuristics for Offline 3D-BPP

Constructive (placement) heuristics are generally the basis of other approximation algorithms that attempt to create a feasible solution based on a set of rules. Constructive algorithms can involve three components: sequencing, managing empty spaces, and loading patterns.

Sequencing

[126] categorizes sequences as static or dynamic. In static sequencing, the order of items is determined before the packing procedure and will not be changed. Conversely, in dynamic sequencing, considering the problem state, the order of items can be changed during the packing process.

Static sequencing. In terms of static sequencing, [21] presented five ranking criteria to order items.

1. Select the item with the largest base dimensions (i.e., length and width).
2. Select the item with the largest base area.
3. Select the item with the largest x dimension.
4. Select the item with the largest y dimension.
5. Select the item with the largest z dimension.

Dynamic sequencing. Regarding dynamic sequencing, there are two mostly used criteria:

1. **Volume utilization**: items are first categorized based on their types, and then in each step of the packing procedure, an item type that yields the best volume utilization is selected [76].
2. **Score of feasible potential placements[81]**. Each next placement is chosen greedily, i.e., a maximal score is searched. Scoring incorporates 3 relevant properties of feasible placements: unutilized volume, characteristics of remaining boxes, and the extent to which the boxes have support. For a feasible potential placement, each property is quantified by a scoring component such that a better placement with respect to the property yields a higher value of the component.

3.1.3. Managing the Empty Spaces

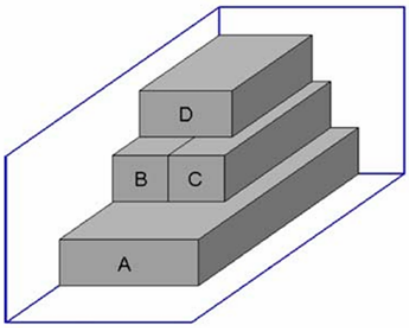
This class of heuristics reduces the action space so that cargos can only be placed in specified areas of the bin. Then a *Loading Pattern* is exploited to pack each box in the bin.

Empty space representation

There exist multiple possibilities for representing the bin's empty space.

Cellular decomposition. One of the first attempts, by Ngoi and Whybrew [79], proposes a three-dimensional matrix data structure to represent variable orthorhombic cells. This 3D matrix is a vector of 2D matrices, each one representing a change in the cross-section of the object. Bischoff proposed a simplified representation with a 2-dimensional matrix [12]. The first row and the first column of the matrix show the values obtained by projecting the positions of all vertical box/bin edges onto the two horizontal axes. The other cells represent the height of the load above the corresponding floor section as shown in [Fig. 3.1].

Corner Points. A different approach is introduced by Martello et Al. [71] called "Corner Points (CPs)". Corner Points are defined *as non-dominated locations where an item can be placed into an existing packing.* In 2D-space, corner points are defined where the envelope of the boxes in the bin changes from vertical to horizontal, i.e., where vertical and horizontal lines intersect. In 3D-space, CPs result from the intersection between the three planes formed by the back, right, and top sides of the packed boxes [1] (see [Fig. 3.2a]). Once CPs are defined, one can place a new box on any of the corner points and nowhere else.



Matrix 1:

0	60	100	120	150	200
150	150	150	100	50	0
200	100	100	100	50	0
300	50	50	50	50	0
400	0	0	0	0	0

Container dimensions (cm): 400 x 200 x 180 (ht)

Box	Dimensions (cm)	Weight (kg)	Load bearing limit (g/cm^2)
A :	300 x 150 x 50 (ht)	2.250	140
B :	200 x 60 x 50 (ht)	600	120
C :	200 x 60 x 50 (ht)	600	80
D :	150 x 100 x 50 (ht)	750	30

Matrix 2:

0	60	100	120	150	200
150	30	30	80	140	∞
200	90	80	80	140	∞
300	140	140	140	140	∞
400	∞	∞	∞	∞	∞

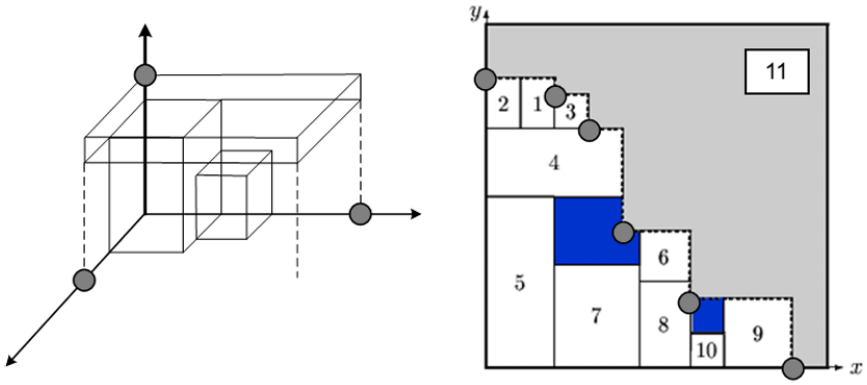
Figure 3.1: Keeping track of the height-level of available loading surfaces (Matrix 1) and their load-bearing abilities (Matrix 2). Image by [12].

Extreme Points. Since CPs may ignore free spaces in inaccessible regions [1], Crainic et Al. [24] defined "Extreme Points (EPs)". Instead of intersecting the three planes, when a new box k is placed, the points on coordinates $(x_k + w_k, 0, 0)$ $(0,_k + l_k, 0)$ $(0, 0, z_k + h_k)$ are projected on the orthogonal axis of the container. The EP lies on the nearest item on which such coordinates are projected. The point with coordinate (x_k, y_k, h_k) is the rear bottom-left point of the item and (w_k, l_k, h_k) are its width, length, and height.
The origin of the orthogonal axis of the bin is on its rear bottom left (see [Fig. 3.2b]).

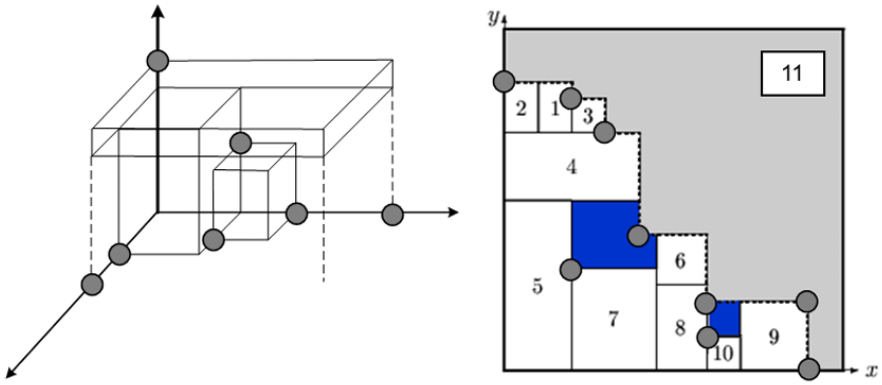
Empty Spaces. Another approach, proposed by George and Robinson [39], is to represent the unused space in the bin: the placement of a new box results in multiple partitions of the bin's unused space in three new spaces. While George and Robinson only considered one possible partition, Bortfeldt et Al. [14] extended it to consider all possible partition variants (see comparison in Fig. 3.3). Moreover, Parreño et Al. [84] introduced the concept of Empty Maximal Spaces (EMSs) as the largest available empty spaces, as shown in

Loading Patterns

Given the empty space representation, Pisinger [85] first proposed four different packing strategies, and a few years later Fanslau and Bortfeldt [32] extended them to five major strategies: wall building, stack building, horizontal layer building, block building, and guillotine cutting This section gives an overview of these strategies; for a more comprehensive

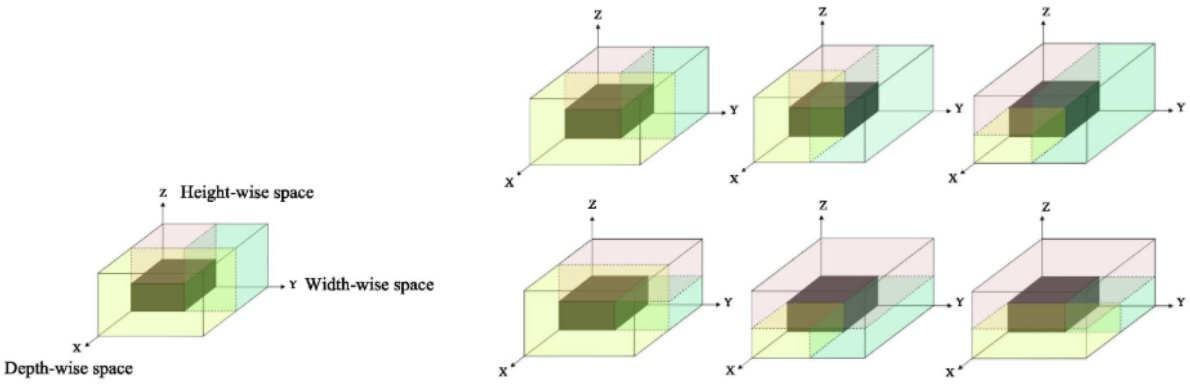


(a) Example of definition of CPs in 3D and 2D packing.



(b) Example of definition of EPs in 3D and 2D packing.

Figure 3.2: Comparison between CPs and EPs. Image by [24].



(a) [39] version

(b) [14] version

Figure 3.3: Comparison between empty spaces subdivision algorithms proposed by [39] and [14].

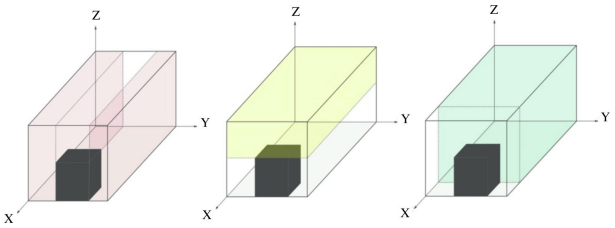


Figure 3.4: EMS proposed by [84]. Image by [1].

discussion, see [1].

Wall Building. The wall building pattern constructs the loading layout as a sequence of vertical layers ("walls") placed along the container depth. Each wall is defined by the first box inserted, and the method focuses on filling one wall at a time to reduce fragmentation and simplify spatial decisions. Originally introduced by George and Robinson [39], it became a foundational constructive heuristic for weakly heterogeneous instances. Subsequent adaptations (among others, Moura and Oliveira [76], Kurpel et al. [60]) used the same principle mainly to generate columns or initialize more complex optimization procedures taking into account also geometric and safety constraints.

Layer Building. The layer building pattern packs items by constructing horizontal layers, each fully supported by the one below. The method begins with a base layer on the container floor and continues stacking layers upward until no additional height is available. Its purpose is to simplify the 3D loading process by reducing it to a sequence of 2D layouts, naturally promoting stability.
This pattern has been adopted in several variants [47, 89, 92], either as a standalone constructive heuristic or as the initialization module of hybrid approaches. Applications span SKP, SLOPP, and BPP settings, often focusing on selecting or generating feasible horizontal layers of identical items or stable mixed configurations.

Stack Building. The stack building pattern constructs the loading by first forming "vertical stacks", i.e., columns of items built by placing boxes on top of a chosen base. Then, these stacks are arranged on the container floor through a 2D packing procedure. Early work, such as [38], generated stacks via a greedy volume-maximizing procedure and then positioned them using a genetic algorithm.

Block Building. The block building pattern groups items into blocks that can be either collections of identical boxes or mixed blocks combining different item types. These pre-assembled structures are then loaded in the bin, rather than loading every single box.

This reduces combinatorial complexity and stabilizes the packing process.
Araújo and Armentano [4] first used the method to pack weakly heterogeneous boxes, and later extended by Fanslau and Bortfeldt [31] to generate "general blocks" comprising different boxes with small internal gaps. Elhedhl et al. [29] proposed an approach based on Block building to form "superitems" which are then considered as single boxes in the formation of layers for Layer building.

Guillotine Cutting. The guillotine cutting pattern recursively partitions the container into smaller rectangular regions through axis-parallel cuts, ensuring that no cut intersects any placed item. Often applied to satisfy the guillotine cutting constraints, as in [32, 66, 94].

3.1.4. Meta-Heuristics for Offline 3D-BPP

Solutions generated by constructive heuristic methods can be improved using local search and meta-heuristic algorithms. These methods generally try to explore a search space and iteratively make changes to improve the quality of the current solution. A comprehensive review of the many studies that are covered by this umbrella of solutions is provided by [1]. This section will highlight only the solutions that are more relevant for this thesis.

Tabu Search. Tabu Search (TS) is a meta-heuristic that enhances local search by using short-term memory to forbid recently applied moves, preventing cycling and enabling the exploration of new regions of the search space. Its objective is to iteratively improve an initial solution while systematically avoiding local optima.
In 3D packing, TS is commonly embedded within constructive heuristics. Notable applications include [19, 65, 68, 69].

Genetic Algorithms. Genetic Algorithms (GAs) are evolutionary meta-heuristics that iteratively refine a population of candidate solutions through mutation. Guided by a fitness function, they promote "genes" with desirable characteristics and explore new regions of the search space.
In the 3D-PP literature, GAs are typically hybridized with constructive heuristics and can be combined with adaptive scoring schemes, ILP subproblem evaluation, or physics-based stability assessment to improve their packing quality. Notable applications include [78, 80, 82, 111].

Biased Random Key Genetic Algorithms. Biased Random-Key Genetic Algorithms (BRKGAs) are a class of evolutionary algorithms where each solution is encoded as a vector of random keys, and a decoder function translates these vectors into a feasible

solution for the optimization problem. The biased selection of elite individuals guides the search toward high-quality solutions while preserving diversity through random-key representations.
In 3D packing, BRKGAs are typically combined with constructive layout heuristics to evolve item sequences, layer types, or orientation choices. Notable applications include [41, 42, 90, 127].

Simulated Annealing. Simulated Annealing (SA) is a meta-heuristic inspired by the physical annealing of solids: it begins at a high "temperature" that allows the algorithm to accept worse solutions with a certain probability, enabling escape from local optima. As the temperature decreases, this probability diminishes, gradually steering the search toward more refined solutions.
In 3D packing, SA is typically paired with constructive heuristics to improve feasibility and solution quality under various constraints. Representative applications [25, 74, 95].

Greedy Randomized Adaptive Search Procedure. The Greedy Randomized Adaptive Search Procedure (GRASP) is an iterative meta-heuristic that constructs a randomized greedy solution and then improves it through local search. Its goal is to balance diversification (via randomized constructive steps) and intensification (via neighborhood exploration) to obtain high-quality solutions.
In 3D packing, GRASP is commonly paired with constructive heuristics and then refined through local improvement strategies. Representative applications include [3, 64, 75], addressing SLOPP and multi-CLP settings under various stability, weight, and feasibility constraints.

3.1.5. Tree Search Methods for Offline 3D-BPP

Tree search methods explore the packing space by incrementally extending partial solutions and branching over alternative placements until no further items can be inserted. In 3D packing, these methods typically rely on constructive heuristics to structure the branching process and control the combinatorial explosion. Their goal is to load one or multiple bins by exploring feasible successor states while pruning unpromising branches.

General Tree Search Approaches. Most tree-based algorithms integrate domain heuristics to guide node expansion. Wall-building-based methods, as [67, 86, 96], branch over alternative wall depths or strip widths and evaluate resulting residual spaces. Block -based approaches, as [67, 86, 96], treat blocks as atomic units and expand the search by

selecting space–block combinations. These strategies differ mainly in their encoding of candidate moves and in the criteria used to evaluate residual free spaces or block feasibility.

Beam Search Variants. Beam search introduces a restricted form of tree exploration: instead of expanding all children at a given depth, only the most promising nodes (according to a greedy evaluation) are retained. This makes beam search computationally lighter and suitable for highly heterogeneous or constraint-rich settings. Representative work includes [5–7], where scoring functions incorporate geometric and stability considerations, and multi-objective variants balance volume utilization with additional criteria such as profit.

3.1.6. Hyper-Heuristics for Offline 3D-BPP

Hyper-heuristics (HHs) are high-level search strategies designed to automate the selection or generation of low-level heuristics. Unlike meta-heuristics that operate directly on solution spaces, HHs search in the space of heuristics, aiming to generalize across instances and reduce human intervention [18].

In the context of packing problems, HHs are typically classified as:

- **selective**, which chooses among existing heuristics;
- **generative**, which constructs new heuristics from components of pre-existing ones.

They can also be categorized as online, offline, or non-learning depending on how feedback is obtained during the search [16]. Machine learning techniques can enhance these mechanisms by learning heuristic performance patterns or by evolving composite rules [15]. Although most applications have focused on 1D and 2D packing, primarily through selective HHs, several studies have applied generative HHs to 3D-PPs. These works predominantly rely on genetic programming (GP) to evolve constructive heuristics tailored to strongly heterogeneous item sets. Representative examples include [2, 17, 26].

3.2. Non-Learning Solutions for Online 3D-BPP

This section reviews the approaches developed for the online scenario, where items arrive sequentially and must be placed immediately in the order of arrival, without the possibility of repositioning previously packed items.
A key aspect is that online methods do not employ local search and meta-heuristics. The reason is intrinsic to the online setting: decisions must be immediate upon the arrival of each item, and unloading or rearranging the current packing configuration is not allowed.

This removes the possibility of iterative refinement [1].

The literature on online 3D-BPPs can be grouped into three main methodological families:

- Constructive heuristics
- Artificial intelligence techniques.
- Theoretical analyses, to find theoretical bounds, and to measure the performance of methods.

The remainder of this section follows the classification proposed in [1] and discusses these three categories in detail.

3.2.1. Constructive Heuristics for Online 3D-BPP

In online 3D packing, items must be packed immediately in the order they arrive, and no reordering or repositioning is allowed; this makes the sequencing heuristics irrelevant in the online setting. As a result, several loading patterns commonly used offline, such as wall building, layer building, stack building, and block building, are not directly applicable because their effectiveness depends on grouping items by similar size or shape before loading.

However, the management of empty spaces remains fully valid, and all the approaches reviewed in offline settings are also applicable online. The main challenge, therefore, lies in designing a loading strategy that can place each incoming item efficiently into one of the available free spaces while adhering to the online requirements. Therefore, due to the lack of online constructive heuristics in the literature, the remaining part of this section presents some offline rules that can be applied to develop online heuristics in future studies [1].

Deepest-Bottom-Left-Fill. Deepest-Bottom-Left-Fill (DBLF) [58] is a trivial loading strategy consisting of placing the next box in the rear-left-most area of the bin (as the name itself suggests). This heuristic has been widely used in offline studies [34, 57].

Best Match First. Best Match First (BMF) is a constructive heuristic introduced by Li and Zhang [63] that selects the placement producing the most suitable "match" between an item and a free space. For each incoming box, the heuristic evaluates all feasible combinations of bin, orientation, and empty space, and chooses the combination that minimizes the residual gaps around it.

Online Constructive Heuristic. Ha et al. [46] proposed a method that combines EMS-based space management with the DBLF priority rule and the BMF heuristic to handle the multiple heterogeneous bins. Their algorithm selects free spaces in lexicographic order (x, then z, then y), ensures that the item can be inserted from the container door, and applies a best-match criterion to choose the most suitable bin, orientation, and space for each incoming item. When multiple feasible placements exist, the option with the smallest margin is selected.
This is effectively the only proposed heuristic for online 3D-BPP.

Corner Distances. The corner distances rule evaluates each EMS by measuring how close it is to the nearest container corner. For every EMS, the Manhattan distance between each of its corners and the closest corner of the container is computed; the smallest of these values defines the EMS's anchor distance. Among all candidate EMSs, the heuristic selects the one with the minimum anchor distance, favoring placements that lie closer to the container structure and potentially lead to more compact packings [129].

Distance to the Front-Top-Right Corner of the Container (DFTRC). The DFTRC rule evaluates each EMS by computing the Manhattan distance between its minimum coordinates and the container's front-top-right corner. The EMS with the largest DFTRC value is selected, favoring placements that push items deeper into the container and leave more accessible free space for future arrivals. A variant, DFTRC-2, extends this idea by considering all feasible box orientations at each EMS and selecting the placement whose front-top-right corner yields the maximum DFTRC value. Both methods aim to produce compact packings that preserve usable space for upcoming boxes [43].

Minimum Vertex Coordinates. The minimum vertex coordinates rule compares EMSs by examining their deepest-bottom-left vertices in lexicographic order. For each EMS, the three coordinate values are sorted from smallest to largest, and the EMS with the lexicographically smallest triplet is preferred. In practice, this means comparing the smallest coordinate first; if tied, comparing the second smallest; and, if still tied, the third. The EMS whose sorted coordinates are lexicographically smaller is selected, favoring placements that are closer to the container's origin [63].

Back Bottom. The back bottom rule selects the EMS with the smallest x-coordinate, placing priority on positions closest to the container's back. If multiple EMSs share the same x-value, the one with the lowest z-coordinate is chosen. In the event of a further tie, the method differs from DBLF by selecting the EMS closest to either of the two

back-bottom corners (left or right), favoring placements that push items toward the rear base of the container [89].

Verma et al. heuristic Verma et al. [106] introduced an online constructive heuristic inspired by the principles of layer building and wall building. For each incoming item, the algorithm evaluates all feasible placements in the container and assigns a stability score that reflects how well the item fits into the current configuration. The scoring function favors locations that create smooth, stable upper surfaces, similar to layer building. Among all feasible location-orientation pairs, the algorithm selects the one with the highest score, aiming to maintain compactness while preserving stable support for future items.

3.3. Comparison and Summary of Non-Learning Methods

The performance of the solution techniques reviewed in the previous two sections varies significantly depending on item heterogeneity, constraint richness, and problem formulation.

A comprehensive empirical comparison of these methods is beyond the scope of this thesis. For a detailed benchmarking, we refer the reader to [1], which provides an extensive evaluation across multiple problem classes and constraint sets.

3.4. Reinforcement Learning Solutions for Offline 3D-BPP

Recent advancements in combinatorial optimization have increasingly focused on RL methodologies. Within the domain of 3D-BPP, numerous offline solutions have been proposed. Before reviewing specific contributions, we define the recurring notation used to describe problem dimensions: L, W, H refer to the bin dimensions, while l_i, w_i, h_i refer to the dimensions of the i-th unpacked box. Given the high dimensionality of the action space in 3D-BPP, prevalent research favors model-free, policy-based frameworks, typically Actor-Critic architectures, over value-based methods [27, 49, 52, 53, 61, 88, 108, 119, 120].

3.4.1. Sequence Optimization and Heuristic Placement

Early DRL applications often decomposed the problem complexity by focusing the learning agent solely on the packing sequence, delegating the geometric placement to constructive heuristics.

Hu et al. [49] pioneered this approach, utilizing a REINFORCE algorithm to minimize the surface area of a bin with non-fixed dimensions. Their state representation consists solely of the sequence of box dimensions (l_i, w_i, h_i). To manage complexity, they employ RL exclusively for the selection task, approximating the policy via a Pointer Network (Ptr-Net), while orientation and placement are determined by standard heuristics.

Building on this sequence-optimization approach, Duan et al. [27] introduced a Multi-task Selected Learning (MTSL) framework using PPO. While retaining the Ptr-Net for sequence generation, they enhanced the encoder with intra-attention to better model dependencies. A key distinction in their work is the treatment of orientation as a supervised classification task rather than a fixed heuristic, employing a hill-climbing strategy to generate ground-truth labels. Consequently, their total loss combines the RL selection loss and the supervised orientation loss.

Similarly, Wang et al. [108] proposed an on-policy Actor-Critic framework utilizing a modified Ptr-Net with a 1D convolutional encoder. While relying on a height-map-based heuristic for top-down loading, their innovation lies in a distinct reward shaping designed to optimize "Compactness" and "Pyramid" structures, explicitly guiding the agent toward stable configurations. Like [49], they relied on a heuristic for placement.

3.4.2. Joint Action Space and Exploration Strategies

Moving beyond heuristic placement, other works aim to learn the full joint action space, requiring advanced exploration strategies to handle the resulting dimensionality and reward sparsity.

