Journal of Computer and Communications, 2023, 11, 156-173
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Online 3D Packing Problem Based on Bi-Value Guidance

Mingkai Qi, Liye Zhang*

School of Computer Science and Technology, Shandong University of Technology, Zibo, China
Email: qmk3218@163.com, *zhangliye@sdut.edu.cn

How to cite this paper: Qi, M.K. and Zhang, L.Y. (2023) Online 3D Packing Problem Based on Bi-Value Guidance. *Journal of Computer and Communications*, **11**, 156-173.
https://doi.org/10.4236/jcc.2023.117010

Received: June 6, 2023
Accepted: July 28, 2023
Published: July 31, 2023

Abstract

The online 3D packing problem has received increasing attention in recent years due to its practical value. However, the problem itself possesses some peculiar properties, such as sequential decision-making and the large size of the state space, which have made the use of reinforcement learning with Markov decision processes a popular approach for solving this problem. In this paper, we focus on the problem of high variance in value estimation caused by reward uncertainty in the presence of highly uncertain dynamics. To address this, proposed a solution based on auxiliary tasks and intrinsic rewards for the online 3D bin packing problem, guided by a binary-valued network, to assist the agent in learning the policy within the framework of actor-critic deep reinforcement learning. Specifically, the maintenance of two-valued networks and the utilization of multi-valued network estimates are employed to replace the original value estimates, aiming to provide better guidance for the learning of policy networks. Experimentally, it has been demonstrated that our model can achieve more robust learning and outperform previous works in terms of performance.

Keywords

Deep Learning, Reinforcement Learning, Bin Packing, Value Estimation

1. Introduction

As one of the most classic combinatorial optimization problems, the Bin Packing Problem (BPP) can be traced back to Gauss's research on layout problems in 1831. In previous studies, most of them focused on 1D and 2D packing problems. With the accumulation of research results and the development of computer technology, the three-dimensional packing problem has gradually become a popular academic topic [1]. As one of the variants of the 3D packing problem, the online 3D packing problem has received additional attention in the engineering community due to its inherent complexity and practicality. In particular, the combination of online three-dimensional packing and AI has gradually become a popular solution with the development and generalization of AI technology. By applying point cloud techniques, deep reinforcement learning, and other methods, solutions to online 3D packing problems are becoming more complex and practical, and can achieve excellent performance when faced with complex practical constraints and large-scale packing problems. The online three-dimensional packing simulation solution can be widely applied in various industries and scenarios that require cargo loading, such as automated palletizing assembly lines, warehouse management, vehicle loading, etc. Excellent packing solutions can help businesses or individuals quickly select suitable vehicles or containers for different types and quantities of goods, and generate optimal packing solutions based on algorithms, helping to save transportation costs and time. Even in scenarios with low logistics volume, a 1% increase in loading efficiency can result in millions of dollars in savings per year [2].

In general, the definition of the 3D-BPP problem is as follows: given a sequence of rectangular parallelepipeds I and a three-dimensional container, where the dimensions of the container are S_x, S_y and S_z, and each rectangular parallelepiped has dimensions s_i^x, s_i^y and s_i^z in the three directions, the objective is to pack all rectangular parallelepipeds in an axis-aligned manner into the container, subject to some realistic constraints, such as no overlapping or ensuring physical stability, so as to maximize the container utilization or the number of placed rectangular parallelepipeds [3]. Given the availability of prior knowledge about the objects to be packed, 3D-BPP problems can be classified into offline 3D-BPP and online 3D-BPP problems. Offline 3D-BPP refers to the scenario where the agent has prior access to complete information about the packing sequence and follows a pre-determined plan to pack the items into the containers. On the other hand, online 3D-BPP involves the case where the agent has partial prior knowledge of the object sequence and needs to pack them into the container upon their arrival without any additional adjustments. In comparison, online 3D-BPP is more complex but also more practical. An illustration of the packing process is shown on the left of **Figure 1**, while the two images on the right represent the perspective from which the object information is obtained and the height map of the container, respectively.

In solving the 3D-BPP problem, reward design is usually neglected. Typically, rewards are designed with the goal of solving the problem in mind. However, when using reinforcement learning to train an intelligent agent, exploration becomes challenging due to the scarcity of rewards. To address this issue, a common approach is to reshape rewards to aid exploration [4]. For example, the heuristic experience can be leveraged to shape the reward as a dense reward related to the volume of the box and the container to aid the learning of an intelligent agent. A major drawback of this reward reshaping is that sometimes, an intelligent agent may learn in the direction of the reshaped reward instead of the true

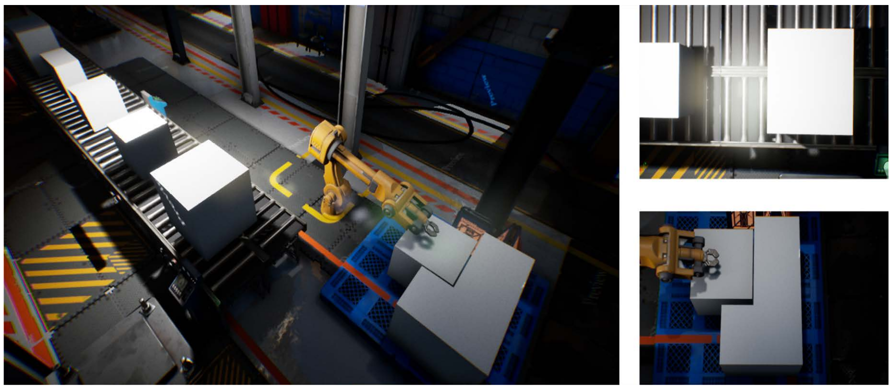

Figure 1. Palletizing schematic diagram.

objective. In other words, the reshaped reward objective is included in the original objective, resulting in a learning effect for the agent that does not satisfy the desired outcome. And, since reward design is closely related to value, problems with reward design can often affect an intelligent agent's judgment of value, especially in the actor-critic framework, where both actor and critic networks rely on value estimates for their updates. When the estimation is inaccurate, the performance of the network suffers, leading to poorer output policy solutions. To address this issue, we design an intrinsic reward-based dual-value guided network that targets dual rewards and complements the problem of large value estimation variance caused by uncertainty in future rewards through multi-valued estimation.

2. Related Work

2.1. 3D Bin Packing Problem

The bin packing problem holds a prominent place among the well-known challenges within the domain of combinatorial optimization. It is typically solved using heuristic-based algorithms or learning-based methods. Heuristic methods typically rely on existing empirical knowledge and involve manually designing a large number of rules to guide the solution process. However, these approaches heavily rely on substantial prior knowledge to support their design. In the past few years, there has been a growing trend to integrate heuristic algorithms with various techniques to enhance the effectiveness of problem-solving approaches. For example, Huang *et al.* [5] combined a Differential Evolution (DE) algorithm with a ternary search tree model. They used a ternary tree model to generate a set of suboptimal solutions as the initial population for DE, with the goal of solving the 3D bin packing problem. Ntanjana [6] combined the fi-heuristic algorithm with the genetic algorithm. They divided the packing process into two stages: the first stage employs a fi-heuristic algorithm supplemented with other techniques, while the second stage employs a genetic algorithm. This approach aims to optimize the initial permutation of a limited set of frames or the entire set of sequences without compromising the original pattern (elite strategy). In another study, Zhao *et al.* [7] solved the problem of three-dimensional irregular stacking by introducing a three-grid approximation technique to approximate irregular objects. They use a hybrid heuristic to place and compress individual objects, where chaos search is embedded into the Firefly algorithm to enhance algorithmic diversity.