Laterre et al. [61] introduced the Ranked Reward (R2) algorithm to learn the full joint action space of selection, orientation, and placement. Their state encodes the full configuration of boxes and their current positions. To overcome the sparsity of rewards in single-player BPP, the R2 mechanism compares the agent's performance against a buffer of its own recent episodes, mimicking the self-play dynamic of AlphaZero [97]. Their function approximator combines a deep neural network with Monte Carlo Tree Search (MCTS) to navigate the complex decision tree.

To handle larger-scale instances with discrete placement coordinates, Jiang et al. [52] developed a Multimodal DRL agent using the A2C framework. Their state representation combines the list of box dimensions with a heightmap of the bin. The function approximator is a multimodal encoder featuring sparse attention for the box state and a CNN for the heightmap. They addressed the massive discrete action space by implementing Action Representation Learning [20], allowing the agent to generalize across spatially similar

actions while optimizing a step-wise gap reduction reward.
Jiang et al. [53] then combined this pre-trained DRL policy into a Constraint Programming (CP) solver to guide the branching process, combining the speed of learning-based methods with the systematic search capabilities of CP.

3.4.3. Transformer-based Architectures and State Encodings

Recent research argues that traditional state representations or decision orders may introduce limitations, proposing Transformer-based architectures and novel embeddings to capture spatial relationships more effectively.

Zhang et al. [120] proposed "Attend2Pack", an Actor-Critic model where the state tuple includes the box sequence and a "frontier embedding" – a $W \times H$ front view of the bin's filling profile. They decomposed the action space into a sequence policy, approximated by a Ptr-Net, and a placement policy, approximated by a CNN-encoded Transformer decoder. Their approach employs a "prioritized oversampling" technique to refine sequence generation for complex packing scenarios.

Challenging the traditional decision order, Que et al. [88] presented a Transformer-based PPO model that selects the target position *before* selecting the box and orientation. This reordering prioritizes the geometry of the remaining space over the specific order of incoming items. Their bin state is enriched with "plane features", explicitly encoding distances to container boundaries and neighboring surfaces, processed by a full Transformer encoder–decoder setup.

Finally, Yin et al. [119] argue that the traditional three-stage decomposition (sequence, orientation, position) introduces error propagation. They propose a two-stage scheme where the box index and orientation are combined into a single selection stage, followed by position selection. To manage placement in large bins, they introduce a Bidirectional Cooperative Packing (BCP) method, which generates policies for both x-axis and y-axis packing directions simultaneously, selecting the action that maximizes the value function across both views.

3.5. Online Reinforcement Learning Strategies for 3D-BPP

The online 3D-BPP frames the packing task as a sequential decision-making process. While the fundamental constraint requires the immediate placement of the current item without

re-adjustment of packed boxes, many frameworks assume a limited lookahead, granting visibility of the subsequent k boxes on the conveyor.
Recent literature has largely converged on model-free, policy-based DRL frameworks, typically Actor-Critic architectures, while diverging significantly in state representation, action space management, and the integration of physical constraints.

3.5.1. Constrained Exploration via Action Masking

A critical challenge in online packing is restricting the agent to physically feasible actions. Zhao et al. [123] address this by formulating the problem as a Constrained MDP, solved via the ACKTR framework [112].
To encode the spatial state, the authors employ a CNN. The dimensions of the current box are projected onto a 2D plane matching the resolution of the bin's height map, creating a unified multi-channel input. To enforce feasibility, they define a conservative mask based on three criteria: boundary limits, collision avoidance, and static stability support. This mask serves as ground truth to supervise an auxiliary network that predicts valid actions, effectively pruning the search space. Finally, to exploit the visibility of incoming items on the conveyor, the framework integrates Monte Carlo Tree Search (MCTS), allowing the agent to simulate future states and optimize the long-term packing density.

[124] extend this framework by implementing a "stacking tree" structure to compute the feasibility map. This method provides efficient $O(N \log N)$ stability analysis, eliminating the dependence on approximate MLP predictions. To support high-resolution packing (100×100 grids), the action space is decoupled into three discrete heads: orientation, x-coordinate, and y-coordinate. The resulting architecture utilizes a CNN state encoder feeding sequential actor heads and a unified critic within the ACKTR algorithm. Finally, a "far-to-near" reward structure is proposed to mitigate robotic collision risks.

Similarly, Xiong et al. [114] address packing reliability by introducing a "Candidate Map" mechanism. This approach masks the action space, restricting selections to feasible, stable placements derived from Extreme Points (EPs) heuristic. The action space is represented as two distinct 2D maps, corresponding to the feasible positions for each of the two possible box orientations. The framework utilizes the Advantage Actor-Critic (A2C) algorithm, employing a shared CNN encoder to process multi-channel input (heightmap and box dimensions) for both policy and value estimation.

Addressing the computational cost of stability verification, Gao et al. [37] introduce the "Load Bearable Convex Polygon" (LBCP) method. This geometric heuristic rapidly computes feasibility masks by analyzing support areas. The framework integrates this

fast validation into a Stable Rearrangement Planning (SRP) system: when the immediate feasible action space is empty, a hybrid Monte Carlo Tree Search (MCTS) and an A* algorithm rearrange previously packed items to create valid placement opportunities.

Zhou et al. [128] address the computational latency of stability verification within the PPO framework. They integrate the convexHull-α algorithm, which ensures the box's Center of Mass is sustained by a structurally sound base, avoiding the high computational cost of full physics simulations.

More recently, Zhang et al. [121] introduced a physics-aware framework designed to handle items with varying densities and rigidities. To capture these complex physical attributes, the state representation is expanded into a 2-channel 3D voxel grid, encoding the bin's spatial and material distribution, alongside a 5-dimensional tuple representing the item's geometry and physical properties. The approach utilizes "Online Action Masking Learning", where a supervised segmentation model predicts stability masks based on physics engine feedback, effectively replacing manual heuristics with learned dynamics.

3.5.2. State Representation and Architecture Innovations

Verma et al. [105] propose a generalized approach designed to handle bins of arbitrary sizes without retraining. They achieve scale-independence by encoding the container's top-down height map into a fixed-size representation. The decision-making agent, termed "PackMan", utilizes a Deep Q-Network (DQN) to select the optimal placement from a discrete set of candidates. These candidates are generated by a novel constructive heuristic named "WallE", which mimics standard constructive algorithms by building walls of uniform height to ensure stability and maximizing packing density.

To capture global spatial topology more effectively than grid-based methods, Zhao et al. [122] introduce "Packing Configuration Trees (PCT)". This hierarchical state representation encodes packed items and empty spaces using Graph Attention Networks (GATs). Within this framework, an ACKTR agent selects the optimal placement from a set of leaf nodes, which correspond to candidate positions generated by standard heuristics such as Extreme Points or Empty Maximal Spaces.

Addressing the complexity of large-scale packing with long sequences, Li et al. [62] propose Recurrent Conditional Query Learning (RCQL) within a Soft Actor-Critic (SAC) framework. To avoid the computational overhead of global bin representations, they utilize a sequence-based state comprising the dimensions of incoming items and a finite buffer of recently packed boxes. The problem is formulated as semi-online, allowing the agent to select the next item from a sliding lookahead window. Consequently, the action space is

decoupled into item selection, orientation, and placement.
The RCQL mechanism employs a recurrent encoder with attention to process the packed item buffer, effectively propagating spatial information over time to approximate the global packing state.

Xiong et al. [116] propose a flexible state representation that explicitly pairs the dimensions of the incoming box, accounting for both orientations, with a sequence of EMS derived from the current bin configuration. The method, called GOPT, operates within a PPO framework and utilizes cross-attention to correlate item features with these available spaces. This architecture enables the policy to generalize effectively across bins of varying dimensions, while static stability is verified via a convex hull method.

3.5.3. Heuristic Integration and Multi-Objective Optimization

Several approaches explicitly embed heuristic logic or multiple objectives into the DRL framework to guide exploration.

Yang et al. [118] present "PackerBot," a framework based on PPO. The state represents box dimensions with complementary "corner action indexes," and the bin height map. They define an action mask based on "PackE," an algorithm combining multiple constructive heuristics. This mask is utilized to shape the reward function, introducing a penalty term if the agent's selected action diverges significantly from the actions suggested by the mask.

Building on this, Yang et al. [117] evolve the strategy by upgrading PackE to "Pack-Heu," which generates an expert map to directly mask the action space. Additionally, they propose "Phy-Heu," a feasibility estimator comprising a static heuristic and a neural network trained on physical simulations. The final action mask is derived from the intersection of the expert map and the feasibility mask. Implemented within the PPO framework, this method shapes the reward to penalize wasted space. It also introduces an Unpacking Heuristic to allow the agent to explore new packing trajectories when the loading process stagnates.

Wong et al. [110] propose Hybrid Heuristic PPO (HHPPO) to facilitate robust learning. The state is represented as a semantic 3D voxel grid that explicitly categorizes space into three states: occupied, supported empty space, and unsupported empty space. This encoding allows the agent to inherently perceive placement stability. The learning process is guided by two heuristic mechanisms: an EP sorting strategy based on minimizing wasted space and the DBLF rule. These heuristics are integrated via reward shaping, providing dense feedback to optimize packing efficiency.

3.5.4. Complex Manipulation and Rearrangement

Recent advancements have expanded the problem scope beyond simple placement to include object rearrangement and complex logistical constraints, building on the unpacking capabilities seen in [37, 117].

Addressing continuous throughput in multi-bin scenarios, Tsang et al. [103] apply a DDQN to Online 3D-BPP variants where item selection extends to a conveyor lookahead queue. They utilize an EMS-based state representation, extending the generation algorithm to simultaneously track available spaces for multiple candidate items. A key contribution is the Bin Replacement Strategy, which ensures operational continuity by intelligently determining the optimal timing for replacing filled bins. To ensure physical feasibility, the framework enforces a stability heuristic based on three conditions.

Finally, Hu et al. [50] invert the typical control hierarchy to address the Transport-and-Pack (TAP) problem. Unlike standard BPP, TAP requires extracting items from an obstructed initial configuration represented in the state as a dynamic precedence graph. "TAP-Net" employs an encoder-decoder architecture to solve this as a Set-to-Sequence task: the DRL agent learns the optimal unpacking sequence and orientation to resolve retrieval constraints; the specific placement coordinates in the target container are handled by a standard constructive heuristic.

3.5.5. Results

In this section, the experimental results are presented. The evaluated methods are grouped based on their problem settings and specific constraints: standard online 3D-BPP setups, large-scale strip packing, and physically constrained packing with rearrangement mechanisms.

Standard On-line 3D-BPP. In the standard online 3D Bin Packing Problem, evaluations are frequently conducted on discretized $10 \times 10 \times 10$ container resolutions. For instances containing 64 heterogeneous box types, uniformly sampled up to half the bin size (e.g., $1 \leq l, w, h \leq 5$), an average space utilization of 66.9% and an average of 17.5 packed boxes per bin are achieved [123]. When testing on 1000 dynamically generated instances comprising 125 distinct box types, superior volume utilization and a higher average number of packed boxes are consistently recorded compared to standard heuristic baselines [116]. Furthermore, when evaluating the impact of different single-objective heuristics on identical $10 \times 10 \times 10$ setups, space utilizations of 63.24%, 61.62%, 64.56%, and 61.77% are reported by optimizing for extreme point height, height variance, visible

area, and compactness, respectively [115].

Large-Scale and Strip Packing. For problem variants requiring massive scalability, specifically, on large-scale instances containing up to 1000 boxes, a 5.64% higher space utilization ratio is achieved, while the average bin gap ratio is reduced by 7.84% in 3D cases compared to state-of-the-art heuristic methods [62].

Physically Constrained Packing and Box Rearrangement. By combining object rearrangement strategies with stable placement heuristics, a space utilization of 62.5% is achieved on the CUT-2 dataset in only 18.6k training epochs, outperforming baseline performances that required up to 100k epochs [128]. Similarly, higher overall space utilization is successfully achieved through the introduction of unpacking mechanisms and a limited-capacity staging area, balancing maximal packing efficiency with computational time limits [117]. In more complex robotic palletization tasks, stable placement is successfully maintained by actively tracking box orientations and positional deviations to filter out structurally unstable states [37, 121]. Finally, strict obstruction constraints are effectively satisfied to generate reliable, physically realizable loading sequences in real-world transport-and-pack scenarios [50, 114].

3.6. Scheme of Reward Shaping Strategies

The design of the reward function is critical in Reinforcement Learning, as it dictates the gradient direction for policy updates. Thus, this section categorizes all of the strategies found in the literature.

Surface Area Minimization

Used primarily in early offline works like [49] and [27], the focus is on minimizing the surface area of the bounding box needed to pack all items (effectively minimizing wrapping material). The reward is negative, treating the surface area (SA) as a cost.

$$R = -SA,$$

where SA is the total surface area of the cuboid formed by the packed items.

Volume Utilization (Terminal vs. Step-Wise)

Volume utilization remains the standard metric. However, the timing of the reward differs significantly between offline and online approaches.

Terminal Reward: Standard offline approaches, such as [120], provide a sparse terminal reward based on the final filling ratio. This encourages the agent to maximize density within the final bounding box dimensions (L_{C_π}, W, H):

$$R_{terminal} = \frac{\sum_i^N l_i w_i h_i}{L_{C_\pi} W H}.$$

Step-Wise Volume Reward: To mitigate reward sparsity in sequential decision-making, the majority of online methods, including [62, 116, 122, 123, 128], assign a reward at each time step t proportional to the volume of the immediate item packed:

$$r_t = \frac{l_i w_i h_i}{LWH}.$$

In some variations, such as [52], this is normalized by the current maximum stack height $\tilde{H}_t$ rather than the total bin height H.

Terminal Packing Fraction with Baseline

Verma et al. [105] employ a terminal reward structure by comparing the agent's performance against its own historical average. They define a terminal score ζ as the global packing fraction minus a baseline τ, where τ represents the moving average of packing fractions achieved in previous episodes:

$$\zeta = \frac{V_{packed}}{T_{used} \cdot L \cdot B \cdot H} - \tau.$$

Here, V_{packed} is the total volume of packed items, and T_{used} is the number of bins utilized. If there are N boxes in a given episode, then the step reward at time step t is given by

$$r_t = \rho^{N-t}\zeta,$$

where ρ is a discount factor.

Step-wise Gap and Waste Reduction

Instead of rewarding filled volume, some approaches penalize the creation of empty space. [53, 88] propose a reward based on "gap reduction", where r_t is the reduction in wasted space compared to step $t-1$:

$$r_t = gap_{t-1} - gap_t.$$

Similarly, in the online domain, Zhou et al. [128] explicitly introduce a penalty term for wasted space to guide the PPO agent:

$$r_t = \alpha \cdot r_{vol} - \beta \frac{V_{waste}}{LWH}.$$

Structural Stability (Compactness, Pyramid, and Stability)

[103, 108] proposed a reward optimizing "Compactness" (C) and "Pyramid" (P) structures:

$$R = 5 \cdot \left[\alpha \cdot \left(1 - \frac{\sum C}{k-1} \right) + \beta \cdot \left(1 - \frac{\sum P}{k-1} \right) \right],$$

where P encourages items to rest on a solid base without overhangs.

In the online context, Hu et al. [50] (TAP-Net) utilize a similar composite metric averaging Compactness, Pyramidality, and a specific Stability term (S):

$$R = (C + P + S)/3,$$

where S is defined as the percentage of items whose center of mass falls within the convex hull of their supporting points.

Heuristic Conformity

A trend specific to online methods involves shaping rewards to accelerate training by penalizing deviations from known heuristics. Yang et al. [118] (PackerBot) define a reward that penalizes the agent if the selected action diverges from candidates suggested by the "PackE" heuristic ($\mathcal{E}$):

$$R_{heur} = w_h \left(\exp \left[- \min_{(e_x, e_y) \in \mathcal{E}} \|(x, y) - (e_x, e_y)\|_2 \right] - 1 \right).$$

Similarly, [110] design a hybrid reward function that adapts dynamically based on the chosen action type. The total reward aggregates the average placement score, a penalty for unused space at the container's bottom, and a global volume utilization term. The step reward r_t is conditional:

$$r_t = \begin{cases} \rho \cdot S_{score} - \mu \cdot W_{waste} & \text{if } a_t \in \{0, 1\}; \\ \omega \cdot r_{t-1} + \tau \cdot S_{score} & \text{if } a_t = 2, \end{cases}$$

where S_{score} is the score of occupied grids and W_{waste} is the generated wasted space. When the agent selects specific EPs (actions 0 or 1), it is penalized directly for waste. However, when it selects the heuristic-derived move (action 2, based on priority sorting), the reward blends the current score with the previous step's reward (r_{t-1}). This mechanism prevents the strong heuristic signal from overly dominating the policy learning.

Robotic Feasibility and Safety

[124] introduce a "far-to-near" packing incentive. To prevent the robot arm from colliding with previously packed items, they reward placements that are closer to the back of the bin relative to the loading position:

$$r_{safe} = \beta \frac{V_{safe}}{LWH},$$

where V_{safe} represents valid loading positions reachable without trajectory collisions.

Ranked Reward (R2)

To handle the single-player nature of offline BPP, Laterre et al. [61] propose a relative performance metric. The reward is binary, determined by comparing the agent's performance r against a dynamic threshold B_α derived from its own recent history:

$$z = \begin{cases} 1 & \text{if } r > B_\alpha, \\ -1 & \text{if } r < B_\alpha. \end{cases}$$

3.7. Stability Assessment Methodologies in Online 3D-BPP

Ensuring physical stability is a fundamental constraint in robotic packing. The approaches to assess placement validity can be categorized into three main streams: geometric heuristics based on the Center of Mass (CoM), rule-based surface support conditions, and data-driven physics predictions.

3.7.1. Geometric Heuristics and Center of Mass Analysis

The most prevalent methods rely on the "static equilibrium" assumption, verifying that the item's projection falls within the support polygon formed by items below. Here we list the most common implementations:

- **Standard Convex Hull (CoM Support):** A widely adopted criterion considers an item stable if its Center of Mass (CoM) projected onto the XY plane lies strictly within the **convex hull** of its supporting points. [50] (TAP-Net) utilizes this method to compute the stability score S for their composite reward function.
  [114, 116, 128] verify the stability for the placement using the convex hull method and mask all unstable actions.
- **Load Bearable Convex Polygon (LBCP):** [37] refines this approach with the **LBCP** method. This technique conservatively ensures structural stability by analyzing the exact contact interface between stacked layers. It computes a feasibility mask derived from the convex hull of the intersection between the box's base and the supporting surfaces, ensuring the center of mass rests on a valid load-bearing region.
- **ConvexHull-α (Global Connectivity):** [128] proposes the **convexHull-α** algorithm to address the limitations of local surface checks. Unlike standard methods that only check immediate contact, this algorithm maintains a global "empty map" to identify and filter out "wasted voxels" (unsupported spaces lower in the stack). The convex hull is then computed solely on grounded support points, preventing the agent from placing items on stacks that appear stable locally but are structurally unsound.

3.7.2. Surface Support Conditions and Voxel Logic

Other approaches approximate stability by analyzing the contact surface area or using graph-based structures, avoiding explicit polygon computation. Namely:

Support Ratio and Stacking Trees: [124] introduces a **"Stacking Tree"** structure to efficiently track support relationships. They assess stability by calculating the support ratio—the percentage of the item's base area in contact with the items below. This method allows for $O(N \log N)$ stability analysis, replacing slower checks.

Semantic Voxel States: [110] encodes stability directly into the state representation. They utilize a semantic 3D voxel grid that explicitly categorizes space into "occupied", "supported empty space", and "unsupported empty space". This allows the agent to inherently perceive stability rules without an external calculation step during action selection.

Heuristic Alignment and Support Rules: [103] enforces feasibility through a two-step validation process. First, physical stability is approximated by requiring that at least 50% of the item's base be supported by the floor or underlying items. Second, placements

must align with the axes of the EMS, explicitly prohibiting items from being placed in the center of free spaces without boundary contact.

Graduated Support Thresholds: [123] defines a conservative stability mask based on three graded criteria. A placement is considered feasible if it satisfies any of the following support conditions:

1. over 60% of the bottom area supported with all four corners grounded;
2. over 80% of the area supported with at least three corners grounded;
3. over 95% of the total bottom area supported, allowing strictly limited overhangs only when the mass is almost fully grounded.

3.7.3. Physics-Informed and Learned Stability

To handle complex dynamics (e.g., varying densities, friction, soft bodies) that simple geometry cannot model, recent works employ Neural Networks trained on physics engines. Among others:

- **Phy-Heu (Hybrid Estimator):** Yang et al. [117] propose **Phy-Heu**, a feasibility estimator composed of two parts: a fast static heuristic for initial filtering and a Neural Network trained on data collected from a physical simulator. This network learns to predict the probability of stability for a given configuration, creating a "feasibility mask" that is intersected with the expert heuristic map.
- **Online Action Masking Learning:** Zhang et al. [121] introduce a fully physics-aware framework. They employ a supervised segmentation model to predict stability masks. The ground truth for this model comes directly from **physics engine feedback** (e.g., PyBullet/MuJoCo simulations) considering material properties and dynamics, effectively replacing manual geometric heuristics with learned physical dynamics.

3.7.4. Soft Constraints via Reward Penalties

Finally, some approaches do not strictly mask unstable actions but discourage them through reward shaping. Verma et al. [105] integrate stability into the objective function by penalizing the height and horizontal deviation of the packing's collective **Center of Mass**. This encourages the agent to learn policies that naturally keep the load low and centered, promoting stability implicitly.

3.8. Summary of Methodologies and Results

In this section, a comprehensive comparison of the different methodologies and a summary of the experimental results are provided. The offline solvers are detailed in Table 3.1 and Table 3.2, while the online configurations are synthesized in Table 3.3 and Table 3.4.

The notation adopted across these tables is defined as follows:

hm (heightmap): A single-channel 2D map of shape (L, W) representing the current surface height of the packed volume.

fm (feature map): A 7-channel 2D tensor of shape $(7, L, W)$, where the first channel corresponds to the heightmap and the remaining six channels encode geometric and plane-occupancy features.

fv (frontier view): A compact representation of the exposed "frontier" surfaces of the current packing configuration.

vg (voxel grid): A full 3D occupancy grid of the current bin, typically represented as a binary or multi-channel tensor.

d^3**:** Dimension triplet (l, w, h) of a single box.

d^5**:** Item vector including dimensions and physical properties: length (l), width (w), height (h), density, and rigidity.

$d^{n\times 3}$**:** Matrix of dimensions for n look-ahead boxes, where n denotes the prediction horizon.

$d^{N\times 3}$**:** Matrix of dimensions for all N boxes in the order sequence.

$d^{N\times \mathcal{O}\times 3}$**:** Matrix of box dimensions for all N items, considering all $\mathcal{O}$ feasible rotations.

$d^{N\times \mathcal{O}\times 7}$**:** Richer item tensor used in [52, 53], where four additional channels encode the packed/unpacked status and the (x, y, z) placement coordinates of already packed items.

dm**:** Representation of an item as a 3-channel map $(3 \times W \times L)$, where each channel is the projection of a specific dimension (w, l, h).

$\mathcal{M}^c$**:** Heuristic reduction of the set of possible placement coordinates in the bin (action masking).

1^{rp}**:** One-hot encoding of the receptive field surrounding a target placement cell.

$hm_{multibin}$**:** Heightmap representing T bins: $W \times (L \cdot T)$, where T is the number of bins.

d-hm**:** Height values of the cells bordering the proposed placement location of the box.

$\mathbf{B}_t$**:** Internal nodes in a tree structure representing the configuration of already packed items.

$\mathbf{L}_t$**:** Leaf nodes representing the packable candidate placements.

ctx**:** Context window in semi-online 3D-BPP, representing the visibility of upcoming items.

$d_p^{N\times 6}$**:** Packed boxes represented by dimensions (w, l, h) and placement coordinates (x, y, z).

d_{bound}^{6}**:** Information regarding the upcoming box including its spatial boundaries.

Paper	State Space	Action Space	Function Approximator	DRL Framework
[49]	$d^{N\times 3}$	$idx \in [1, N]$	Ptr-Net over RNN	Reinforce
[27]	$d^{N\times\mathcal{O}\times 3}$	$idx \in [1, N]$ $o \in \mathcal{O}$	Ptr-Net over Attn-Encoder	PPO
[108]	$\langle d^{N\times\mathcal{O}\times 3}, hm\rangle$	$idx \in [1, N]$	Ptr-Net over GRU and CNN	Actor-Critic
[61]	not specified	$idx \in [1, N]$ $o \in \mathcal{O}$ $(x, y) \in (L, W)$	not specified	Self-play
[52, 53]	$\langle d^{N\times\mathcal{O}\times 7}, hm\rangle$	$idx \in [1, N]$ $o \in \mathcal{O}$ $(x, y) \in (L, W)$	Transformer and CNN	A2C
[120]	$\langle d^{N\times\mathcal{O}\times 3}, fv\rangle$	$idx \in [1, N]$ $(x, y, o) \in (L, W, \mathcal{O})$	Transformer and CNN	GPOMDP with baseline
[88]	$\langle d^{N\times\mathcal{O}\times 3}, fm\rangle$	$idx \in [1, N]$ $o \in \mathcal{O}$ $(x, y) \in (L, W)$	Transformer and CNN	PPO
[119]	$\langle d^{N\times\mathcal{O}\times 3}, hm\rangle$	$idx \in [1, N]$ $(x, y) \in (L, W)$	Transformer and CNN	Policy-Based

Table 3.1: Comparison of the main modeling choices and algorithmic differences among DRL solvers for the offline 3D-BPP, covering state and action formulations, neural architectures, and training frameworks.

Paper	Results (Volume Utilization / Bin Resolution)
[49]	+7.84% space reduction over SOTA heuristics (Tested on $100 \times 100 \times 100$ and larger instances)
[61]	70%–78% average utilization (Tested on Cut-1 and Cut-2 datasets, 10^3 res.)
[52]	83.1% average utilization with 50 items (Standard $10 \times 10 \times 10$ resolution)
[120]	87.0% (100 items); 83.1% (50 items) (Bin resolution $10 \times 10 \times 10$)
[53]	83.1% average utilization (50 items) Integrated DRL and Constraint Programming approach
[88]	82.4% (100×100); 76.7% (400×200) Scalable to highly heterogeneous item sets
[119]	∼80.0% utilization on large-sized containers Superior performance via two-stage DRL
[27]	+9.66% improvement over Greedy (BIN10/12) 5.47% average cost reduction in A/B tests
[108]	75%–82% average utilization (10^3 res.) Tested on instances with up to 200 items

Table 3.2: Performance summary of offline 3D-BPP solutions using the specified citations.

Paper	State Space	Action Space	Function Approximator	RL Algorithm
[123]	$\langle d^{\mathcal{O}\times 3}, hm\rangle$	$o \in \mathcal{O}$ $(x, y) \in (L, W)$	CNN + MLP	ACKTR
[124]	$\langle d^{\mathcal{O}\times 3}, hm, \mathcal{M}^c\rangle$	$o \in \mathcal{O}$ $x \in L, y \in W$	CNN + MLP	ACKTR
[114]	$\langle d^3, hm, \mathcal{M}^c\rangle$	$(o, x, y) \in (\mathcal{O}, L, W)$	CNN + MLP	A2C
[128]	$\langle dm, hm\rangle$	$o \in \mathcal{O}$ $(x, y) \in (L, W)$	not specified	PPO + heuristics
[121]	$\langle d^5, vg\rangle$	$(o, x, y) \in (\mathcal{O}, L, W)$	not specified	PPO
[105]	$\langle hm_{multibin}, \mathcal{M}^c,$ $d - hm, 1^{rp}\rangle$	$\text{argmax}\{V_t\}$	MLP	DQN
[122]	$\langle \mathbf{B}_t, \mathbf{L}_t\rangle$	$idx \in \mathbf{L}_t$	MLP + GAT	ACKTR
[62]	$\langle d_p^{N\times 6}, d^{ctx}, 3\rangle$	$idx \in ctx,$ $o \in \mathcal{O}$ $(x, y) \in (ns, ns)$	Transformer	SAC
[116]	$\langle EMS, d^{3\times\mathcal{O}}\rangle$	$o, ems \in$ $(\mathcal{O}, \lvert EMS\rvert)$	Transformer	PPO
[118]	$\langle d^6_{bound}, hm, \mathcal{M}^c\rangle$	$(o, x, y) \in (\mathcal{O}, L, W)$	MLP	PPO
[117]	$\langle d^3, vg, \mathcal{M}^c\rangle$	$(o, x, y) \in (L, W, H)$	3D CNN + MLP	PPO + heuristics
[110]	$\langle d^3, vg, \mathcal{M}^c\rangle$	$(o, x, y) \in (L, W, H)$	not specified	HHPPO
[103]	$\langle d^{ctx\times 3}, hm\rangle$	$idx \in ctx,$ $o \in \mathcal{O}$ $(x, y) \in (ns, ns)$	CNN + MLP	DQN
[115]	$\langle d^3, hm\rangle$	$(o, x, y) \in (\mathcal{O}, L, W)$	CNN + MLP	A2C

Table 3.3: Comparison of modeling choices for online 3D-BPP solvers, detailing state representations, action spaces, architectures, and reinforcement learning algorithms.

Paper	Results (Volume Utilization / Bin Resolution)
[123]	71.7%–78.3% average utilization (Bin resolution $10 \times 10 \times 10$ e $20 \times 20 \times 20$)
[124]	76.5%–81.1% average utilization (Decoupled dimension learning, high-res discretization)
[114]	76.4% average utilization (Physical stability verified on real robot system)
[121]	82.1% average utilization (Rigorous setting) (Online masking inference for structural stability)
[105]	67.0%–75.0% utilization on varying bin sizes (Preview window $k = 1$ and $k = 5$)
[122]	78.2%–84.6% average utilization (Continuous solution space via hierarchical tree)
[62]	+5.12% space reduction over online heuristics (Adaptive to thousands of items via recurrent queries)
[116]	78.5%–82.3% average utilization (Generalization across multiple bin dimensions)
[118]	74.8% average utilization (Simulation) (Heuristic DRL for variable-sized products)
[117]	77.2% average utilization (Heuristics Integrated) (Stable packing with physics and packing heuristics)
[110]	79.1% average utilization (Hybrid HHPPO) (Standard benchmarks, improved over state-of-the-art)
[50]	72.4%–76.8% average utilization (Transport-and-pack sequence optimization)
[103]	76.5%–83.2% average utilization (Dual bin replacement strategy for concurrent packing)
[128]	81.4% average utilization with object rearrangement (Significant utilization gain through stable relocation)
[115]	DRL-based approach vs. MOO heuristics (Superiority of candidate map-based DRL demonstrated)

Table 3.4: Summary of performance results for DRL-based online 3D-BPP solutions.

4 | Problem Statement

In this work, we focus on the offline scenario of the Single Knapsack Problem (SKP) [Table 2.1], in which the primary objective is to maximize the volumetric occupancy of a defined set of boxes into a single bin.

The central research question is formulated as follows:
"Is it possible to rapidly generate solutions for the SKP that strictly satisfy geometric, orientation, stability, and pattern complexity constraints?"
In this chapter, we aim to translate such a question into a formal mathematical framework.

4.1. Problem Assumptions and Constraints

To ensure a rigorous formulation, specific assumptions regarding the physical properties of the entities involved and the operational constraints are defined.

Boxes (Items). The set of items to be packed consists of rectangular cuboids with uniform mass distribution; consequently, the Center of Mass (CoM) coincides with the geometric center. Each box is characterized by four scalar properties: length (l), width (w), height (h), and mass (m). Boxes are categorized by "article" (SKU); each article possesses a unique identifier and fixed dimensions, meaning all boxes belonging to the same article are identical in shape and size.

Bins (Containers). The containers are identical rectangular cuboids defined by three dimensions: length (L), width (W), and maximum height (H). The bin dimensions are strictly standardized to the Euro-pallet specifications (1200×800 mm) or defined sub-multiples. Operationally, a sequential usage constraint is applied: bins are filled one at a time. A bin is considered "closed" when no further boxes can be feasibly allocated to it, at which point a new bin is initialized. Once a bin is closed, it cannot be reopened for further loading.

Operational Constraints. Any feasible solution is required to satisfy the following set of constraints (refer to [Par. 2.1.2] for detailed definitions):

- **Geometric Containment:** All packed boxes must lie entirely within the spatial boundaries of the bin.
- **Non-overlapping:** No two boxes may occupy the same volume in space.
- **Orthogonality:** Boxes must be placed with their faces parallel to the walls of the bin.
- **Orientation:** Box rotation is restricted to the vertical z-axis. Boxes may be rotated by 90° on the horizontal plane, but tilting is prohibited.
- **Stability Constraints:**
  - *Local Vertical Support:* Every box must be supported from below by either the bin floor or the top surface of other boxes.
  - *Sequential Global Stability:* Ensures that the altered mass distribution resulting from each new placement does not compromise the static equilibrium of the entire stack.
- **Pattern Complexity and Accessibility:** As the loading process is performed by a robotic arm, the solution must respect kinematic accessibility. The placement sequence must account for an "entrance line" to prevent collisions between the robotic manipulator and previously placed items.

4.2. Mathematical Formulation

The packing process is inherently sequential: at each discrete time step t, a new box is selected, assigned an orientation, and positioned within the active bin. Since the optimal subsequent placement depends solely on the remaining unpacked boxes and the current spatial configuration of the active bin, the system naturally satisfies the Markov property (memoryless condition). Consequently, the problem is formally modeled as a Constrained Markov Decision Process (CMDP).

Let $\mathcal{I}_t$ be the set of remaining unloaded physical boxes at time t, where each box $i \in \mathcal{I}_t$ has spatial dimensions (l_i, w_i, h_i). This physical set is mapped into a structured inventory state representation, denoted as s_t^u. Let $\mathcal{C}_t$ represent the spatial state of the currently active bin, characterized by maximum dimensions $L \times W \times H$ and a current maximum stacked height $\tilde{H}_t$.

The global state of the environment at time t is formulated as the tuple encompassing both the inventory representation and the active bin configuration:

$$\mathcal{S}_t = \langle s_t^u, \mathcal{C}_t \rangle. \tag{4.1}$$

The action space at time t involves selecting a specific box type from s_t^u, choosing a valid horizontal orientation from the set $\mathcal{O} = \{0, 1\}$, and defining a spatial placement coordinate within $\mathcal{C}_t$. Its maximum theoretical size is bounded by the Cartesian product $|s_t^u| \times |\mathcal{O}| \times |\mathcal{C}_t|$.

Let R_t be the scalar reward received after executing an action at time t. The reward function is fundamentally designed to encourage the agent to pack items densely while minimizing the total number of bins utilized. A natural baseline metric for the packing problem is the episodic volumetric occupancy, computed at the terminal step of each bin as the ratio between the total packed volume and the maximum bin capacity:

$$R_{\text{episodic}} = \frac{\sum_{i \in \mathcal{P}} l_i w_i h_i}{LWH}, \tag{4.2}$$

where $\mathcal{P}$ is the set of successfully packed items.

To ensure physical and operational viability, the maximization of this objective function is strictly subject to a set of hard physical constraints within the active bin. Let (x_i, y_i, z_i) represent the spatial coordinates of the newly selected box i, and $o_i \in \mathcal{O}$ denote its selected horizontal orientation. The feasible action space is bounded by the following conditions:

$$x_i + l_i(o_i) \leq L \tag{4.3}$$

$$y_i + w_i(o_i) \leq W \tag{4.4}$$

$$z_i + h_i \leq H \tag{4.5}$$

$$\begin{aligned} &[x_i + l_i(o_i) \leq x_k] \vee [x_k + l_k(o_k) \leq x_i] \vee \\ &[y_i + w_i(o_i) \leq y_k] \vee [y_k + w_k(o_k) \leq y_i] \vee \quad \forall k \in \mathcal{C}_t \\ &[z_i + h_i \leq z_k] \vee [z_k + h_k \leq z_i] \end{aligned} \tag{4.6}$$

Geometric containment is mandated by Eqs. (4.3)–(4.5), requiring that the placed box fits entirely within the active bin's dimensions, accounting for its specific orientation o_i. Furthermore, non-overlapping conditions are enforced by Eq. (4.6), ensuring that the spatial domain of the newly placed box i does not intersect with the domain of any previously packed box k currently residing in the active bin.

While this general CMDP framework establishes the theoretical mathematical bounds of the 3D-BPP, solving it directly via reinforcement learning is computationally intractable due to the combinatorial nature of the continuous state-action space. Building upon this theoretical foundation, the following section details the specific, optimized MDP formulation developed for this research, introducing the feature-rich state representations, the heuristically reduced action spaces, and the dense reward signals required to make the learning process viable.

Before detailing the MDP, a fundamental geometric relaxation is defined.

Geometric Approximation: The Pseudo-Planar Support

In real-world applications, strict planarity of the supporting surface is not a rigid requirement for stability. Small height differentials between adjacent boxes often form a *step* that can securely support a new box.

To align the simulation with these industrial conditions, the concept of a **Pseudo-Planar Surface** is introduced. A region of the bin's top surface is treated as effectively planar if it satisfies specific tolerance criteria. Primarily, the height difference Δh between adjacent supporting points must fall within a defined threshold δ_h (typically defined as a small percentage of the box height).

Furthermore, the strict planarity constraint is relaxed in the presence of narrow gaps. If a local depression exists where $\Delta h > \delta_h$, it is mathematically ignored if its maximum spatial dimensions are strictly smaller than a theoretical minimum bounding volume $w_{min} \times w_{min} \times w_{min}$ (where w_{min} represents the minimum possible dimension across all available boxes). Since no physical box could feasibly intersect with or fall into such small cavities, the surface spanning these gaps is considered structurally viable.
This relaxation significantly increases the achievable packing density by enabling the stacking of boxes across slightly uneven or porous surfaces, without violating the stability constraints.

The process of filtering the heightmap to ignore negligible gaps is mathematically equivalent to a morphological closing operation, consisting of a dilation (maximum filter) followed by an erosion (minimum filter). The procedure is formalized in Algorithm A.3.
Once the geometric filtering is applied, the verification of the pseudo-planarity condition becomes a strict equality check over the filtered representation, as detailed in Algorithm A.4.

4.2.1. State Representation

As established, the environment state $\mathcal{S}_t$ at any time step t is bipartite, comprising the observation of the remaining inventory and the spatial configuration of the active bin. Formally, $\mathcal{S}_t = \langle s_t^u, s_t^c \rangle$.

Unpacked Items State (s_t^u)

Let s_t^u denote the state of the remaining unpacked boxes. To manage the inventory efficiently, regardless of the absolute number of boxes, identical items are grouped into clusters. Furthermore, to embed the rotational degrees of freedom directly into the state observation, each physical box type is expanded into two distinct virtual clusters corresponding to its valid horizontal orientations.

Assuming there are N_k unique physical box types in the current order, the state is represented as a list of $2N_k$ virtual clusters:

$$s_t^u = [\nu_{1,0}, \nu_{1,1}, \nu_{2,0}, \nu_{2,1}, \dots, \nu_{N_k,0}, \nu_{N_k,1}]. \tag{4.7}$$

Each virtual cluster is defined by the tuple $\nu_{k,o} = (l_k(o), w_k(o), h_k, o, q_k)$, where:

- $l_k(o)$ and $w_k(o)$ denote the effective spatial length and width of the box type k under the specific orientation $o \in \mathcal{O}$.
- h_k denotes the vertical height of the box, which remains invariant under horizontal rotation.
- $o \in \{0, 1\}$ explicitly indicates the embedded orientation state.
- $q_k \in \mathbb{N}$ represents the active quantity counter of available physical boxes belonging to this specific type.

Since the virtual pairs $\nu_{k,0}$ and $\nu_{k,1}$ represent the exact same physical inventory, their quantity counters are strictly coupled. Whenever an item belonging to either cluster is selected, the shared physical inventory counter q_k is decremented by exactly one unit, dynamically updating the state for both orientations simultaneously.

For the sake of brevity, any subsequent reference to selecting or placing "an item from s_t^u" implicitly denotes a single physical box already configured with the specific horizontal orientation extracted from the cluster $\nu_{k,o}$.

Container State ($\mathcal{C}_t$)

The configuration of the active container, denoted as $\mathcal{C}_t$, is decomposed into two distinct components: a spatial feature map s_t^c and a set of heuristic action masks $\mathcal{M}_t^c$. Formally, this is expressed as $\mathcal{C}_t = \langle s_t^c, \mathcal{M}_t^c \rangle$.

Let $s_t^c \in \mathbb{Z}^{L \times W \times 5}$ represent the observable spatial state of the active container. Following the methodology proposed in [88], the 3D volume is projected into a 5-channel 2D spatial feature map:

$$s_t^c = \begin{bmatrix} \mathbf{v}_{11} & \mathbf{v}_{12} & \dots & \mathbf{v}_{1W} \\ \mathbf{v}_{21} & \mathbf{v}_{22} & \dots & \mathbf{v}_{2W} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{v}_{L1} & \mathbf{v}_{L2} & \dots & \mathbf{v}_{LW} \end{bmatrix}, \tag{4.8}$$

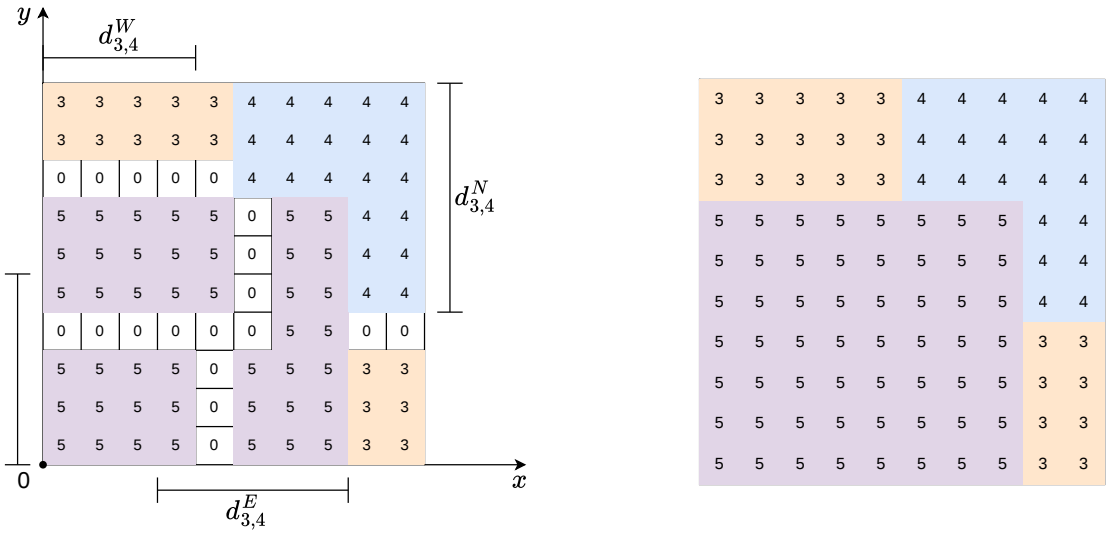
where L and W refer to the discrete length and width of the bin grid, respectively. Each element $\mathbf{v}_{ij}$ represents a 5-dimensional feature vector at the top-down grid coordinate (i, j), defined as:

$$\mathbf{v}_{ij} = \{h_{ij}, d_{ij}^N, d_{ij}^S, d_{ij}^E, d_{ij}^W\}. \tag{4.9}$$

The primary channel, h_{ij}, indicates the current maximum stacked height at position (i, j). The remaining four channels $\{d_{ij}^N, d_{ij}^S, d_{ij}^E, d_{ij}^W\}$, referred to as *plane features* [88], provide explicit structural information regarding the extension of the **pseudo-planar** surface. Specifically, they measure the contiguous distance from the coordinate (i, j) to the nearest edge or height discontinuity in the four cardinal directions (North, South, East, West) across the active support plane.

To illustrate, consider a 10×10 container grid [Fig.4.1]. The stacked height h_{ij} defines the base topography. For any given position (i, j) of the base heightmap, the plane features explicitly quantify how far an item can extend along the x and y axes before encountering a drop or an obstacle.
The plane features are extracted from the pseudo-planar heightmap $\mathcal{H}_{plane}^m$, as detailed in Algorithm A.5.

The secondary component of the container state, denoted by the mask set $\mathcal{M}_t^c$, enforces geometric boundaries, ensures physical stability, and heuristically reduces the overall action space. $\mathcal{M}_t^c$ is structured as a collection of binary spatial masks, in one-to-one correspondence with the items in s_t^u. For each available item in s_t^u, its associated mask



(a) Example of a 10×10 feature map

(b) Example of a 10×10 pseudo-planar heightmap

Figure 4.1: Construction of the spatial plane features (a) and the corresponding pseudo-planar heightmap (b), considering a minimum box dimension $w_{min} = 2$.

$\mathbf{M} \in \{0,1\}^{L \times W}$ acts as a dedicated action filter. It evaluates the current topography of the bin and marks only a limited subset of grid cells as feasible placement coordinates. The detailed logic governing the construction and application of these masks is presented in [Par. 5.2].

4.2.2. Feasibility and Rigorous Stability Validation

To enforce the Stability Constraint, the stability validation approach of *Load Bearable Convex Polygon* (LBCP) framework introduced in [37] is followed, which models load-bearing regions as 2D convex polygons. Each successfully packed item contributes a new LBCP on its top surface, while a static ground LBCP $(P_0^{\triangle}, 0)$ is permanently defined at the bottom of the bin. An item is considered statically stable if its Center of Mass (CoM), which accounts for a bounded weight distribution uncertainty δ_{CoM}, lies entirely within its designated support polygon.

To integrate this framework with the gap-tolerant heuristics developed in this work, the support polygon $P^{\triangle}_{\text{new}}$ is computed leveraging the pseudo-planar heightmap $\mathcal{H}^m_{plane}$ rather than the raw heightmap $\mathcal{H}^m_t$. This ensures that micro-variations within the tolerance threshold δ_h are correctly recognized as a continuous supporting surface. The evaluation follows these procedural steps:

1. Compute the physical support height h^s based on the maximum elevation within the item's spatial footprint.
2. Identify the contact points $\mathrm{PS}_{\mathrm{contact}}$ by extracting the cells where the pseudo-planar heightmap $\mathcal{H}^m_{plane}$ perfectly matches the placement plane.
3. Determine the structurally feasible regions $\mathrm{PS}_{\mathrm{feasible}}$ from the global feasibility map $\mathcal{F}^m_t$, which acts as the cumulative 2D projection of all active LBCPs.
4. Calculate the new support polygon as the Convex Hull (CH) of the intersected active areas:
$$P^{\triangle}_{\mathrm{new}} \leftarrow \mathrm{CH}(\mathrm{PS}_{\mathrm{contact}} \cap \mathrm{PS}_{\mathrm{feasible}}). \tag{4.10}$$
5. Validate stability: the placement is structurally sound if the expanded CoM region, C_{new}, satisfies $C_{\mathrm{new}} \subseteq P^{\triangle}_{\mathrm{new}}$.

If the placement is evaluated as stable, $P^{\triangle}_{\mathrm{new}}$ is officially added to the active LBCP set. Subsequently, the global feasibility map must be refreshed through a **Structural Stability Update (SSU)**.

The baseline SSU algorithm is purely additive: it simply superimposes $P^{\triangle}_{\mathrm{new}}$ onto the existing feasibility map. However, this naive addition fails to mask out the non-load-bearing regions of the newly placed item, potentially leading to critical false positives in future stability validations. To rectify this, the modified SSU proposed in Algorithm A.2 strictly enforces a boolean override: all coordinates on the new item's top face that fall outside the calculated $P^{\triangle}_{\mathrm{new}}$ are explicitly set to `false` (0). This modification guarantees that $\mathcal{F}^m_{t+1}$ accurately mirrors the true structural support surface, preventing the DRL agent from stacking future items on physically unsupported cantilevers. See Algorithms A.1, A.2 for the detailed procedure.