In contrast to heuristic methods, learning-based methods utilize an end-to-end neural network to solve the packing problem. These methods generally exhibit stronger generalization and robustness, especially under complex real-world constraints and large state spaces. Zhu *et al.* [2] designed a data-driven tree search algorithm for 3D-BPP problems in large-scale scenes, aiming at faster computation. They use tree search to explore the solution space with complex constraints, and a convolutional neural network trained on historical data to guide tree pruning. Zhang *et al.* [8] focuses on BPPs in different contexts and draws inspiration from how the human brain solves similar decision-making problems. They proposed a brain-inspired model called BERM that leverages empirical information and learning paradigms to make decisions in different environments, thereby providing an optimal decision process for BPP tasks across multiple environments. Zhao *et al.* [9] introduced a different tree-based representation called the Packed Configuration Tree (PCT). PCT completely describes the packed state and action spaces in a tree structure and uses deep reinforcement learning algorithms for policy learning. PCT includes relations among all spatially configured nodes, which are enhanced during training based on heuristic rules, they achieve the solution in the continuous action space for the first time. In the context of DACC [10], the packaging problem is assumed to be solved in an approximately predictable dynamic environment, given the characteristics of datasets in industrial settings. They first discussed the high variance of the value estimates in the bin packing problem, and based on the effect of the two-stage MDP partition on the value estimates, designed a two-stage evaluation framework to amortize the uncertainty and reduce the variance of the value estimates. In Attend2Pack, Zhang *et al.* [11] proposed an end-to-end learning model based on a self-attention mechanism. They use a decomposable combinatorial action space and a preferential oversampling training technique to speed up policy learning. For an arbitrary number and size of boxes, Verma *et al.* [12] designed the RT-3D-BPP algorithm, dividing the decision-making process into two steps: using ground rules to select feasible position/orientation combinations, and employing a value-based RL algorithm for selection. The CQL [13] model utilizes conditional query learning to handle 2D and 3D packing problems. Conditional queries are embedded into the attention model and trained using a reinforcement learning approach, they claim to reduce the bin gap ratio by more than 10% in almost every case when compared to the contemporaneous method.

2.2. Dee Reinforcement Learning (DRL)

Deep Reinforcement Learning (DRL) synergistically merges the perceptual prowess of deep learning with the decision-making aptitude of reinforcement learning, resulting in an interactive learning approach. Unlike supervised learning, it focuses on finding a balance between exploration and exploitation [14]. The development of DRL can be broadly categorized into the following phases: Value-based Deep Reinforcement Learning: This approach uses a neural network to approximate the value function to estimate the long-term reward obtained by taking actions in various states. It is suitable for dealing with large, high-dimensional state spaces and complex action spaces. However, it requires a large number of interactions to update the value function, and may still suffer from the curse of dimensionality in high-dimensional state spaces. Representative algorithms include Deep Q-Network (DQN) [15], Double DQN [16], etc. Policy-based Deep Reinforcement Learning: This approach trains neural networks using policy gradient methods to learn the optimal policy that maximizes the rewards. An example of such an approach is the actor-critic algorithm. Model-based Deep Reinforcement Learning: This approach involves using neural networks to approximate a dynamical model of the environment, and performing planning and control based on the learned model. Examples include Alpha-Go [17] and Alpha Zero [18].

The optimization problem aims to find the optimal configuration or value among different possibilities. Depending on the nature of the configuration, these problems can be classified into two categories: problems with continuous variable configurations and problems with discrete variable configurations. Optimization problems in discrete spaces are commonly referred to as Combinatorial Optimization (CO) problems [19]. The defining feature of the CO problem is that the decision space is defined over a finite set of points, and the optimal solution to the problem can be intuitively obtained by exhaustive search. However, the number of solutions typically grows exponentially with the size of the problem, making it infeasible to exhaustively search for the optimal solution in polynomial time [20]. Previously, most methods for solving CO problems were concentrated in two categories: exact methods and approximation methods. With the advancement of neural network research, the use of neural network models to solve CO problems has gained attention. Inspired by natural language processing tasks, Vinyals *et al.* applied the Sequence-to-Sequence learning framework, a neural network model, to solve combinatorial optimization problems and introduced the Pointer Network (Ptr-Net) model [21]. In contrast to supervised learning methods such as Ptr-Net, which require the construction of a large number of labeled samples, reinforcement learning approaches can interactively make decisions in unknown and complex environments without prior knowledge of the full problem structure or the form of a feasible solution. Thus, reinforcement learning is more suitable for large-scale CO problems. Bello *et al.* built upon the Ptr-Net approach and combined neural networks with reinforcement learning to propose the Neural Combinatorial Optimization (NCO) model, improving the generalization capability of network models when solving CO problems [22]. When using reinforcement learning to solve the CO problem, the problem is typically defined as a Markov Decision Process, and the interaction between the agent and the environment is established by defining the state, action, and reward functions. For example, in the context of a packing problem, the agent's state can represent the items already placed in the container and the list of items to be placed, actions can correspond to the placement positions and orientations of the items in the container, and the reward can be defined as the packing utilization. Additionally, to address the issue of sparse rewards in combinatorial optimization problems, techniques such as hierarchical reinforcement learning or intrinsic rewards can be employed [23]. In hierarchical reinforcement learning, the agent learns a set of sub-tasks and utilizes intrinsic re-wards to facilitate learning during each sub-task.