4.2.3. Action Space Representation

A fundamental strategy in packing problems is to decompose the decision-making process into two sequential sub-actions: first, selecting a specific box type from the inventory state s^u_t, and subsequently determining its optimal spatial coordinate within the active bin. The placement coordinate is defined by the Loading Position (LP), which corresponds to the spatial location of the box's Front-Left-Bottom (FLB) corner.

The full theoretical action space is therefore defined by the Cartesian product of the

available oriented clusters and the spatial grid dimensions:

$$\mathcal{A} = 2N_k \times L \times W \times H. \tag{4.11}$$

However, in standard bottom-up packing heuristics, the vertical placement (z-coordinate) is implicitly and deterministically established by the physical support surface (i.e., the current heightmap) once the planar coordinates (x, y) are selected. Thus, the effective action space is reduced to the 2D grid:

$$\mathcal{A}_{\text{eff}} = 2N_k \times L \times W. \tag{4.12}$$

Furthermore, the geometric containment constraint is explicitly enforced directly upon this effective action space. Any spatial coordinate (x, y) on the grid that would cause a selected item to protrude beyond the physical boundaries of the container (L or W) is deterministically marked as unfeasible. As detailed in the subsequent formulation of the container state masks [Par. 5.2].

4.2.4. Reward Function

As outlined in the general CMDP formulation, episodic volumetric utilization provides a natural baseline objective. However, relying solely on such a delayed and sparse reward signal hinders sample efficiency and policy convergence during training. To address this limitation and provide immediate, actionable feedback to the agent, a dense, step-wise reward formulation is introduced. This step-wise reward is composed of three primary elements: a positive gap-reduction term, a structural dead-space penalty, and an instability penalty.

Component 1: Gap Reduction (r_t^{gap}). This component incentivizes the agent to maximize area utilization within the current layer before expanding vertically. It rewards the step-by-step reduction of the bounding empty space:

$$\begin{aligned} r_t^{\text{gap}} &= g_{t-1} - g_t, \\ g_t &= \left(LW\tilde{H}_t\right) - \sum_{i \in \mathcal{P}_t} l_i w_i h_i, \end{aligned} \tag{4.13}$$

where $\tilde{H}_t$ is the current maximum elevation in the bin (as defined by the heightmap) at step t, and $\mathcal{P}_t$ is the subset of items packed up to step t. By minimizing g_t, the policy is

mathematically driven to select actions that fill existing voids rather than increasing the overall pile height $\tilde{H}_t$.

Component 2: Dead Space Penalty (r_t^{dead}). While the gap reduction reward promotes density, it does not explicitly penalize the creation of permanently inaccessible voids. To explicitly discourage configurations that leave fragmented, unusable regions, a strict heuristic penalty is applied, evaluating the immediate topological consequences of the current placement.

This is achieved by analyzing both the *forward projected space* (blocked lanes) and the *backward covered space* (depressions). Let the newly packed item have a spatial footprint defined by the coordinates $[x_{start}, x_{end}) \times [y_{start}, y_{end})$ and height h. The algorithmic evaluation of the wasted volume is formalized in Algorithm A.7.

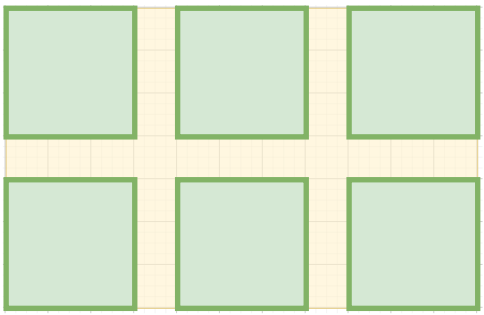
As showcased in Fig. 4.2, the loading of a specific item can irreversibly "remove" previous valid Corner Points. If the remaining gaps are smaller than the fictitious $w_{min} \times w_{min}$ footprint, they are flagged as wasted space.

Final Step-wise Reward (R_t). The ultimate scalar reward observed by the agent at step t consolidates these three components into a unified piecewise function. The first two components are aggregated as a weighted sum during successful placements.
Let w_{gap} and w_{dead} denote their respective weighting hyperparameters, the final reward R_t is computed as:

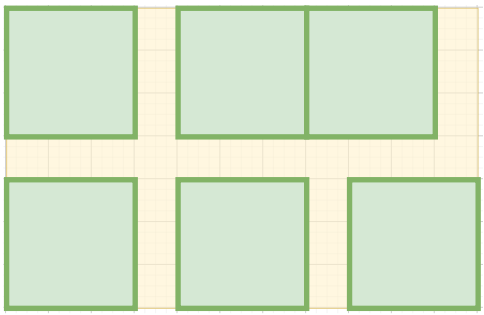
$$R_t = \begin{cases} w_{gap} r_t^{\text{gap}} + w_{dead} r_t^{\text{dead}} & \text{if the action is strictly stable;} \\ -\dfrac{L \times W}{50} & \text{if the action is unstable.} \end{cases} \tag{4.14}$$

4.3. Dataset

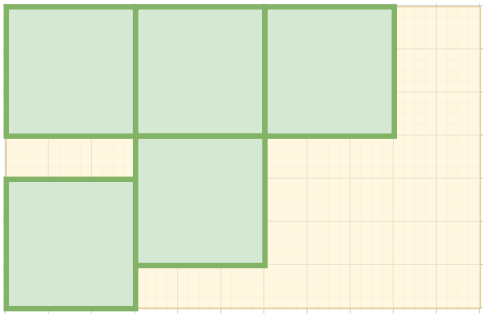
To ensure reproducibility and comparability of results, the benchmarking dataset proposed by Kagerer et al. [56] is adopted as the baseline. The dataset consists of distinct "orders", where each order comprises a set of items $\mathcal{I}$ and associated metadata. As illustrated in the listing 4.4, the data is structured as a dictionary. The target bin is defined in the "properties" field and adheres to standard dimensions: either a "Euro-Pallet" (1200 × 800 mm) or a "Rollcontainer" (800 × 700 mm).



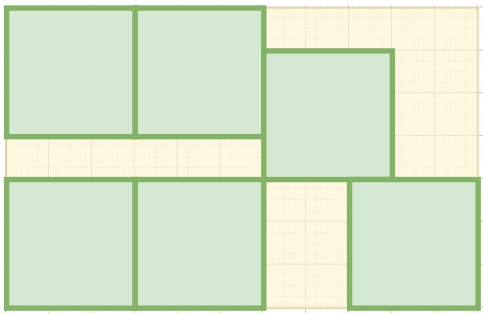
(a) $r_2 = 0$



(b) $r_2 = -1 \cdot 3 \cdot 3$

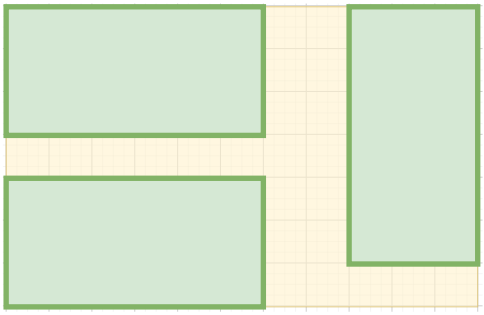


(c) $r_2 = -(3 \cdot 1 \cdot 3 + 2 \cdot 3 \cdot 3)$

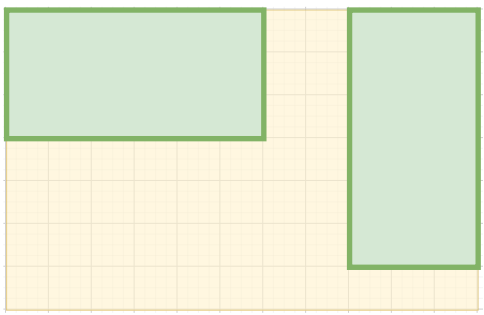


(d) $r_2 = -(1 \cdot 3 \cdot 3 + 3 \cdot 2 \cdot 3)$

Figure 4.2: Simple example for penalty assigned by reward component 2 (cases 1-2) in the case of a bin with base dimensions 7×11 and all equal items with dimensions $3 \times 3 \times 3$.



(a) $r_2 = -(3 \cdot 1 \cdot 3)$



(b) $r_2 = -(8 \cdot 3 \cdot 3 + 3 \cdot 1 \cdot 3)$

Figure 4.3: Simple example for penalty assigned by reward component 2 (cases 3-4) in the case of a bin with base dimensions 7×11 and all equal items with dimensions $6 \times 3 \times 3$.

```
"order": {
    "items": {
      "1": {
        "article": "art-id1",
        "id": "id1",
        "product-group": "pg-1",
        "length/mm": 300,
        "width/mm": 200,
        "height/mm": 100,
        "weight/kg": 3.14,
      },
      ...
    },
    "properties": {
      "id": "0123456",
      "order.nr": "01234567",
      "type": "chilled frozen grocery",
      "target": "euro-pallet"
    }
}
```

Figure 4.4: Representation of an Order in [56] dataset

5 | Methodology

To efficiently solve the 3D Bin Packing Problem, this research decomposes the packing process by adopting the layer-building pattern. To mathematically formalize this approach, the CMDP formulation is extended into the *Options framework*, which allows for expanding the atomic placement of a single box into a temporally macro-action responsible for the construction of an entire layer of items over multiple time steps.

This multi-level decision-making process is addressed by adopting Hierarchical Reinforcement Learning (HRL). Furthermore, targeted heuristics are applied at various levels of the architecture to facilitate the learning agent's task by streamlining the state-action space and enforcing physical constraints.

5.1. Hierarchical Framework Definition

To manage the combinatorial complexity of the 3D-BPP, the decision-making process is structured using Hierarchical Reinforcement Learning (HRL). This approach decomposes the control architecture into a two-tiered system: a high-level *Manager* that selects broad strategic goals at a coarse temporal resolution, and a low-level *Worker* that translates these directives into precise, primitive spatial placements.
To mathematically formalize this hierarchical decomposition, the *Options framework* provides the formal temporal abstraction required to encapsulate sequences of primitive actions into macro-actions.

This section first defines the specific Options framework adopted for the layer-building strategy, and subsequently details the operational mechanics of both the Manager and the Worker policies.

5.1.1. The Option Framework

The option space $\Omega(S_t)$ represents the full repertoire of strategic choices available to the hierarchical agent at decision time t. It is defined as the mathematical union of a primitive

base action and a family of parameterized, temporally extended macro-actions:

$$\Omega(S_t) = \{\omega_{free}\} \cup \{\omega_{layer}(h) \mid h \in H_{cand}(S_t)\}. \tag{5.1}$$

Where H_{cand} is the set of unique heights with tolerance extracted from the items of the inventory s_t^u, mathematically:

$$H_{cand} = \{h_k = \nu_{k,o}(3) \mid \nu_{k,o} \in s_t^u \wedge h_k < h_i^{cand} - 2\delta_h \quad \forall h_i^{cand} \in H_{cand}\}.$$

To establish a unified notation, the primitive base option is integrated into the same parametric family as the layer-building macro-actions. Let $\omega(h)$ denote a generalized option parameterized by h. By simply evaluating the value of the parameter h, the system automatically infers the operational mode of the agent: $h = 0$ triggers the "Free Mode", whereas $h > 0$ specifies the target height for the "Layer Mode".

Consequently, the overarching option space can be compactly redefined as:

$$\Omega(S_t) = \{\omega(h) \mid h \in \{0\} \cup H_{cand}(S_t)\}. \tag{5.2}$$

Primitive Action ($\omega(0)$). Triggered when $h = 0$, this base option corresponds to the standard, atomic step of the underlying CMDP. It involves the selection of a single item from s_t^u, followed by its placement at any geometrically and structurally feasible coordinate within the active bin. Because it represents a discrete, single-step decision, its temporal duration $\bar{t}$ corresponds exactly to one environment step ($\bar{t} = 1$).

Layer Option ($\omega(h)$ for $h > 0$). The layer-building strategy constitutes a set of temporally extended actions differentiated by a strictly positive vertical height parameter h. The inventory s_t^u typically comprises multiple subsets of items sharing the exact same height. Each of these uniform-height subsets is eligible to form a continuous horizontal layer.
Consequently, to initiate a layer option, the agent must specify which subset to utilize by passing the target layer height h as a parameter.

Following the standard Options framework, each parameterized option $\omega_{layer}(h)$ is formally defined by the tuple $\langle \mathcal{I}_h, \pi_h, \beta_h \rangle$:

- **Initiation Set ($\mathcal{I}_h$):** In the Options framework, $\mathcal{I}_h \subseteq \mathcal{S}$ represents the subset of environment states from which the option can be executed. Let $s_h^u = \{\nu_k \in s_t^u \mid$

$\delta_h \geq |h - h_k| \ \wedge \ q_k > 0\}$ denote the subset of currently unpacked physical boxes possessing height h with tolerance δ_h. The option $\omega_{layer}(h)$ can be triggered if and only if two state conditions are strictly met:

1. the current placement surface inside the bin is evaluated as structurally pseudo-planar (Algorithm A.3),
2. It exists at least a non empty inventory subset s_t^u, mathematically: $\exists h \in H_{cand} \mid s_h^u \neq \emptyset$).

- **Intra-option Policy (π_h):** This is the internal policy that sequentially selects and places an oriented item from s_t^u. This policy is *shared* across all options; that is, the agent learns a single, unified policy to drive the intra-option behavior.
  During the execution of $\omega_{layer}(h)$, this shared policy is constrained via conditional masking. First, the item selection is restricted exclusively to the subset s_h^u. Second, the vertical placement coordinate is rigidly locked to the elevation of the current pseudo-planar surface (z_{curr}). This spatial restriction is mathematically enforced by converting the bin state action mask, $\mathcal{M}_t^c$, into an option-specific container mask, $\mathcal{M}_h^c$. Specifically, $\mathcal{M}_h^c$ actively filters the 2D action space, marking as admissible only the grid coordinates where the resulting placement elevation exactly matches z_{curr}.
- **Termination Condition (β_h):** The execution of the temporally extended option concludes (i.e., the termination probability $\beta_h(S_t) = 1$) when at least one of the following environment conditions is met:
  1. The specific subset of available boxes s_h^u is empty.
  2. The current layer becomes geometrically saturated, meaning no valid physical placement exists on the z_{curr} plane for any remaining item within s_h^u.

5.1.2. High-Level Policy (Heuristic Manager)

To mitigate the parameter overhead and sample inefficiency associated with training multi-level neural architectures, the Manager policy (π_m) is designed as a heuristic-driven module. Operating at a coarse temporal resolution, its primary function is to evaluate the global environment state S_t, select an optimal placement strategy $\omega \in \Omega(S_t)$, and dictate the corresponding execution parameters to the lower-level Worker (see [Fig. 5.1]).

At decision time, the Manager first evaluates the initiation condition for the layer-option, verifying whether the current placement surface in the bin is pseudo-planar.

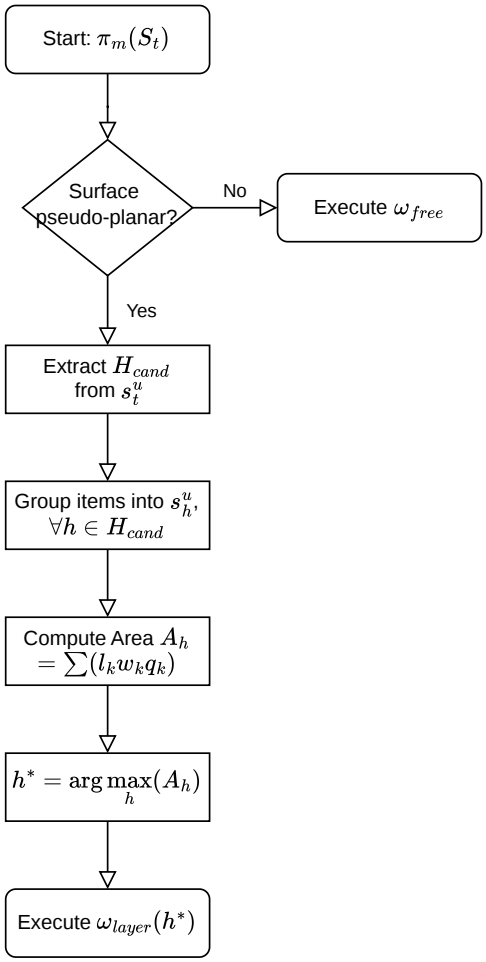


Figure 5.1: Flowchart of Manager Policy.

If this condition is not met, the environment is deemed unsuitable for layer construction, and the Manager defaults to the primitive action, executing $\omega(0)$.
If the pseudo-planar condition is satisfied, the Manager defaults to the layer-option. The parameter h is selected through the following steps:

1. **Candidate Extraction:** The Manager scans the inventory state s_t^u and extracts all unique item heights currently available to define the candidate parameter set H_{cand}.
2. **Subset Grouping:** For each candidate height $h \in H_{cand}$, the inventory is logically partitioned to create a specific subset s_h^u, containing all and only the virtual oriented items where the height $h_i = h$.
3. **Coverage Computation:** To estimate the geometric utility of each candidate layer, the Manager computes the total potential horizontal coverage area A_h. This is derived by summing the base areas of all items within the corresponding subset,

directly accounting for their available quantities q_k:

$$A_h = \sum_{\nu_k \in s_h^u} (l_k \times w_k \times q_k). \tag{5.3}$$

4. **Parameter Selection:** To greedily maximize volumetric packing density, the Manager selects the optimal height parameter h^* that yields the greatest potential base area:
$$h^* = \arg \max_{h \in H_{cand}} (A_h). \tag{5.4}$$

5. **Option Initiation:** Finally, the Manager triggers the option layer parameterized by h^* ($\omega_{layer}(h^*)$).

5.1.3. Low-Level Policy (Shared Worker)

As previously established, the hierarchical architecture employs a **Single Shared Intra-Option Policy** $\pi_w(a \mid S_t, \omega)$, parameterized by θ. The rationale behind a unified worker policy is that at the single-step level, the fundamental spatial decision-making behaviors remain identical regardless of the high-level strategy being pursued.

The behavioral differentiation between the options "Free Mode" and "Layer Mode" is achieved exclusively through conditional state observation and **Conditional Action Masking**, without requiring any structural changes to the policy structure.

1. Execution under the Primitive Option ($\omega(0)$): In this mode, the agent observes the full inventory state $s^u_{h=0} = s^u_t$. The applied mask $\mathcal{M}^c_{h=0}$ corresponds directly to the standard container action mask $\mathcal{M}^c_t$.

$$\mathcal{M}^c_{h=0}(i,x,y) = \begin{cases} 1 & \text{if } i \in s^u_t \wedge \mathcal{M}^c_t(i,x,y) = 1; \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

2. Execution under the Layer Option ($\omega(h)$): In this mode, the agent observes the reduced inventory state s^u_h. The applied mask set $\mathcal{M}^c_h$ zeros out all the masks associated with non-eligible items (i.e. any item outside of s^u_h). In the non-zeroed masks, any spatial placement coordinate (x, y) that does not align with the elevation of the targeted

pseudo-planar surface z_{curr} is invalidated:

$$\mathcal{M}_h^c(i,x,y) = \begin{cases} 1 & \text{if } i \in s_h^u \wedge \mathcal{M}_t^c(i,x,y) = 1 \wedge z_{pos}(x,y) = z_{curr}; \\ 0 & \text{otherwise.} \end{cases} \tag{5.6}$$

Through this dynamic state-masking mechanism, the fundamental architecture of the Worker policy remains strictly invariant regardless of the selected option $\omega(h)$. However, exactly computing the optimal policy within such a high-dimensional, combinatorial state-action space is computationally intractable. Consequently, the policy function in this research is approximated using Deep Neural Networks (DNNs).

Furthermore, the Worker policy is structurally decomposed into two interdependent sub-policies, both parameterized by neural networks:

1. **Selection Policy:** This module is tasked with selecting a specific virtual-oriented item from the conditionally filtered inventory state s_h^u.
2. **Placement Policy:** Conditioned on the chosen item, this module determines the optimal spatial coordinate on the bin's heightmap to position the item's FLB corner. The action space of this placement policy is strictly constrained by the option-specific action mask $\mathcal{M}_h^c$.

5.2. Heuristic Construction of Action Mask

Before delving into the specific layer architecture and training methodology of these neural networks, the following section will detail the step-by-step mathematical construction of the spatial action mask $\mathcal{M}_h^c$.

Reduced Action Space via CAR

A standard heuristic approach to constrain the spatial action space relies on the extraction of **Corner Points (CPs)**. The set of Corner Points, denoted as $\mathscr{S}_{CP}$, comprises the specific coordinates generated by the orthogonal intersections between the boundaries of currently packed boxes and the container walls. Confining the permissible placement coordinates strictly to this $\mathscr{S}_{CP}$ set drastically reduces the computational complexity.

However, restricting the placement coordinates exclusively to exact CP coordinates proves overly conservative, precluding the DRL policy from exploring local spatial adjustments.

To resolve this limitation, the concept of **Corner-Anchored Regions (CARs)** is

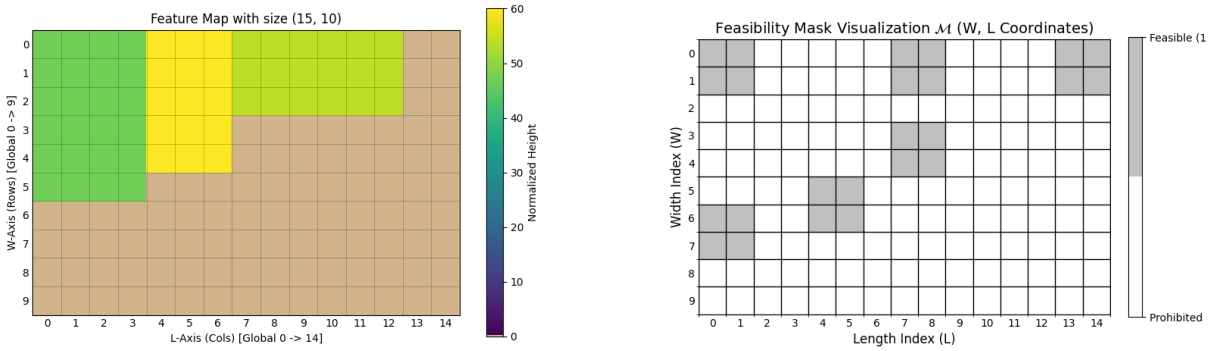
proposed. The discrete $\mathscr{S}_{CP}$ set is spatially expanded by utilizing a theoretical "fictitious box" ι_{min}, characterized by the minimum planar dimensions present in the current order (i.e., a base footprint of $w_{min} \times w_{min}$).
For each identified corner point $cp_k = (x_k, y_k) \in CP$, the corresponding CAR is defined as the rectangular planar area that would be covered by the base of ι_{min} if its FLB corner were anchored exactly at cp_k. Formally, this region is defined as:

$$\mathrm{CAR}_k = \left\{(x, y) \in \mathbb{Z}^2 \mid x_k \leq x < x_k + w_{min} \wedge y_k \leq y < y_k + w_{min}\right\}. \tag{5.7}$$

The global preliminary action mask $\mathcal{M}_{\mathrm{CAR}}$ is then generated by the boolean union of all individual regions:

$$\mathcal{M}_{\mathrm{CAR}}(x, y) = \begin{cases} 1 & \text{if } (x, y) \in \bigcup_k \mathrm{CAR}_k; \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

A visual comparison between the standard heightmap and the resulting CAR action mask is illustrated in Fig. 5.2.



(a) Heightmap with 3 placed items.

(b) CARs action mask assuming a $w_{min} \times w_{min}$ fictitious item.

Figure 5.2: Comparison between the active heightmap (a) and the corresponding preliminary CAR action mask (b) for a 15×10 container grid.

Containment Constraint Check. Even if a placement coordinate (x, y) is validated by the preliminary $\mathcal{M}_{\mathrm{CAR}}$ mask, it must strictly adhere to the physical boundaries of the container. Thus, a geometric containment constraint is enforced.

This constraint is computed through efficient vectorized array operations. For each of the $2N_k$ oriented virtual clusters $\nu_{k,o}$ in the inventory, a distinct boolean containment mask $\mathcal{M}_{\mathrm{cont}}^{(k,o)} \in \{0, 1\}^{L \times W}$ is generated. Each cell (x, y) is marked as valid (boolean 1) if and

only if the addition of the item's planar dimensions $l_k(o)$ and $w_k(o)$ does not exceed the maximum discrete length L and width W of the container grid.

Mathematically, the containment mask is defined as:

$$\mathcal{M}_{\text{cont}}^{(k,o)}(x,y) = \begin{cases} 1 & \text{if } x + l_k(o) \leq L \quad \wedge \quad y + w_k(o) \leq W; \\ 0 & \text{otherwise.} \end{cases} \tag{5.9}$$

By applying this boolean filter, all physically out-of-bounds coordinates are systematically discarded, ensuring that the agent's action space is strictly confined to fully contained placements.

Fast Stability Check. While the $\mathcal{M}_{\text{CAR}}$ mask significantly reduces the action space, it may still encompass coordinates that result in physically unstable placements. Performing a rigorous physical stability validation based on convex hull analysis (as detailed in [Par. 4.2.2]) for every single cell within $\mathcal{M}_{\text{CAR}}$ would be computationally prohibitive during the training phase.

Instead, a vectorized *fast stability check* is introduced, leveraging the pre-computed plane features extracted from the spatial state s_t^c and the "feasibility" mask computed in the rigorous stability assessment [Par. 4.2.2]. Assuming a uniform mass distribution for all items, a box is considered statically stable if its Center of Mass (CoM) is adequately supported by the underlying pseudo-planar surface.

For each of the $2N_k$ oriented virtual clusters $\nu_{k,o}$ in s_t^u, a distinct boolean stability mask $\mathcal{M}_{\text{stab}}^{(k,o)} \in \{0,1\}^{L \times W}$ is computed. For any candidate Loading Position (x,y), the directional plane features $\{d^N, d^S, d^E, d^W\}$ provide the contiguous extension of the supporting plateau. Stability is heuristically guaranteed if three spatial criteria are met:

1. The supporting plane extends from the footprint boundaries beyond the Center of Mass (CoM) along both spatial axes,
2. The CoM itself rests at the exact same pseudo-planar height level as the FLB corner,
3. The CoM lies within a region marked as viable by the global feasibility mask $\mathcal{F}^m$.

Mathematically, a cell (x,y) is marked as valid (boolean 1) in the mask $\mathcal{M}_{\text{stab}}^{(k,o)}$ if the following conditions are simultaneously satisfied:

$$\begin{cases} d^E_{x,y} > \dfrac{l_k(o)}{2} \quad \vee \quad d^W_{x+l_k(o),y} > \dfrac{l_k(o)}{2}; \\ d^N_{x,y} > \dfrac{w_k(o)}{2} \quad \vee \quad d^S_{x,y+w_k(o)} > \dfrac{w_k(o)}{2}; \\ \mathcal{H}^m_{plane}(x,y) = \mathcal{H}^m_{plane}\left(x + \dfrac{l_k(o)}{2}, y + \dfrac{w_k(o)}{2}\right); \\ \mathcal{F}^m\left(x + \dfrac{l_k(o)}{2}, y + \dfrac{w_k(o)}{2}\right) = 1. \end{cases} \tag{5.10}$$

The resulting set of $2N_k$ boolean maps constitutes the global stability mask $\mathcal{M}_{\text{stab}}$, efficiently filtering out structurally unsound actions before they are evaluated by the policy.
The vectorized procedural implementation is detailed in Algorithm A.6.

Edge-Alignment Constraint. To further improve packing efficiency and minimize dead space, an additional hard restriction is introduced: the *edge-alignment constraint*. This heuristic enforces a dense layer-building strategy originating from the container's front-left corner. Specifically, if a CAR is located in proximity to the container's South ($x = 0$) or West ($y = 0$) boundaries, only candidate positions strictly aligned with these edges are retained. Unaligned cells within those edge-adjacent CARs are discarded to prevent the creation of unfilled micro-gaps near the walls.

To maintain structural consistency during the final intersection of the action space, a distinct boolean edge-alignment mask $\mathcal{M}^{(k,o)}_{\text{edge}} \in \{0,1\}^{L \times W}$ is instantiated for each of the $2N_k$ oriented virtual clusters in s^u_t. For any coordinate (x, y) belonging to the preliminary CAR space, the edge mask for the n-th item evaluates to:

$$\mathcal{M}^{(k,o)}_{\text{edge}}(x,y) = \begin{cases} 1 & \text{if } (x = 0 \vee x \geq w_{min}) \quad \wedge \quad (y = 0 \vee y \geq w_{min}); \\ 0 & \text{otherwise.} \end{cases} \tag{5.11}$$

Final Container Action Mask ($\mathcal{M}^c_t$). The ultimate container state component, $\mathcal{M}^c_t$, is structured as a set of $2N_k$ independent boolean maps, one for each cluster $\nu_{k,o}$ in the inventory s^u_t.

For each specific physical box type $k \in \{1, \ldots, N_k\}$ and its valid orientation $o \in \{0, 1\}$, the final valid placement mask $\mathcal{M}^{c,(k,o)}_t$ is computed as the logical intersection (boolean AND) of the heuristic masks ($\mathcal{M}_{\text{CAR}}$ and $\mathcal{M}^{(k,o)}_{\text{edge}}$) and the item-specific physical constraint

masks ($\mathcal{M}_{\text{stab}}^{(k,o)}$ and $\mathcal{M}_{\text{cont}}^{(k,o)}$):

$$\mathcal{M}_t^{c,(k,o)} = \mathcal{M}_{\text{CAR}} \cap \mathcal{M}_{\text{edge}}^{(k,o)} \cap \mathcal{M}_{\text{stab}}^{(k,o)} \cap \mathcal{M}_{\text{cont}}^{(k,o)} \tag{5.12}$$

An illustrative example of the resulting combined action map is provided in Fig. 5.3.

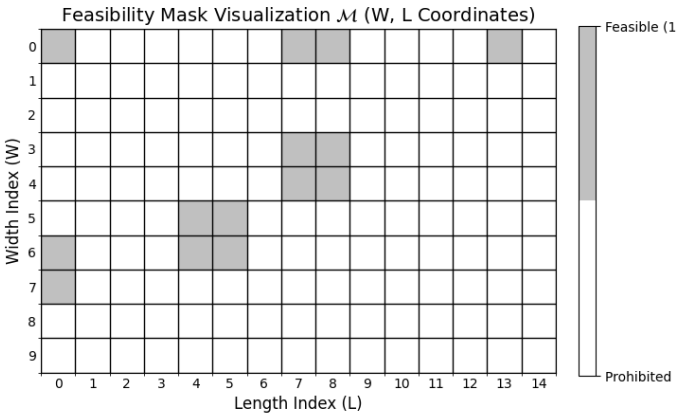


Figure 5.3: Final action map corresponding to the heightmap in Fig. 5.2a, considering a $w_{min} \times w_{min}$ fictitious item footprint.

5.3. Worker Policy Architecture

As established in the hierarchical framework, the unified Worker policy π_w is responsible for executing the single steps of the option. This policy is structurally decomposed into two interdependent sub-policies: a selection policy π_{sel} and a placement policy π_{pla}.

The selection policy is tasked with choosing a specific item alongside its optimal orientation from the available inventory. Subsequently, the placement policy determines the spatial coordinate for the item's FLB corner, strictly conditioned on the item chosen by π_{sel}. The total Worker policy is mathematically defined as the joint probability of these two sequential decisions:

$$\pi_w(a \mid S_t, \omega) = \pi_{sel}(i \mid S_t, \omega) \cdot \pi_{pla}(x, y \mid S_t, i, \omega). \tag{5.13}$$

Both sub-policies are approximated using Deep Neural Networks featuring specialized layers designed to facilitate spatial reasoning and representation learning. The high-level pipeline operates as follows: first, the inventory state s_h^u and the spatial feature map s_t^c are processed by dedicated state encoders. s_h^u is processed via a Transformer Encoder, while s_t^c

utilizes a CNN followed by a Transformer Encoder. These two encoded representations are then fed into a Pointer Network to model the selection policy π_{sel}. Finally, the placement policy π_{pla} uses a Transformer Decoder to cross-attend the encoding of the newly selected item with the encoded heightmap to output a probability distribution over the spatial grid.

A schematic overview of the entire network architecture is provided in [Fig. 5.4], and a detailed explanation of each module is provided in the following subsections.

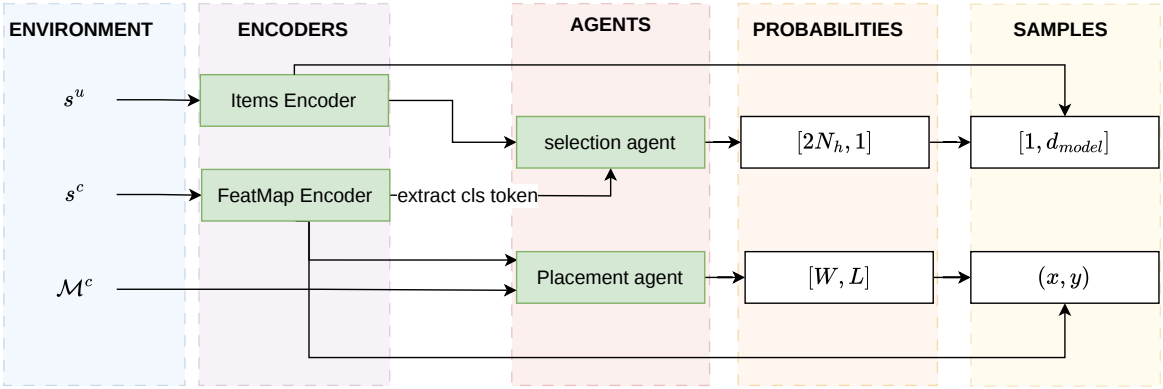


Figure 5.4: Scheme of the Worker Policy Architecture.

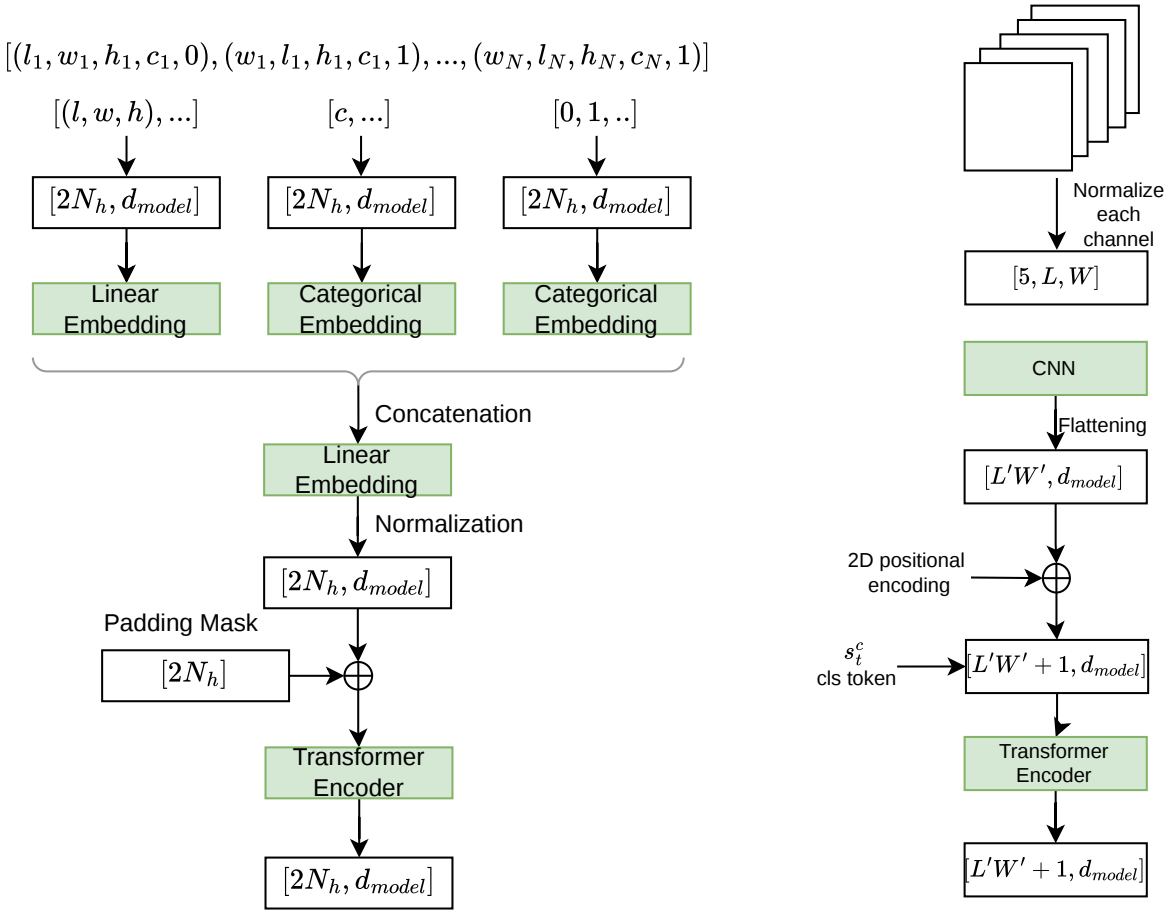
5.3.1. Item Encoder

s_h^u is presented to the network as a state matrix of shape $2N \times 5$, representing the $2N_h$ virtual oriented clusters. This matrix is structurally decomposed into three distinct feature arrays: physical dimensions, available count, and orientation flag.

To ensure stable gradient flow and effective representation, these arrays are embedded separately. The dimensions array (l, w, h) is first normalized against the maximum container boundaries (L, W, H) to yield scale-invariant inputs $(l/L, w/W, h/H)$, which are then projected through a standard Linear layer. The discrete orientation and count arrays are mapped to dense continuous vectors via categorical embedding layers.

Furthermore, dynamic "feasibility features" are explicitly computed to provide the network with immediate spatial awareness. For every oriented item, the environment calculates the total number of geometrically and structurally valid placement cells ($N_{feasible}$) derived from the option-specific action mask $\mathcal{M}_h^c$. Two distinct metrics are computed from this count: the standard linear fraction relative to the total grid area, $N_{feasible}/(W \times L)$, and a logarithmic fraction, $\log(1 + N_{feasible})/\log(W \times L)$.
These two metrics provide complementary signals to the learning agent. The linear fraction offers a direct, absolute measure of spatial availability across the bin. In contrast, the

logarithmic fraction amplifies the network's sensitivity at the lower bounds, providing a steeper gradient when the feasible count for a specific item approaches zero.
These extra features are stacked, linearly embedded, and added to the physical dimension embeddings. Finally, the dimensional, count, and orientation embeddings are concatenated and passed through a multi-head Transformer Encoder block, yielding a context-aware representation of each available item within the inventory.



(a) Scheme of the items encoder module.

(b) Scheme of the spatial heightmap encoder.

Figure 5.5: Architectural details of the state encoding modules.

5.3.2. Feature Map Encoder

The state of the container's surface is processed using a hybrid CNN-Transformer architecture, drawing inspiration from the methodology proposed in [88]. Before feature extraction, the multi-channel 2D spatial feature map s_t^c undergoes a normalization process to ensure scale invariance and stabilize the network's learning dynamics.

Specifically, the heightmap channel is normalized by the bin's maximum permissible height H. The supplementary spatial planar features, along the four cardinal directions (North, South, East, and West), are respectively normalized by the container's physical base dimensions L and W.

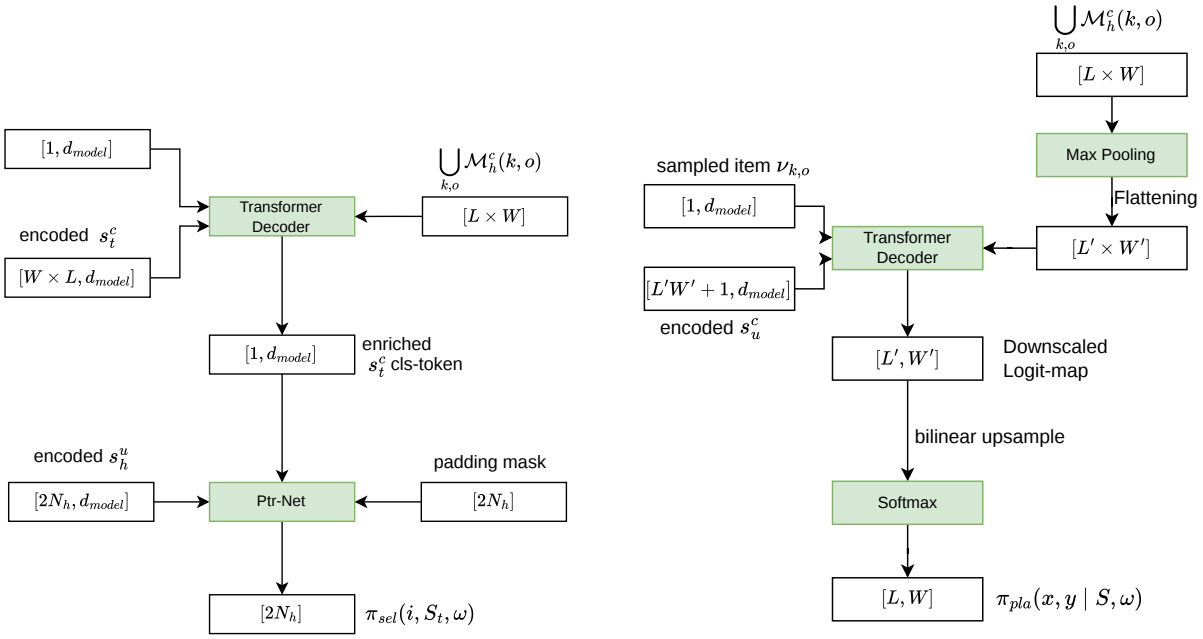
The normalized bin's state representation s_t^c is then passed through a CNN consisting of convolutional layers followed by average pooling operations. This downscaled spatial map is flattened into a 1D sequence and enriched with a 2D positional encoding to allow the permutation-invariant Transformer to retain the absolute and relative 2D coordinate geometry of the individual grid cells.

Finally, a learnable Classification Token (`[CLS]`) is appended to the beginning of the flattened sequence. The entire sequence is then fed into a Transformer Encoder, leveraging its self-attention mechanism to model long-range global dependencies and contextual interactions across the entire spatial grid. The output corresponding to the `[CLS]` token serves as the aggregated, global representation of the container's current state.

5.3.3. Selection Agent

The selection agent, responsible for the sub-policy π_{sel}, is designed to identify the optimal next item to pack alongside its orientation. It leverages a two-stage attention-based mechanism that fuses spatial context with discrete item selection (see [Fig. 5.6a]).

In the first stage, the network performs a masked cross-attention operation to enrich the global spatial representation. The Query is the global `[CLS]` token derived from the feature map encoder, while the Keys and Values are the sequence of the flattened, encoded spatial feature map. The mask is generated by taking the logical union of the option-specific action masks $\mathcal{M}_h^c$. To align the dimensions of this mask with the downscaled feature map produced by the CNN, the union mask undergoes an identical pooling operation. Specifically, Max Pooling is applied: this guarantees that if at least one raw spatial cell within a pooling window is marked as feasible, the corresponding downscaled region retains its feasibility status. This targeted cross-attention explicitly enriches the global `[CLS]` token with localized feasibility awareness.
The architectural decision to route this information through the `[CLS]` token rather than the full spatial map is dictated by efficiency. Computing direct cross-attention between all $2N_h$ item embeddings and every individual cell of the spatial grid would incur a prohibitively high computational cost. By condensing the spatial and feasibility information into a single representative token, the model successfully retains the global context of the container's state while drastically reducing computational complexity.



(a) Scheme of Selection agent module. (b) Scheme of Placement agent module.

Figure 5.6: Architectural details of the sub-policy modules.

In the second stage, the selection agent employs a Pointer Network architecture to evaluate the available inventory. It computes attention scores using the $2N_h$ encoded items as Queries against the feasibility-enriched `[CLS]` token. The output of this stage is passed through a softmax activation function, yielding a valid probability distribution over all possible item indices, represented as a vector of length $2N_h$.

Finally, a discrete index $idx \in [0, 2N_h - 1]$ is extracted from this distribution. To balance exploration and exploitation, the index is sampled from the probability distribution during the training phase, whereas it is deterministically selected via an *argmax* operation during inference. This single scalar index encodes both the physical item selection and its 2D spatial orientation:

$$k = \left\lfloor \frac{idx}{2} \right\rfloor;$$
$$o = idx \bmod 2.$$

5.3.4. Placement Agent

Following the item selection, the placement agent is tasked with executing the sub-policy $\pi_{pla}(x, y \mid S_t, i, \omega)$. Its objective is to determine the optimal, spatial coordinate (x, y) on the bin's heightmap to position the FLB corner of the chosen item.

The architecture employs a Transformer Decoder module where the flattened sequence of the downscaled heightmap tokens acts as the Queries, while the, encoding of the explicitly selected item (and its chosen orientation) serves as the Key-Value pair.
To enforce the reduction of the action space, the decoder performs masked cross-attention. The mask $\mathcal{M}_h^c(k, o)$ is extracted from the option-specific action masks $\mathcal{M}_h^c$ using the item-orientation index (idx) selected by π_{sel}. Because the spatial Queries operate at the downscaled resolution produced by the CNN (defined as $W' \times L'$), the mask is dimensionally aligned by applying the same spatial downsampling procedure used for the raw feature map, utilizing Max Pooling instead of Average Pooling. Consequently, non-feasible spatial regions are completely masked out prior to the final coordinate selection, forcing the agent to act in the reduced action space.

Following the cross-attention blocks, the decoder's output sequence is projected into a single scalar per spatial token via a linear layer, forming a coarse placement logits map of size $W' \times L'$. To obtain a probability distribution aligned with the bin's original heightmap resolution ($W \times L$), this coarse map is upsampled using bilinear interpolation. This upsampling technique is chosen among the others since it projects a smooth, continuous gradient field. This mathematical flexibility allows the network to gracefully distribute probability mass and refine its placement decisions at the original high resolution.
The upscaled logits map is passed through a masked Softmax layer. This layer zeroes out any remaining unfeasible coordinates at the native resolution (using the raw $\mathcal{M}_h^c(k, o)$ mask) and computes the final 2D probability distribution over all valid cells. To effectively balance exploration and exploitation, the final placement coordinate (x, y) is sampled from this distribution during the training phase, whereas the agent deterministically selects the coordinate corresponding to the maximum probability ($argmax$) during inference.

5.4. Critic Function Approximator

In the context of the adopted Actor-Critic framework, the Critic network is responsible for estimating the state value function $V(S_t)$. This function evaluates the expected cumulative reward from the current global state S_t, serving as a crucial baseline to reduce variance during the optimization of the Worker policy. To provide an accurate and stable value estimation, the Critic is also approximated using a dedicated DNN.

To synthesize a comprehensive understanding of the complex 3D-BPP environment, the input to the Critic network is constructed by aggregating three distinct streams of information:

1. **Global Spatial Context:** The `[CLS]` token extracted from the Heightmap Encoder,

which encapsulates the rich, downscaled topographic representation of the container's current state.

2. **Global Inventory Context:** The `[CLS]` token extracted from the Item Encoder, summarizing the geometric and quantitative distribution of the remaining items in s_t^u.

3. **Macroscopic State Scalars:** A highly condensed, six-dimensional feature vector containing explicit statistical and physical metrics that summarize the overall packing progression. These six scalars are:

   - The logarithm of the total count of remaining items.
   - The logarithm of the aggregate volume of the remaining inventory.
   - The mathematical mean of the container's heightmap (μ_{hmap}), representing the average filling level.
   - The maximum value of the heightmap ($\max_{hmap}$), monitoring the proximity to the container's physical vertical bound H.
   - The standard deviation of the heightmap (σ_{hmap}), acting as a proxy for surface roughness and topographic irregularity.
   - A boolean indicator ($I_{planar} \in \{0, 1\}$) signaling the presence of a valid pseudo-planar surface. This directly aligns the Critic's spatial awareness with the initiation condition evaluated by the high-level Manager.

These three components are concatenated into a single, unified state feature vector processed through a Multi-Layer Perceptron (MLP). The hidden layers of the MLP are ReLU activation functions to model non-linear interactions between the global embeddings and the heuristic scalars. Finally, the network terminates with a linear output layer that predicts a single continuous scalar, representing the approximated state value $V(S_t)$.

5.5. Implementation Details

This section details the practical implementation strategies employed to train the DRL agent. It outlines the dynamic reward normalization technique used to stabilize the learning and describes the methodology for constructing the extensive training dataset required to properly optimize the policy.

5.5.1. Reward Normalization and Clipping

The magnitude of the immediate reward $R^{(k)}$ at any given step k can exhibit significant variance. This fluctuation primarily arises from the diverse geometric properties of the items: placing a bigger item yields a disproportionately larger reward compared to fitting a smaller object. Furthermore, the inherent dynamics of the packing episode can lead to highly variable returns depending on the current density and the remaining available space.

To mitigate the effects of this high variance on the neural network's gradient updates and to stabilize the learning process, a dynamic reward normalization strategy is implemented:

$$R' = \frac{R^{(k)} - \mu_{batch}}{\sigma_{batch}}. \tag{5.14}$$

This normalization centers the advantage estimates and ensures a consistent scale across different policy updates.

Following the normalization phase, reward R' is clipped within a symmetric range $[-c, c]$ (empirically set to $c = 10$). This clipping acts as a robust safeguard against outlier reward signals that could otherwise induce exploding gradients, cause drastic unrecoverable shifts in the policy distribution, and destabilize the Critic's value function estimation.

5.5.2. Synthetic Data Generation Pipeline

Since DRL requires massive amounts of interaction data to converge, the static dataset provided by the literature is insufficient for training. Consequently, a generative procedure was developed to produce an infinite stream of realistic orders. This procedure is based on the statistical analysis of a corpus of 8432 real-world logistic orders.

Statistical Analysis

Three primary probability distributions were extracted from the source data to model the composition of an order:

1. **Order Cardinality Distribution (D_K):** Modeling the number of distinct articles (SKUs) present in a single order. Let K be the random variable representing this count.
2. **Article Probability Distribution (D_P):** Modeling the marginal probability $P(a_i)$ that a specific article a_i appears in an order, for all a_i in the universe of articles $\mathcal{A}$.

3. **Item Quantity Distribution (D_Q):** Modeling the number of physical items (quantity) associated with a specific article a_i within an order.

In addition to these univariate distributions, the inter-dependencies between items were modeled by computing the **Correlation Matrix C**, where each entry C_{ij} represents the likelihood of article a_j appearing in an order given the presence of article a_i.

Distribution Fitting

To generalize the Order Cardinality (K), theoretical distributions were fitted to the empirical data. A comparative analysis was performed between the Poisson distribution and the Negative Binomial distribution [Fig. 5.7]. Goodness-of-fit tests indicated that the **Negative Binomial distribution** provides a superior approximation of the real-world data, capturing the over-dispersion typical of retail orders. Consequently, the parameters r (number of failures) and p (probability of success) of the Negative Binomial were estimated and stored.

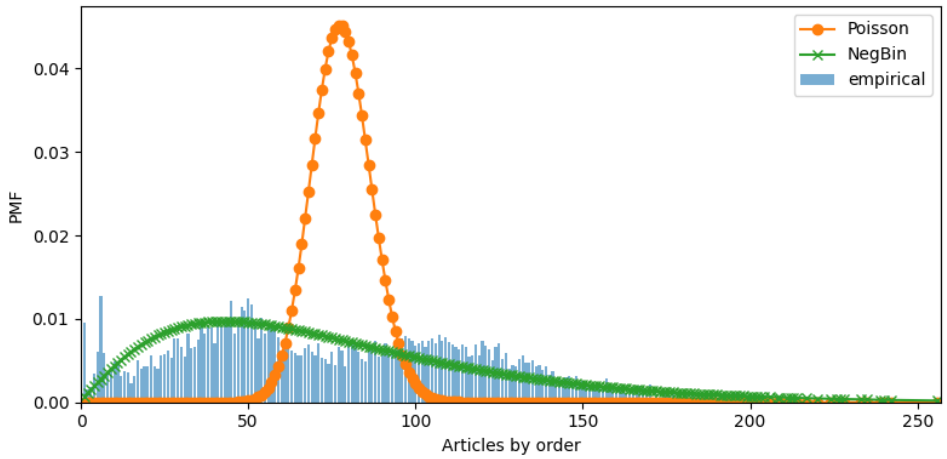


Figure 5.7: Distribution fitting over the Order Cardinality Distribution.

Generative Algorithm

The generation of a new synthetic order $\mathcal{O}_{new}$ follows a three-step stochastic process:

1. Order Size Determination. A maximum limit K_{max} is imposed on the number of distinct articles. The previously estimated Negative Binomial distribution is truncated and renormalized over the interval $[1, K_{max}]$. The number of distinct articles k for the current order is sampled from this truncated distribution:

$$k \sim \text{TruncatedNegBin}(r, p, K_{max}).$$

2. Article Selection. A set of k distinct articles $\{a_1, \ldots, a_k\}$ is selected. The first article is sampled according to the marginal distribution D_P. Subsequent articles are sampled iteratively, where the selection probability is conditioned by the Correlation Matrix $\mathbf{C}$ to reflect realistic co-occurrence patterns (e.g., articles often ordered together).

3. Quantity Assignment. For each selected article a_i, the specific quantity of items q_i is sampled from the conditional quantity distribution $D_Q(a_i)$:

$$q_i \sim D_Q(a_i).$$

This three-step generative process provides a continuous and statistically representative stream of packing orders. By incorporating empirical probability distributions and inter-item correlations, the pipeline replicates the structural variance of real-world logistics. This dynamic data generation prevents the DRL agent from overfitting to a static dataset.

6 | Experimental Results

In this chapter, the performance of the proposed HRL model is evaluated. The assessment is conducted using the Key Performance Indicators (KPIs) established by [56], and the results are compared against selected baseline heuristics and algorithms.

The experimental methodology is structured to manage the hierarchical decomposition of the 3D-BPP by isolating the training of the *layer_ option*. To achieve this, a simplified environment is employed wherein an episode terminates immediately upon the completion of the first layer of boxes. In this configuration, the policy is never exposed to states involving the placement of boxes on top of others, as the focus is restricted entirely to the construction of the initial horizontal packing layer.

Subsequently, the model trained exclusively on this single-layer setup is deployed and evaluated within the context of the full, unrestricted bin environment. This experimental design is deliberately chosen to investigate the generalization capabilities of the learned policy and to explore strategies for training simplification. Since training a model on the complete environment entails significant computational complexity, it is hypothesized that a policy optimized on a simplified sub-task can provide a robust spatial foundation for the full packing problem. Evaluating the model under these conditions serves to demonstrate the versatility of the proposed architecture when subjected to unseen vertical stacking dynamics, validating the effectiveness of a simplified training approach.

6.1. Training Setup

In this section, the experimental training setup is presented. Specifically, the dimensional configuration of the environment and the selected network parameters are detailed.

6.1.1. Environment Setup

The environment is structured to replicate a realistic industrial scenario. Consequently, a standard Euro-pallet is utilized as the target bin, and packing orders are sampled from the dataset described in [Par. 5.5.2]. To bound the problem scope, the number

of distinct articles is limited to 50. However, the total number of boxes to be packed remains unlimited, as each article can be associated with a quantity greater than one. It is noted that the focus is placed entirely on the optimization of a Single Knapsack Problem; therefore, increasing the number of unique articles is deemed unnecessary for this evaluation.

Furthermore, to mitigate computational complexity during training, all spatial dimensions are scaled down by a factor of 50. This resolution adjustment results in a discrete bin representation with a base size of 24×16 units and a maximum allowable height of 32 units.

6.1.2. Network Setup

The configuration of the neural architecture is defined by a uniform embedding dimension of 64, which is maintained across all encoder outputs and Transformer layers. For the spatial feature map encoding, the CNN is structured with three layers, transitioning through channel dimensions of 5 to 5, 5 to 32, and 32 to 64. Furthermore, all Transformer modules within both the encoding and decoding phases are uniformly configured with 3 layers and 4 attention heads.

The complexity of the dual-encoder and dual-decoder structure dictates specific choices for the reinforcement learning hyperparameters. To prevent premature convergence and encourage sufficient exploration of the action space, a relatively high initial entropy coefficient is applied. It is scheduled to decay from 10^{-2} to 10^{-4} as training progresses, stabilizing the policy once a robust behavior is learned. Additionally, given the extensive number of parameters, a conservative learning rate of 3×10^{-5} is maintained, and the PPO clipping parameter is set to 0.14 to restrict policy updates within a safe trust region and avoid catastrophic updates.
The hyperparameters were established after several iterative tuning cycles, and are summarized in Table 6.1.

Network Parameters

d_{model}	n_{heads}	n_{layers}	Params
64	4	3	$\sim 1.5 \times 10^6$

PPO Hyperparameters

LR	n_{steps}	n_{epochs}	Batch	γ	λ	Clip	Entropy	Value
3×10^{-5}	4096	4	512	0.99	0.95	0.14	$10^{-2} \rightarrow 10^{-4}$	0.4

Table 6.1: Optimal hyperparameters identified for the PPO training phase.

6.1.3. Baseline Algorithms

In this section, the baseline algorithms utilized for comparative analysis are briefly outlined.

Genetic Algorithm

To establish a sound baseline, the selected solver, for direct comparison, is one currently deployed by a commercial logistics company to generate real-world 3D packing configurations. A Genetic Algorithm (GA) is utilized to optimize both the sequence of boxes and their spatial placement within the bin. The methodology is structured to prioritize a layer-building approach rather than directly solving the full 3D packing problem. Initially, the formation of distinct layers is attempted; to ensure a consistent comparison with the proposed model, a valid layer is strictly defined by the pseudo-planar surface approximation ([Par. 4.2]). If discrete layers are successfully formulated, they are sequentially packed. Subsequently, for any remaining boxes that cannot be structured into uniform layers, the genetic optimization strategy is applied to determine their placement within the unrestricted 3D bin environment.

Heuristic Algorithm

The second baseline, as defined by [56], is implemented as an online 3D bin packing solver configured with a preview and selection window of five boxes ($p = 5, s = 5$). Although the overall evaluation context is strictly offline, the deployment of this heuristic requires an online-style windowing configuration due to its inherent computational complexity. Specifically, the computational cost of the lookahead search scales as $O(p!)$, rendering a full offline preview of the entire order computationally intractable. To mitigate this restriction and leverage the available offline data, the complete sequence of boxes is pre-ordered by decreasing volume. During the packing process, valid placement coordinates are obtained by identifying the bin's corner points. To determine the final packing action, the spatial arrangement of the preview boxes is simulated and evaluated by sequentially placing boxes into the bin and exploring various packing combinations.

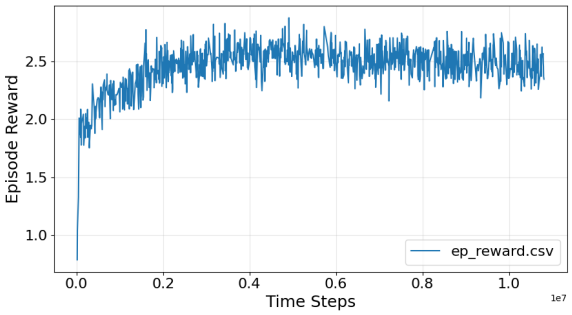
The quality of a single placement action is calculated as a weighted sum of four specific metrics:

1. Estimated support area.
2. Normalized estimated height at the target location.
3. Box orientation.
4. Normalized spatial distance to the origin.

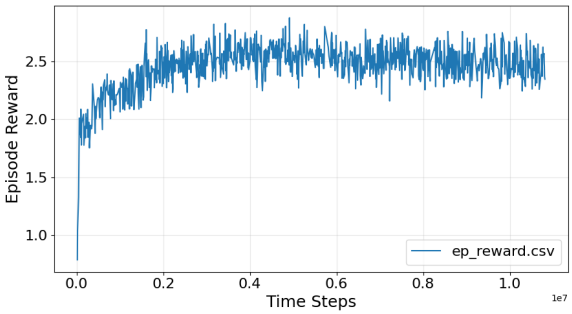
The corresponding weights applied to these criteria are consolidated into the vector $w = [1.3, -2.0, -1.00001, -1.2]$. The immediate action that yields the highest cumulative rating is selected for execution.

6.1.4. Training Convergence and Model Results

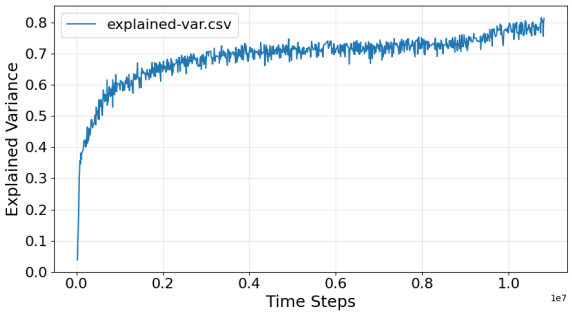
The training phase for the *layer_ option* was conducted utilizing a single random seed. While this configuration precludes the establishment of general statistical boundaries on overall training stability, the extracted learning curves provide empirical evidence that the architecture has the mathematical capacity to converge toward an effective spatial policy (see [Fig. 6.1]).



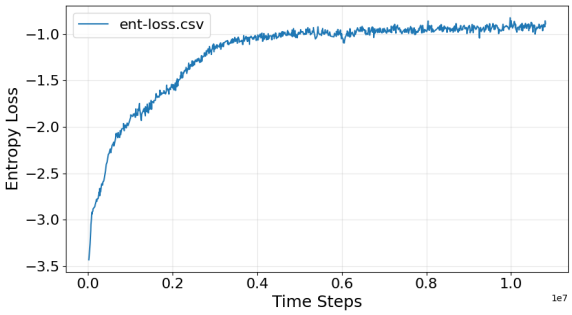
(a) Episodic Length



(b) Episodic Reward



(c) Explained Variance



(d) Entropy Loss

Figure 6.1: Learning curves illustrating the training dynamics of the *layer_ option*. The progression of episodic length and episodic reward (top row) confirms the continuous optimization of the policy. Concurrently, the explained variance and entropy loss (bottom row) demonstrate the stable transition from exploration to exploitation.

The episodic reward and the episodic length steadily increase, demonstrating that a greater number of boxes are successfully packed into the bin and, overall, a greater packing efficiency.
Also, the entropy loss steadily declines, tracking the scheduled reduction of exploration. In

contrast, the explained variance of the value function increases continuously throughout training, reaching approximately 0.8 with minor oscillations. A high explained variance indicates that the critic network accurately approximates the empirical returns. The joint behavior of decreasing entropy and increasing explained variance confirms a robust transition from exploration to exploitation, driven by reliable state evaluations rather than a premature policy collapse.

The inherent variability of the test data directly impacts the final results. Due to the random dimensions and quantity of the sampled boxes, a geometrically compatible subset to form a layer cannot be guaranteed in every order. Consequently, a 100% success rate is structurally improbable for this dataset.

6.1.5. Comparative Analysis

Despite the inherent geometric constraints of the dataset, the proposed HDRL architecture successfully constructed a valid pseudo-planar layer, as strictly defined in [Par. 4.2], in 79.22% of the 1000 evaluated orders. Across these successful instances, the mean area utilization at the base of the bin was 83.67%. To fully contextualize the efficacy of the proposed architecture, these results are benchmarked against established GA and O3DBP-5-5 heuristic baselines. The comparative performance metrics are summarized in Table 6.2.

Metric	**GA**	**HDRL**	**O3DBP-5-5**
Layer Success Rate	**80.76%**	79.22%	17.81%
Area Util. (Mean ± Std)	**88.17% ± 6.36%**	83.67% ± 5.92%	63.31% ± 5.36%
Exec. Time (Mean ± Std)	9.10 s ± 16.78 s	**1.10 s ± 0.19 s**	3.14 s ± 1.10 s

Table 6.2: Performance comparison of the *layer_option* sub-task across 1000 evaluated orders.

Regarding spatial efficiency, the GA establishes the empirical upper bound for this dataset, completing layers in 80.76% of orders with an area utilization of 88.17%. As observed in Table 6.2, the proposed HDRL model demonstrates highly competitive spatial capabilities that closely track this upper bound. Conversely, the heuristic baseline is structurally unsuited for targeted layer-building, fulfilling the pseudo-planar condition in merely 17.81% of the instances.

While the GA provides a marginal advantage in spatial density, its practical applicability is severely hindered by computational latency. The analysis of execution times reveals the operational advantage of the HDRL architecture, with an average execution time of

1.10 seconds per order and a standard deviation of 0.19 seconds. In contrast, the iterative genetic optimization processes of the GA require an average of 9.10 seconds with high variance (±16.78 seconds).
Thus, for a minor reduction of 1 percentage point in the capability to construct a complete layer, the HDRL methodology achieves a substantial computational acceleration, generating a packing configuration approximately 900% faster compared to the GA baseline.

The full distributions of both area utilization and execution times are visualized through the box plots presented in [Fig. 6.2].

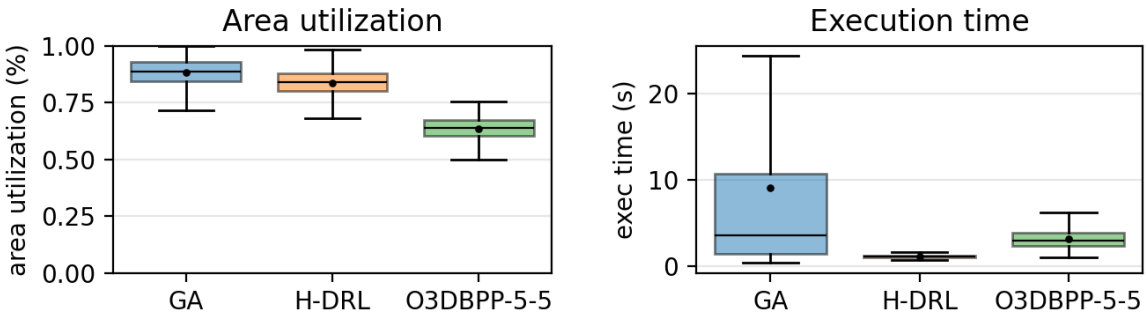


Figure 6.2: Box plots illustrating the distribution of area utilization (left) and execution time (right) across the evaluated algorithms.

6.2. Full Environment Evaluation

To evaluate performance in the unrestricted bin environment, the evaluation method proposed by [56] is used. The assessment relies on a specific set of metrics, grouped into a KPI vector, to quantify the quality of the generated packing configurations. These metrics are defined as follows:

1. **Ratio of unpalletized boxes (n_u/N_{total}):** The fraction of boxes that are not successfully placed into the bin.
2. **Volume utilization (η_{util}):** The ratio of the total volume of packed boxes to the volume of their circumscribed cuboid.
3. **Maximum stacking height (h_n).**
4. **Mean support area ($\overline{A}_{supp}$):** The average proportion of the base area of each packed box that is directly supported by underlying surfaces.
5. **Interlocking ratio (r_{interl}):** The fraction of boxes resting on multiple underlying

boxes or on a single rotated box.

6. **Physical stability ($\mathbb{I}_{stable}$):** A binary indicator derived from a rigid body simulation using the 3D modeling library Blender to verify that the generated pile does not collapse.

Furthermore, an overall benchmark score (Σ_{algo}) is calculated to evaluate the general methodology across multiple orders. It measures the percentage of successfully packed boxes that form stable configurations within strict height limits, and it is defined as:

$$\Sigma_{algo} = \sum \mathbb{I}_{stable} \left(1 - \frac{\tilde{n}_u}{N_{total}}\right)$$

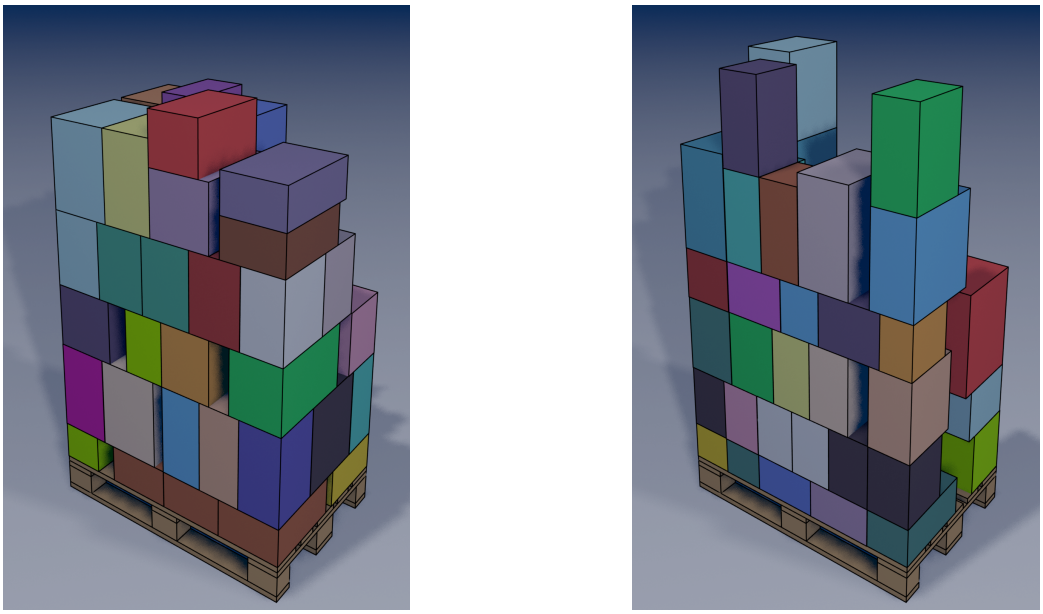
The model is evaluated under two distinct settings: with and without the activation of the rigorous stability check [Par. 4.2.2]. The purpose of this dual evaluation is to demonstrate the efficacy of the fast heuristic stability check, performed using action masking [Par. 5.2], in yielding practical configurations while preserving structural stability in the vast majority of instances. For simplicity of notation, the two settings are named respectively HDRL (Rigorous) and HDRL (Fast-Check).

6.2.1. Generalization and Stability Analysis

For real-world applications, physical stability is a critical requirement. In this regard, the rigid-body simulation verifies that stability is maintained in 99.9% of test cases when the HDRL (Rigorous) configuration is used. It is notable that a high stability rate of 95.3% across 1000 orders is also achieved by the HDRL (Fast-Check) configuration. The significance of this high accuracy is determined by the operational consequences of executing an unstable action. During the training phase, if a chosen placement is deemed unstable, the environment is terminated immediately, and the palletization process is prevented from continuing. During the inference phase, selecting an unstable action requires sequentially testing up to K alternative actions until a stable placement is identified or all options are exhausted. Consequently, the high success rate of the fast check is essential to mitigate inefficiencies arising from unstable actions.

In terms of packing efficiency, both HDRL configurations exhibit adequate performance. It is observed that a higher volume utilization of 66.42% is achieved under the HDRL (Fast Check) setting, compared to 64.21% under the HDRL (Rigorous) configuration. This variation is attributed to the increased packing freedom permitted by the heuristic evaluation. Nonetheless, the rigorous setting achieves a higher final algorithmic score of 0.598, compared to the fast check's 0.576, thanks to the strict stability guarantees, which

effectively eliminate algorithmic penalties for unstable configurations.



(a) Best configuration. (b) Average configuration.

Figure 6.3: 3D visualization of the best (a) and average (b) packing configurations.

For comparative analysis, the Genetic Algorithm (GA) baseline and the O3DBP-5-5 heuristic are employed. The numerical comparisons are detailed in Table 6.3, while the full distributions of both area utilization and execution times are visualized through the box plots presented in [Fig. 6.4].

Model Configuration	Volume Util.	Final Score	Stable Orders	Exec. Time (s)
GA Baseline	**75.73% ± 7.12%**	**0.728**	**100.0%**	717.48 ± 385.78
HDRL (Fast Check)	66.42% ± 8.44%	0.576	95.3%	**2.54 ± 0.54**
HDRL (Rigorous)	64.21% ± 8.55%	0.598	99.9%	**2.54 ± 0.54**
Heuristic O3DBP-5-5	54.58% ± 3.82%	0.002	0.3%	113.90 ± 30.08

Table 6.3: Performance comparison on the full 3D-BPP environment. The HDRL execution time encompasses the complete feedforward generation of the multi-layer packing plan.

The GA baseline represents the upper performance bound for spatial optimization with a volume utilization of 75.73% and a score of 0.728. However, the configurations produced by the HDRL methodology remain highly competitive, particularly given that the underlying policy was never exposed to instances in which boxes are stacked on other boxes during training. The competitiveness of the HDRL method is further underscored by the O3DBP-5-5 heuristic, which is comprehensively outperformed across all evaluated metrics; notably,

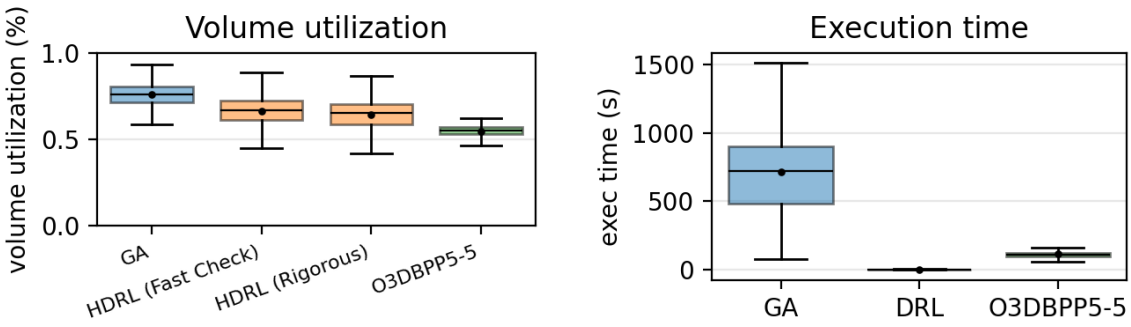


Figure 6.4: Box plots illustrating the distribution of area utilization (left) and execution time (right) across the evaluated algorithms.

the simple heuristic produces stable piles in only 0.3% of orders with an average utilization rate of 54.58%.

Finally, the execution time analysis highlights a fundamental operational advantage. To generate a complete multi-layer packing plan, the HDRL model requires only an average of 2.54 seconds per order. In contrast, the GA necessitates an average of 717.48 seconds (approximately 12 minutes) per order, coupled with a high temporal variance of ±385.78 seconds.

In conclusion, the numerical results indicate practical effectiveness and high generalization capability of the proposed hierarchical decomposition, as despite training only on the single-layer scenario, the HDRL model performs well in the unrestricted full bin-packing scenario.
It is established that this methodology has strong potential for deployment in real-world scenarios, owing to its high physical stability results and low execution times. Specifically, it provides a computational acceleration by a factor of nearly 300 compared to the GA baseline deployed within the commercial logistics enterprise.

7 | Conclusion

To answer the research question, this thesis proposed the employment of Deep Reinforcement Learning and aimed to bridge the gap between theoretical RL and the physical constraints of industrial robotic packing. By exploring the complexities of the 3D Bin Packing Problem, this research demonstrates that relying solely on end-to-end Deep Reinforcement Learning is insufficient for real-world applications, where structural stability and kinematic realizability are critical requirements.
To contextualize this approach, a taxonomy of previous DRL-based 3D-BPP studies was developed, systematically analyzing the literature based on state and action space representations, reward shaping, heuristic integration, and stability assessments. This analysis identified a significant gap: existing methods (for offline 3D-BPP) rarely account for the physical occlusions and reachability limits of real robotic manipulators, nor do they adequately address the structural stability of the loaded items. Additionally, while the integration of heuristics proves effective in reducing problem complexity, current approaches often impose rigid spatial constraints that overly restrict the learning agent's exploration.

To address these limitations, this work introduced Cornered Action Regions (CAR), a novel decomposition of the state and action space. CAR extends traditional Corner Points into continuous surface regions, directly incorporating stability and geometric constraints. By natively modeling real-world occlusions and addressing the stability with a fast and robust heuristic check, CAR prevents the generation of unfeasible coordinates while granting the learning agent a wider degree of freedom compared to standard point-based heuristics (like simple CPs). This research further reduced the overall complexity of the 3D-BPP through a Hierarchical RL architecture focused on layer construction. To formalize this hierarchy, the concept of "pseudo-planarity" was introduced. This concept extends the stability properties of perfectly planar surfaces, such as the bin floor, to newly formed layers that may contain gaps or small steps. Finally, the agent's exploration was guided by a novel reward component designed specifically to incentivize the completion of uniform horizontal layers.

The experimental results of this research provide encouraging evidence of the applicability of the proposed methodology in real-world industrial scenarios.
It is demonstrated that the approach of combining DRL with a layer-building pattern is valid, as even on a dataset of orders with a limited number of articles, the model constructs a layer in 79.22% of instances.
Moreover, the hierarchical decomposition of the learning policy exhibits strong generalization capabilities; specifically, the model trained exclusively for single-layer construction successfully operates within the full 3D environment, outperforming the basic Corner Point-based heuristic. A significant finding concerns the efficacy of the action masking mechanism based on heuristic stability. The validity of this approach is confirmed by the attainment of stable configurations in 95.3% of the tested orders (when not supported by the rigorous stability check), while simultaneously preserving sufficient degrees of freedom to enable learning an efficient policy.
This balance, coupled with the superior performance of the HDRL architecture compared to the O3DBPP-5-5 algorithm and the significantly reduced computational times, establishes a robust solution for automated logistics.

The proposed architecture also supports several promising directions for future work. A primary objective is extending the training in the full unrestricted scenario. Other interesting works involve scaling the framework to handle higher bin resolutions, such as 120×80 grids. To manage the resulting computational load, future iterations could implement dynamic input pruning of the spatial feature maps using the valid action mask. Additionally, the Option logic can be expanded. Currently, the layer-building strategy operates strictly on subsets of items with identical heights (with tolerance); allowing the system to group heterogeneous stacks of smaller items whose cumulative height matches the target layer would enable new packing strategies.
Finally, building on the adaptability of the CAR state representation, future work could extend this framework to strictly online scenarios. This would allow the agent to pack items as they arrive continuously on a conveyor belt, without requiring prior knowledge of the global inventory.

Bibliography

[1] S. Ali, A. G. Ramos, M. A. Carravilla, and J. F. Oliveira. On-line three-dimensional packing problems: A review of off-line and on-line solution approaches. *Computers & Industrial Engineering*, 168:108122, 2022. doi:10.1016/j.cie.2022.108122.

[2] S. Allen, E. K. Burke, M. Hyde, and G. Kendall. Evolving reusable 3d packing heuristics with genetic programming. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, page 931–938, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605583259. doi:10.1145/1569901.1570029.

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

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

[5] I. Araya and M.-C. Riff. A beam search approach to the container loading problem. *Computers & Operations Research*, 43:100–107, 2014. ISSN 0305-0548. doi:10.1016/j.cor.2013.09.003.

[6] I. Araya, K. Guerrero, and E. Nuñez. Vcs: A new heuristic function for selecting boxes in the single container loading problem. *Computers & Operations Research*, 82:27–35, 2017.

[7] I. Araya, M. Moyano, and C. Sanchez. A beam search algorithm for the biobjective container loading problem. *European Journal of Operational Research*, 286(2): 417–431, 2020.

[8] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture, 2016. URL `https://arxiv.org/abs/1609.05140`.

[9] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2016. URL `https://arxiv.org/abs/1409.0473`.

[10] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.

[11] S. Berndt, K. Jansen, and K.-M. Klein. Fully dynamic bin packing revisited. *Mathematical Programming*, 179:109–155, 2020. doi:10.1007/s10107-018-1325-x.

[12] E. E. Bischoff. Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research*, 168(3):952–966, 2006.

[13] A. Bortfeldt and G. Wäscher. Constraints in container loading – a state-of-the-art review. *European Journal of Operational Research*, 229(1):1–20, 2013. doi:10.1016/j.ejor.2012.12.006.

[14] A. Bortfeldt, H. Gehring, and D. Mack. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29(5):641–662, 2003. ISSN 0167-8191. doi:10.1016/S0167-8191(03)00047-4. Parallel computing in logistics.

[15] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. *Hyper-Heuristics: An Emerging Direction in Modern Search Technology*. 01 2003. ISBN 1402072635. doi:10.1007/0-306-48056-5_16.

[16] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. *A Classification of Hyper-heuristic Approaches*, pages 449–468. Springer US, Boston, MA, 2010. ISBN 978-1-4419-1665-5. doi:10.1007/978-1-4419-1665-5_15.

[17] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. Automating the packing heuristic design process with genetic programming. *Evolutionary Computation*, 20 (1):63–89, 2012. doi:10.1162/EVCO_a_00044.

[18] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013. doi:10.1057/jors.2013.71.

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

[20] Y. Chandak, G. Theocharous, J. Kostas, S. Jordan, and P. S. Thomas. Learning action representations for reinforcement learning, 2019. URL `https://arxiv.org/abs/1902.00183`.

[21] C.-F. Chien and J.-F. Deng. A container packing support system for determining and visualizing container packing patterns. *Decision Support Systems*, 37(1):23–34, 2004. ISSN 0167-9236. doi:10.1016/S0167-9236(02)00192-6.

[22] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. URL `https://arxiv.org/abs/1406.1078`.

[23] E. G. Coffman, J. Y.-T. Leung, and D. W. Ting. Bin packing: Maximizing the number of pieces packed. *Acta Informatica*, 9(3):263–271, 1978. doi:10.1007/BF00288885.

[24] T. G. Crainic, G. Perboli, and R. Tadei. Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing*, 20(3):368–384, 2008. doi:10.1287/ijoc.1070.0250.

[25] T. Dereli and G. Sena Das. A hybrid simulated annealing algorithm for solving multi-objective container-loading problems. *Applied Artificial Intelligence*, 24(5): 463–486, 2010.

[26] J. Drake, M. Hyde, K. Ibrahim, and E. Özcan. A genetic programming hyper-heuristic for the multidimensional knapsack problem. *Kybernetes*, 43:1500–1511, 11 2014. doi:10.1108/K-09-2013-0201.

[27] L. Duan, H. Hu, Y. Qian, Y. Gong, X. Zhang, Y. Xu, and J. Wei. A multi-task selected learning approach for solving 3d flexible bin packing problem, 2019. URL `https://arxiv.org/abs/1804.06896`.

[28] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990. ISSN 0377-2217. doi:10.1016/0377-2217(90)90350-K. Cutting and Packing.

[29] S. Elhedhli, F. Gzara, and B. Yildiz. Three-dimensional bin packing and mixed-case palletization. *INFORMS Journal on Optimization*, 1(4):323–352, 2019. doi:10.1287/ijoo.2019.0013.

[30] L. Epstein and E. Kleiman. Resource augmented semi-online bounded space bin packing. *Discrete Applied Mathematics*, 157(13):2785–2798, 2009. ISSN 0166-218X. doi:10.1016/j.dam.2009.03.015.

[31] T. Fanslau and A. Bortfeldt. A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, 22(2):222–235, 2010.

[32] T. Fanslau and A. Bortfeldt. A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, 22(2):222–235, 2010. doi:10.1287/ijoc.1090.0338.

[33] G. Fasano. A mip approach for some practical packing problems: Balancing con-

straints and tetris-like items. *4OR*, 2(2):161 – 174, 2004. doi:10.1007/s10288-004-0037-7. Cited by: 36.

[34] X. Feng, I. Moon, and J. Shin. Hybrid genetic algorithms for the three-dimensional multiple container packing problem. *Flexible Services and Manufacturing Journal*, 27(2):451–477, 2015. ISSN 1936-6590. doi:10.1007/s10696-013-9181-8.

[35] E. P. Ferreira and J. F. Oliveira. *Fekete and Schepers' Graph-based Algorithm for the Two-Dimensional Orthogonal Packing Problem Revisited*, pages 15–31. Gabler, Wiesbaden, 2008. doi:10.1007/978-3-8349-9777-7_2.

[36] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.

[37] Z. Gao, L. Wang, Y. Kong, and N. Y. Chong. Online 3d bin packing with fast stability validation and stable rearrangement planning. *ArXiv*, abs/2507.09123, 2025. doi:10.48550/arXiv.2507.09123.

[38] H. Gehring and A. Bortfeldt. A genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 4(5):401–418, 1997. ISSN 0969-6016. doi:10.1016/S0969-6016(97)00033-6.

[39] J. A. George and D. F. Robinson. A heuristic for packing boxes into a container. *Computers & Operations Research*, 7(3):147–156, 1980.

[40] Glosarix. Temporal difference error, 2025. URL `https://glosarix.com/en/glossary/temporal-difference-error-en/`.

[41] J. F. Gonçalves and M. G. Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 39(2):179–190, 2012.

[42] J. F. Gonçalves and M. G. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International journal of production economics*, 145(2): 500–510, 2013.

[43] J. F. Gonçalves and M. G. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145 (2):500–510, 2013. ISSN 0925-5273. doi:10.1016/j.ijpe.2013.04.019.

[44] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

[45] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012. doi:10.1109/TSMCC.2012.2218595.

[46] C. T. Ha, T. T. Nguyen, L. T. Bui, and R. Wang. An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the physical internet. In G. Squillero and K. Sim, editors, *Applications of Evolutionary Computation*, pages 140–155, Cham, 2017. Springer International Publishing. ISBN 978-3-319-55792-2. doi:10.1007/978-3-319-55792-2_10.

[47] J. Hasan, J. Kaabi, and Y. Harrath. Multi-objective 3d bin-packing problem. In *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, pages 1–5. IEEE, 2019.

[48] M. Hifi, I. Kacem, S. Nègre, and L. Wu. A linear programming approach for the three-dimensional bin-packing problem. *Electronic Notes in Discrete Mathematics*, 36:993–1000, 2010. ISSN 1571-0653. doi:10.1016/j.endm.2010.05.126. ISCO 2010 - International Symposium on Combinatorial Optimization.

[49] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu. Solving a new 3d bin packing problem with deep reinforcement learning method, 2017. URL `https://arxiv.org/abs/1708.05930`.

[50] R. Hu, J. Xu, B. Chen, M. Gong, H. Zhang, and H. Huang. Tap-net: transport-and-pack using reinforcement learning. *ACM Transactions on Graphics*, 39(6): 1–15, Nov. 2020. ISSN 1557-7368. doi:10.1145/3414685.3417796. URL `http://dx.doi.org/10.1145/3414685.3417796`.

[51] M. Hutsebaut-Buysse, K. Mets, and S. Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022. ISSN 2504-4990. doi:10.3390/make4010009.

[52] Y. Jiang, Z. Cao, and J. Zhang. Solving 3d bin packing problem via multimodal deep reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, page 1548–1550. International Foundation for Autonomous Agents and Multiagent Systems, 2021. ISBN 9781450383073.

[53] Y. Jiang, Z. Cao, and J. Zhang. Learning to solve 3-d bin packing problem via deep reinforcement learning and constraint programming. *IEEE Transactions on Cybernetics*, 53(5):2864–2875, 2023. doi:10.1109/TCYB.2021.3121542.

[54] Y. Jin, Y. Ding, X. Pan, K. He, L. Zhao, T. Qin, L. Song, and J. Bian. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem, 2023. URL https://arxiv.org/abs/2304.09407.

[55] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314, 1974. doi:10.1016/S0022-0000(74)80026-7.

[56] F. Kagerer, M. Beinhofer, S. Stricker, and A. Nüchter. Bed-bpp: Benchmarking dataset for robotic bin packing problems. *The International Journal of Robotics Research*, 42(11):1007–1014, 2023. doi:10.1177/02783649231193048.

[57] K. Kang, I. Moon, and H. Wang. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied Mathematics and Computation*, 219(3):1287–1299, 2012. ISSN 0096-3003. doi:10.1016/j.amc.2012.07.036.

[58] K. Karabulut and M. M. İnceoğlu. A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method. In *International Conference on Advances in Information Systems*, pages 441–450. Springer, 2004.

[59] M. Klimek, H. Michalewski, and P. Milos. Hierarchical reinforcement learning with parameters. In S. Levine, V. Vanhoucke, and K. Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 301–313. PMLR, 2017. URL https://proceedings.mlr.press/v78/klimek17a.html.

[60] D. V. Kurpel, C. T. Scarpin, J. E. Pécora Junior, C. M. Schenekemberg, and L. C. Coelho. The exact solutions of several types of container loading problems. *European Journal of Operational Research*, 284(1):87–107, 2020. ISSN 0377-2217. doi:10.1016/j.ejor.2019.12.012.

[61] A. Laterre, Y. Fu, M. K. Jabri, A.-S. Cohen, D. Kas, K. Hajjar, T. S. Dahl, A. Kerkeni, and K. Beguir. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization, 2018. URL https://arxiv.org/abs/1807.01672.

[62] D. Li, Z. Gu, Y. Wang, C. Ren, and F. C. Lau. One model packs thousands of items with recurrent conditional query learning. *Knowledge-Based Systems*, 235:107683, Jan. 2022. ISSN 0950-7051. doi:10.1016/j.knosys.2021.107683. URL http://dx.doi.org/10.1016/j.knosys.2021.107683.

[63] X. Li and K. Zhang. A hybrid differential evolution algorithm for multiple container loading problem with heterogeneous containers. *Computers & Industrial Engineering*, 90:305–313, 2015. ISSN 0360-8352. doi:10.1016/j.cie.2015.10.007.

[64] A. Lim, H. Ma, C. Qiu, and W. Zhu. The single container loading problem with axle weight constraints. *International Journal of Production Economics*, 144(1):358–369, 2013.

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

[66] S. Liu, W. Tan, Z. Xu, and X. Liu. A tree search algorithm for the container loading problem. *Computers & Industrial Engineering*, 75:20–30, 2014.

[67] S. Liu, W. Tan, Z. Xu, and X. Liu. A tree search algorithm for the container loading problem. *Computers & Industrial Engineering*, 75:20–30, 2014.

[68] A. Lodi, S. Martello, and D. Vigo. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, 141(2):410–420, 2002.

[69] A. Lodi, S. Martello, and D. Vigo. Tspack: a unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research*, 131(1):203–213, 2004.

[70] F.-M. Luo, T. Xu, H. Lai, X.-H. Chen, W. Zhang, and Y. Yu. A survey on model-based reinforcement learning, 2022. URL `https://arxiv.org/abs/2206.09328`.

[71] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48, 02 1998. doi:10.1287/opre.48.2.256.12386.

[72] S. Martello, D. Pisinger, and D. Vigo. Three-dimensional bin packing problem. *Operations Research*, 48(2):256 – 267, 2000. doi:10.1287/opre.48.2.256.12386.

[73] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. Model-based reinforcement learning: A survey, 2022. URL `https://arxiv.org/abs/2006.16712`.

[74] H. Mostaghimi Ghomi, B. G. St Amour, and W. Abdul-Kader. Three-dimensional container loading: A simulated annealing approach. 2017.

[75] A. Moura and J. F. Oliveira. A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20(4):50–57, 2005.

[76] A. Moura and J. F. Oliveira. A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20(4):50 – 57, 2005. doi:10.1109/MIS.2005.57.

[77] C. Munien, S. Mahabeer, E. Dzitiro, S. Singh, S. Zungu, and A. E.-S. Ezugwu. Metaheuristic approaches for one-dimensional bin packing prob-

lem: A comparative performance study. *IEEE Access*, 8:227438–227465, 2020. doi:10.1109/ACCESS.2020.3046185.

[78] N. Nepomuceno, P. Pinheiro, and A. L. Coelho. Tackling the container loading problem: a hybrid approach based on integer linear programming and genetic algorithms. In *European conference on evolutionary computation in combinatorial optimization*, pages 154–165. Springer, 2007.

[79] B. K. A. Ngoi and K. Whybrew. A fast spatial representation method (applied to fixture design). *The International Journal of Advanced Manufacturing Technology*, 8 (2):71–77, 1993.

[80] S. Nishiyama, C. Lee, and T. Mashita. Designing a flexible evaluation of container loading using physics simulation. In *International Conference on Optimization and Learning*, pages 255–268. Springer, 2020.

[81] J. Olsson, T. Larsson, and N.-H. Quttineh. Automating the planning of container loading for atlas copco: Coping with real-life stacking and stability constraints. *European Journal of Operational Research*, 280(3):1018–1034, 2020. ISSN 0377-2217. doi:10.1016/j.ejor.2019.07.057.

[82] J. Olsson, T. Larsson, and N.-H. Quttineh. Automating the planning of container loading for atlas copco: Coping with real-life stacking and stability constraints. *European Journal of Operational Research*, 280(3):1018–1034, 2020.

[83] C. Paquay, M. Schyns, and S. Limbourg. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research*, 23(1-2):187–213, 2016. doi:10.1111/itor.12111.

[84] F. Parreño, R. Alvarez-Valdes, J. M. Tamarit, and J. F. Oliveira. A maximal-space algorithm for the container loading problem. *INFORMS Journal on Computing*, 20 (3):412–422, 2008. doi:10.1287/ijoc.1070.0254.

[85] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2):382–392, 2002. ISSN 0377-2217. doi:10.1016/S0377-2217(02)00132-7.

[86] D. Pisinger. Heuristics for the container loading problem. *European journal of operational research*, 141(2):382–392, 2002.

[87] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, NY, USA, 1994. ISBN 978-0471727828.

[88] Q. Que, F. Yang, and D. Zhang. Solving 3d packing problem using transformer network and reinforcement learning. *Expert Systems with Applications*, 214:119153, 2023. ISSN 0957-4174. doi:10.1016/j.eswa.2022.119153.

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

[90] A. G. Ramos, E. Silva, and J. F. Oliveira. A new load balance methodology for container loading problem in road transportation. *European Journal of Operational Research*, 266(3):1140–1152, 2018.

[91] A. G. Ramos, E. Silva, and J. F. Oliveira. A new load balance methodology for container loading problem in road transportation. *European Journal of Operational Research*, 266(3):1140–1152, 2018. ISSN 0377-2217. doi:10.1016/j.ejor.2017.10.050.

[92] A. G. Ramos, E. Silva, and J. F. Oliveira. A new load balance methodology for container loading problem in road transportation. *European Journal of Operational Research*, 266(3):1140–1152, 2018. ISSN 0377-2217. doi:10.1016/j.ejor.2017.10.050.

[93] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

[94] L. Sheng, S. Xiuqin, C. Changjian, Z. Hongxia, S. Dayong, and W. Feiyue. Heuristic algorithm for the container loading problem with multiple constraints. *Computers & Industrial Engineering*, 108:149–164, 2017.

[95] L. Sheng, S. Xiuqin, C. Changjian, Z. Hongxia, S. Dayong, and W. Feiyue. Heuristic algorithm for the container loading problem with multiple constraints. *Computers & Industrial Engineering*, 108:149–164, 2017.

[96] L. Sheng, S. Xiuqin, C. Changjian, Z. Hongxia, S. Dayong, and W. Feiyue. Heuristic algorithm for the container loading problem with multiple constraints. *Computers & Industrial Engineering*, 108:149–164, 2017.

[97] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. URL `https://arxiv.org/abs/1712.01815`.

[98] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks, 2014. URL `https://arxiv.org/abs/1409.3215`.

[99] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[100] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[101] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1): 181–211, 1999. ISSN 0004-3702. doi:10.1016/S0004-3702(99)00052-1.

[102] J.-F. Tsai, P.-C. Wang, and M.-H. L. and. A global optimization approach for solving three-dimensional open dimension rectangular packing problems. *Optimization*, 64 (12):2601–2618, 2015. doi:10.1080/02331934.2013.877906.

[103] Y. Tsang, D. Mo, K. Chung, and C. Lee. A deep reinforcement learning approach for online and concurrent 3d bin packing optimisation with bin replacement strategies. *Computers in Industry*, 164:104202, 2025. ISSN 0166-3615. doi:10.1016/j.compind.2024.104202.

[104] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[105] R. Verma, A. Singhal, H. Khadilkar, A. Basumatary, S. Nayak, H. V. Singh, S. Kumar, and R. Sinha. A generalized reinforcement learning algorithm for online 3d bin-packing, 2020. URL https://arxiv.org/abs/2007.00463.

[106] R. Verma, A. Singhal, H. Khadilkar, A. Basumatary, S. Nayak, H. V. Singh, S. Kumar, and R. Sinha. A generalized reinforcement learning algorithm for online 3d bin-packing. *arXiv preprint arXiv:2007.00463*, 2020.

[107] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks, 2017. URL https://arxiv.org/abs/1506.03134.

[108] B. Wang, Z. Lin, W. Kong, and H. Dong. Bin packing optimization via deep reinforcement learning. *IEEE Robotics and Automation Letters*, 10(3):2542–2549, 2025. doi:10.1109/LRA.2025.3534070.

[109] R. J. Williams. Simple statistical gradient-following algorithms for connectionist

reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. doi:10.1007/BF00992696.

[110] C.-C. Wong, T.-T. Tsai, and C.-K. Ou. Integrating heuristic methods with deep reinforcement learning for online 3d bin-packing optimization. *Sensors*, 24(16), 2024. ISSN 1424-8220. doi:10.3390/s24165370.

[111] Y. Wu, W. Li, M. Goh, and R. De Souza. Three-dimensional bin packing problem with variable bin height. *European journal of operational research*, 202(2):347–355, 2010.

[112] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation, 2017. URL `https://arxiv.org/abs/1708.05144`.

[113] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007. ISSN 0377-2217. doi:10.1016/j.ejor.2005.12.047.

[114] H. Xiong, K. Ding, W. Ding, J. Peng, and J. Xu. Towards reliable robot packing system based on deep reinforcement learning. *Advanced Engineering Informatics*, 57: 102028, 2023. ISSN 1474-0346. doi:10.1016/j.aei.2023.102028.

[115] H. Xiong, K. Ding, W. Ding, X. Qiu, K. Janschek, and J. Xu. Enhancing robotics online 3d bin packing: A comparative study of conventional heuristic and deep reinforcement learning approaches. In *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, pages 4083–4089, 2024. doi:10.1109/CASE59546.2024.10711723.

[116] H. Xiong, C. Guo, J. Peng, K. Ding, W. Chen, X. Qiu, L. Bai, and J. Xu. Gopt: Generalizable online 3d bin packing via transformer-based deep reinforcement learning. *IEEE Robotics and Automation Letters*, 9(11):10335–10342, 2024. doi:10.1109/LRA.2024.3468161.

[117] S. Yang, S. Song, S. Chu, R. Song, J. Cheng, Y. Li, and W. Zhang. Heuristics integrated deep reinforcement learning for online 3d bin packing. *IEEE Transactions on Automation Science and Engineering*, 21(1):939–950, 2024. doi:10.1109/TASE.2023.3235742.

[118] Z. Yang, S. Yang, S. Song, W. Zhang, R. Song, J. Cheng, and Y. Li. Packerbot: Variable-sized product packing with heuristic deep reinforcement learning. In *2021*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5002–5008, 2021. doi:10.1109/IROS51168.2021.9635914.

[119] H. Yin, F. Chen, and H. He. Solving offline 3d bin packing problem with large-sized bin via two-stage deep reinforcement learning. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, page 2576–2578. International Foundation for Autonomous Agents and Multiagent Systems, 2024. ISBN 9798400704864.

[120] J. Zhang, B. Zi, and X. Ge. Attend2pack: Bin packing through deep reinforcement learning with attention, 2021. URL `https://arxiv.org/abs/2107.04333`.

[121] T. Zhang, Z. Wu, Y. Chen, Y. Wang, B. Liang, S. Moura, M. Tomizuka, M. Ding, and W. Zhan. Physics-aware robotic palletization with online masking inference, 2025. URL `https://arxiv.org/abs/2502.13443`.

[122] H. Zhao and K. Xu. Learning efficient online 3d bin packing on packing configuration trees. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=bfuGjlCwAq`.

[123] H. Zhao, Q. She, C. Zhu, Y. Yang, and K. Xu. Online 3d bin packing with constrained deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):741–749, May 2021. doi:10.1609/aaai.v35i1.16155.

[124] H. Zhao, C. Zhu, X. Xu, H. Huang, and K. Xu. Learning practically feasible policies for online 3d bin packing, 2023. URL `https://arxiv.org/abs/2108.13680`.

[125] S. Zhao. *Mathematical Foundations of Reinforcement Learning*. Springer Press, 2025.

[126] X. Zhao, J. A. Bennell, T. Bektaş, and K. Dowsland. A comparative review of 3d container loading algorithms. *International Transactions in Operational Research*, 23(1-2):287–320, 2016. doi:10.1111/itor.12094.

[127] J.-N. Zheng, C.-F. Chien, and M. Gen. Multi-objective multi-population biased random-key genetic algorithm for the 3-d container loading problem. *Computers & Industrial Engineering*, 89:80–87, 2015.

[128] P. Zhou, Z. Gao, C. Li, and N. Y. Chong. An efficient deep reinforcement learning model for online 3d bin packing combining object rearrangement and stable placement, 2024. URL `https://arxiv.org/abs/2408.09694`.

[129] W. Zhu and A. Lim. A new iterative-doubling greedy–lookahead algorithm for the

single container loading problem. *European Journal of Operational Research*, 222(3): 408–417, 2012. ISSN 0377-2217. doi:10.1016/j.ejor.2012.04.036.

A | Appendix A

In this appendix, all the algorithms cited in the body of the thesis are reported

Algorithm A.1 Rigorous Structural Stability Validation

Input: New object footprint $(x_i, y_i, l_i(o), w_i(o))$, Pseudo-planar Heightmap $\mathcal{H}^m_{plane}$, Feasibility Map $\mathcal{F}^m_t$, Uncertainty δ_{CoM}

Output: Boolean stability flag *valid*, Support polygon $P^{\triangle}_{\text{new}}$, Support height h^s

// Find max height in the placement footprint

$h^s \leftarrow \max \mathcal{H}^m_{plane}[x_i : x_i + l_i(o),\ y_i : y_i + w_i(o)]$

// Extract gap-tolerant contact points

$\text{PS}_{\text{contact}} \leftarrow \{(x, y) \mid \mathcal{H}^m_{plane}(x, y) = h^s\}$

// Extract underlying structurally sound points

$\text{PS}_{\text{feasible}} \leftarrow \{(x, y) \mid \mathcal{F}^m_t(x, y) = 1\}$

// Compute Convex Hull of the valid intersection

$P^{\triangle}_{\text{new}} \leftarrow \text{CH}(\text{PS}_{\text{contact}} \cap \text{PS}_{\text{feasible}})$

// Derive CoM area considering bounded uncertainty

$C_{\text{new}} \leftarrow \text{ComputeCoMArea}(x_i, y_i, l_i(o), w_i(o), \delta_{\text{CoM}})$

if $C_{\text{new}} \subseteq P^{\triangle}_{\text{new}}$ **then**

    $valid \leftarrow$ True

else

    $valid \leftarrow$ False

end if

return $(valid, P^{\triangle}_{\text{new}}, h^s)$

Algorithm A.2 Modified Structural Stability Update (SSU)

Input: Current Feasibility Map $\mathcal{F}_t^m$, Validated support polygon $P_{\text{new}}^{\triangle}$, New object footprint $(x_i, y_i, l_i(o), w_i(o))$

Output: Updated Feasibility Map $\mathcal{F}_{t+1}^m$

$\mathcal{F}_{t+1}^m \leftarrow \mathcal{F}_t^m$ *// Initialize next state*

// Apply strict mask over the newly covered spatial footprint

for $x \in [x_i, x_i + l_i(o) - 1]$ **do**
  for $y \in [y_i, y_i + w_i(o) - 1]$ **do**
    if $(x, y) \in P_{\text{new}}^{\triangle}$ **then**
      $\mathcal{F}_{t+1}^m(x, y) \leftarrow 1$ *// Load-bearing region (feasible)*
    else
      $\mathcal{F}_{t+1}^m(x, y) \leftarrow 0$ *// Cantilever region (unfeasible)*
    end if
  end for
end for
return $\mathcal{F}_{t+1}^m$

Algorithm A.3 Computation of the Pseudo-Planar Heightmap

Input: Current active bin heightmap $\mathcal{H}^m \in \mathbb{R}^{L \times W}$, minimum box dimension w_{min}

Output: Pseudo-planar heightmap $\mathcal{H}^m_{plane} \in \mathbb{R}^{L \times W}$

Initialize intermediate heightmap $\mathcal{H}^m_{max} \leftarrow \mathbf{0}^{L \times W}$

Initialize final heightmap $\mathcal{H}^m_{plane} \leftarrow \mathbf{0}^{L \times W}$

// ***Step 1: Dilation (Maximum Filter)***

for each coordinate (x, y) in $\mathcal{H}^m$ **do**

  Define spatial window $\mathcal{W}_{x,y}$ of size $w_{min} \times w_{min}$ anchored at (x, y)

  Let $\tilde{\mathcal{H}}^m[i,j] = \begin{cases} \mathcal{H}^m[i,j] & \text{if } (i,j) \text{ is within bounds} \\ 0 & \text{otherwise} \end{cases}$

  $\mathcal{H}^m_{max}[x, y] \leftarrow \max_{(i,j) \in \mathcal{W}_{x,y}} \tilde{\mathcal{H}}^m[i, j]$

end for

// ***Step 2: Erosion (Minimum Filter)***

for each coordinate (x, y) in $\mathcal{H}^m_{max}$ **do**

  Define spatial window $\mathcal{W}_{x,y}$ of size $w_{min} \times w_{min}$ anchored at (x, y)

  Let $\tilde{\mathcal{H}}^m_{max}[i,j] = \begin{cases} \mathcal{H}^m_{max}[i,j] & \text{if } (i,j) \text{ is within bounds} \\ +\infty & \text{otherwise} \end{cases}$

  $\mathcal{H}^m_{plane}[x, y] \leftarrow \min_{(i,j) \in \mathcal{W}_{x,y}} \tilde{\mathcal{H}}^m_{max}[i, j]$

end for

return $\mathcal{H}^m_{plane}$

Algorithm A.4 Verification of Pseudo-Planarity

Input: Pseudo-planar heightmap $\mathcal{H}^m_{plane} \in \mathbb{R}^{L \times W}$

Output: Boolean indicator is_planar

Extract reference height $h_{ref} \leftarrow \mathcal{H}^m_{plane}[0, 0]$
$is\_planar \leftarrow$ **True**

for each coordinate (x, y) in H_{plane} **do**
    if $\mathcal{H}^m_{plane}[x, y] \neq h_{ref}$ **then**
        $is\_planar \leftarrow$ **False**
        break *// Terminate evaluation early*
    end if
end for

return is_planar

Algorithm A.5 Vectorized Computation of Plane Features

Input: Pseudo-planar heightmap $\mathcal{H}^m_{plane} \in \mathbb{R}^{L\times W}$, height tolerance δ_h
Output: Feature matrices $D^N, D^S, D^E, D^W \in \mathbb{N}^{L\times W}$

// ***Row-wise processing (East and West features)***
for $i = 0$ **to** $L-1$ **do**
  Extract row vector $\mathbf{r} \leftarrow \mathcal{H}^m_{plane}[i,:]$
  Find break indices $\mathcal{B} \leftarrow \{j \in [1, W-1] \mid |\mathbf{r}[j] - \mathbf{r}[j-1]| > \delta_h\}$
  Augment breaks with grid boundaries: $\mathcal{B}^* \leftarrow \{0\} \cup \mathcal{B} \cup \{W\}$
  // *Iterate over contiguous pseudo-planar segments*
  for each consecutive pair (b_{start}, b_{end}) in sorted $\mathcal{B}^*$ **do**
    for $j = b_{start}$ **to** $b_{end} - 1$ **do**
      $D^W[i,j] \leftarrow j - b_{start}$ // *Distance to the left boundary/break*
      $D^E[i,j] \leftarrow (b_{end} - 1) - j$ // *Distance to the right boundary/break*
    end for
  end for
end for

// ***Column-wise processing (North and South features)***
for $j = 0$ **to** $W-1$ **do**
  Extract column vector $\mathbf{c} \leftarrow \mathcal{H}^m_{plane}[:,j]$
  Find break indices $\mathcal{B} \leftarrow \{i \in [1, L-1] \mid |\mathbf{c}[i] - \mathbf{c}[i-1]| > \delta_h\}$
  Augment breaks with grid boundaries: $\mathcal{B}^* \leftarrow \{0\} \cup \mathcal{B} \cup \{L\}$
  // *Iterate over contiguous pseudo-planar segments*
  for each consecutive pair (b_{start}, b_{end}) in sorted $\mathcal{B}^*$ **do**
    for $i = b_{start}$ **to** $b_{end} - 1$ **do**
      $D^N[i,j] \leftarrow i - b_{start}$ // *Distance to the top boundary/break*
      $D^S[i,j] \leftarrow (b_{end} - 1) - i$ // *Distance to the bottom boundary/break*
    end for
  end for
end for

return D^N, D^S, D^E, D^W

Algorithm A.6 Vectorized Fast Stability Check

Input: Inventory state s_t^u (with $2N_k$ clusters $\nu_{k,o}$), Container feature map s_t^c, Feasibility mask $\mathcal{F}^m$

Output: Global stability mask $\mathcal{M}_{\text{stab}} \in \{0,1\}^{2N_k \times L \times W}$

Extract $\mathcal{H}^m_{plane}$ and directional features d^N, d^S, d^E, d^W from s_t^c
Initialize $\mathcal{M}_{\text{stab}}$ as an empty set of masks

for each oriented virtual cluster $\nu_{k,o} \in s_t^u$ **do**
    $l \leftarrow l_k(o)$
    $w \leftarrow w_k(o)$
    $hl \leftarrow \lfloor l/2 \rfloor$ *// Compute integer half-length for CoM offset*
    $hw \leftarrow \lfloor w/2 \rfloor$ *// Compute integer half-width for CoM offset*
    Initialize item-specific mask $\mathbf{M} \leftarrow \mathbf{0}^{L \times W}$

// Vectorized boolean evaluation over the entire (x, y) grid
    $\mathbf{C}_X \leftarrow \left(d^E_{x,y} > hl\right) \vee \left(d^W_{x+l,y} > hl\right)$
    $\mathbf{C}_Y \leftarrow \left(d^N_{x,y} > hw\right) \vee \left(d^S_{x,y+w} > hw\right)$
    $\mathbf{C}_Z \leftarrow \left(\mathcal{H}^m_{plane}[x, y] == \mathcal{H}^m_{plane}[x + hl, y + hw]\right)$
    $\mathbf{C}_F \leftarrow (\mathcal{F}^m[x + hl, y + hw] == 1)$

// Compute final logical AND across all spatial conditions
    $\mathbf{M} \leftarrow \mathbf{C}_X \wedge \mathbf{C}_Y \wedge \mathbf{C}_Z \wedge \mathbf{C}_F$
    Append $\mathbf{M}$ to $\mathcal{M}_{\text{stab}}$
end for

return $\mathcal{M}_{\text{stab}}$

Algorithm A.7 Step-wise Dead Space Penalty Computation

Input: Placed item bounds (x_s, x_e, y_s, y_e, h), Global action mask $\mathcal{M}^{\text{glob}} = \bigvee_n \mathcal{M}_{t+1}^{c,(n)}$, Updated heightmap $\mathcal{H}_{t+1}^m$, Global penalized mask $\mathcal{M}_{penal}$

Output: Penalty value $P_{dead} \in \mathbb{R}^-$

Initialize $V_{waste} \leftarrow 0$

*// **Check 1 & 2: Forward Projected Space (Blocked Lanes)***

Define L-axis strip: $\mathcal{R}_X^+ \leftarrow [x_e, L) \times [0, y_e)$

if $\max_{(x,y)\in\mathcal{R}_X^+} \mathcal{M}^{\text{glob}}(x,y) == 0$ **then** *// No valid actions remain in the lane*

  $V_{waste} \leftarrow V_{waste} + ((L - x_e) \times (y_e - y_s) \times h)$

end if

Define W-axis strip: $\mathcal{R}_Y^+ \leftarrow [0, x_e) \times [y_e, W)$

if $\max_{(x,y)\in\mathcal{R}_Y^+} \mathcal{M}^{\text{glob}}(x,y) == 0$ **then**

  $V_{waste} \leftarrow V_{waste} + ((x_e - x_s) \times (W - y_e) \times h)$

end if

*// **Check 3 & 4: Backward Covered Space (Covered Depressions)***

Support height $z_{ref} \leftarrow \mathcal{H}_{t+1}^m(x_s, y_s) - h$

Define backward regions: $\mathcal{R}_X^- \leftarrow [0, x_s) \times [y_s, y_e)$ and $\mathcal{R}_Y^- \leftarrow [x_s, x_e) \times [0, y_s)$

// Find $w_{min} \times w_{min}$ strictly negative windows using dynamic sliding mask

$\mathbf{M}_X \leftarrow \textsc{NegativeWindows}(\mathcal{H}_{t+1}^m[\mathcal{R}_X^-] - z_{ref} + \delta_h, w_{min})$

$\mathbf{M}_Y \leftarrow \textsc{NegativeWindows}(\mathcal{H}_{t+1}^m[\mathcal{R}_Y^-] - z_{ref} + \delta_h, w_{min})$

// Exclude already penalized cells to prevent overlapping penalties

$\mathbf{M}_{new} \leftarrow (\mathbf{M}_X \vee \mathbf{M}_Y) \wedge \neg\mathcal{M}_{penal}$

$V_{waste} \leftarrow V_{waste} + \sum(\mathbf{M}_{new}) \times$ (depth differentials)

$\mathcal{M}_{penal} \leftarrow \mathcal{M}_{penal} \vee \mathbf{M}_{new}$ *// Update global persistent mask*

return $P_{dead} = -V_{waste}/V_{norm}$ *// Normalized to standard volumetric units (e.g., dm^3)*

List of Figures

List of Tables

List of Symbols

Table A.1: List of Symbols

Symbol	Description	Unit
Physical Dimensions & Coordinates		
L, W, H	Length, width, and maximum height of the bin	mm
$\tilde{H}_t$	Current maximum stacked height of the active bin at time t	mm
$\mathcal{H}_t^m$	Bin heightmap at time t	mm
l_i, w_i, h_i	Length, width, and height of box i	mm
x_i, y_i, z_i	Placement coordinates of the bottom-left-back corner of box i	mm
o_i	Selected horizontal orientation of box i ($o_i \in \mathcal{O}$)	–
w_{min}	Minimum possible dimension across all available boxes	mm
δ_h	Height tolerance threshold for the pseudo-planar surface	mm
MDP Formulation		
$\mathcal{S}_t$	Global environment state at time t ($\mathcal{S}_t = \langle s_t^u, \mathcal{C}_t \rangle$)	–
$\mathcal{I}_t$	Set of remaining physical unpacked boxes at time t	–

Continued on next page

Table A.1 – continued from previous page

Symbol	Description	Unit
s_t^u	Inventory state representation of unpacked boxes	–
$\nu_{k,o}$	Oriented virtual cluster tuple $(l_k(o), w_k(o), h_k, o, q_k)$ in s_t^u	–
N_k	Total number of unique physical box types in the current order	–
q_k	Quantity of items available in cluster k	–
$\mathcal{M}_t^u$	Action mask for the items	–
$\mathcal{C}_t$	Configuration of the active container ($\mathcal{C}_t = \langle s_t^c, \mathcal{M}_t^c \rangle$)	–
s_t^c	Spatial feature map of the active container ($s_t^c \in \mathbb{Z}^{L \times W \times 5}$)	–
$\mathcal{M}_t^c$	Heuristic action mask for the container cells	–
$\mathcal{A}_{\text{eff}}$	Effective 2D reduced action space	–
CP	Set of valid Corner Point coordinates	–
$\mathbf{v}_{ij}$	5-dimensional feature vector at grid coordinate (i, j)	–
$d_{ij}^{N,S,E,W}$	Plane features mapping distance to edge/drop in cardinal directions	mm
$\mathcal{O}$	Set of valid horizontal rotations $\{0, 1\}$	–
$\mathcal{A}_t$	Action executed at time t	–
Hierarchical Framework		
$\Omega(S_t)$	Option space available at state S_t	–
ω_{free}	Primitive single-item placement Option	–

Continued on next page

Table A.1 – continued from previous page

Symbol	Description	Unit
$\omega_{layer}(h)$	Temporally extended Layer-building Option parameterized by height h	–
$\mathcal{I}_h$	Initiation Set for $\omega_{layer}(h)$	–
π_h	Intra-option policy for $\omega_{layer}(h)$	–
β_h	Termination condition for $\omega_{layer}(h)$	–
$\bar{t}$	Temporal duration of an executed Option	steps
H_{cand}	Set of candidate layer target heights (top-K most frequent in inventory)	mm
s_h^u	Subset of the inventory state containing all available items with height h	–
z_{curr}	Vertical elevation of the currently active pseudo-planar surface	mm
$\mathcal{M}_h^c$	Option-specific container state action mask enforcing placement at z_{curr}	–
π_m	High-level Manager policy (heuristic-driven)	–
π_w	Low-level Worker policy (shared neural network parameterized by θ)	–
A_h	Total potential horizontal coverage area for the items in subset s_h^u	mm^2
h^*	Optimal layer height parameter selected by the Manager	mm
$\mathcal{M}_\omega$	Conditional action mask dynamically applied by the selected option ω	–
Action space decomposition		

Continued on next page

Table A.1 – continued from previous page

Symbol	Description	Unit
ι_{min}	Fictitious box with minimum order dimensions $(w_{min} \times w_{min})$	–
CAR_k	Corner-Anchored Region generated at corner point cp_k	–
$\mathcal{M}_{\text{CAR}}$	Preliminary action mask formed by the boolean union of all CARs	–
$\mathcal{F}^m$	Global structural feasibility mask	–
$\mathcal{H}^m_{plane}$	Pseudo-planar heightmap for stability checks	mm
$\mathcal{M}^{(k,o)}_{\text{stab}}$	Item-specific boolean mask enforcing physical stability	–
$\mathcal{M}^{(k,o)}_{\text{cont}}$	Item-specific boolean mask enforcing container boundary constraints	–
$\mathcal{M}^{(k,o)}_{\text{edge}}$	Item-specific boolean mask enforcing edge-alignment heuristic	–
$\mathcal{M}^{c,(k,o)}_t$	Final intersected valid placement mask for oriented cluster $\nu_{k,o}$	–
Reinforcement Learning Parameters		
R_t	Scalar reward received at time t	–
$J(\pi)$	Expected cumulative discounted reward for policy π	–
γ	Discount factor	–
α	Learning rate	–

Table A.2: Abbreviations

Abbreviation	**Full namme**
RL	Reinforcement Learning
DRL	Deep RL
PPO	Proximal Policy Optimization
LP	Loading Position
CP	Corner Point
EP	Extreme Point
EMS	Empty Maximal Space
MCTS	Monte Carlo Tree Search