3. Method

Within the realm of the online 3D bin packing problem, we establish its essence as a sequential decision procedure, where an agent is tasked with promptly assigning a set of items to a bounded 3D bin, armed with incomplete knowledge about the current or upcoming items. To satisfy constraints related to physical stability and sequential dependencies, we formulate this problem within a constrained Deep Reinforcement Learning (DRL) framework, utilizing the ACKTR algorithm [24] to design the network model. A hard constraint is imposed on the agent's action space in the form of an action mask, ensuring that the agent samples actions only from the valid action space. Our aim is to optimize the utilization of space within the container by maximizing the total volume of objects accommodated while adhering to the imposed constraints. In a typical Bin Packing Problem (BPP), the problem can be defined as follows: the decision entities encompass fixed-sized containers represented by $B_i = (L, W, H)$ and a set of items denoted as $N = \{d_1, d_2, \cdots, d_n\}$. *B* and *N* denote the container and item entities, respectively, whereas *L*, *W*, and *H* denote the length, width, and height of the container. Each item, $d_n = (l_n, w_n, h_n)$, represents the length, width, and height of the respective item. On the optimization front, the exact formulation can be formulated as follows.

$$\begin{aligned} &\max_x u \\ &\text{s.t.} \sum_{i=1}^{N} v_i(d_i) \le L * W * H \\ &l_i \le L, i \in N \\ &w_i \le W, i \in N \\ &h_i \le H, i \in N \end{aligned} \tag{1}$$

The optimization objective is represented by the formula: $u = s\sum_{i=1}^{N} v_i(d_i) / v_i(B) * 100\%$ where u represents the optimization objective, $v(.)$ is the size measure function, and *N* is the set of items. The primary objective is to optimize the overall utilization of a pre-determined container *B*. However, this problem is subject to several constraints. These include ensuring that items are not placed in such a way that they cause overlap or extend beyond the boundaries of the container. In addition, the placement of items must adhere to specific stability rules, which are outlined as follows in the paper: a) At least 60% of the bottom area of item n should be supported by existing items at all four corners. b) At least 80% of the bottom area of item n should be supported, and three out of the four corners should have support. c) At least 95% of the bottom area of item n should have support. These stability rules ensure that items do not overlap or are placed outside the boundaries of the container, and that the placement of items is stable. The specific stability description is illustrated in Figure 2.

3.1. Problem Statement and Formulation

Our approach is described in the form of a Markov Decision Process (MDP), which can be represented as (S, A, P, R, γ). The set of states S is utilized to describe the current environment by means of signals. The action set A consists of feasible actions that can be taken at the current state. The transition probability $P: S \times A \times S \rightarrow [0,1]$ determines the likelihood of transitioning from one state to another when action a is taken. Since our approach is based on model-free reinforcement learning, there is no need to explicitly represent the transition probability P. The reward set $R: S \times A \rightarrow \mathbb{R}$ provides the reward signal from the environment. The discount factor γ adjusts the impact of future rewards on the present value. It influences the extent to which the agent considers future rewards, with values closer to 0 emphasizing immediate rewards and values closer to 1 giving more weight to long-term rewards. In our task, γ is set to 1, allowing us to fully utilize future reward information. The policy $\pi: S \rightarrow A$ represents the mapping from states to action probabilities, where $\pi(a \mid s)$ denotes the probability of taking action a in state s. In Deep Reinforcement Learning (DRL), our objective is to find a policy π that maximizes the cumulative discounted reward, denoted as $J(x) = E_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$. Here, $\tau = (s_0, a_0, s_1, \cdots)$ represents a trajectory sampled according to policy π.

- **Environment State:** In the context of the online 3D Bin Packing Problem (3D-BPP), a comprehensive state encompasses both the current information regarding the items that need to be packed and the configuration of the container. It captures a complete snapshot of the problem, including features of the items and permutations within the bins. For tractability, the bottom space

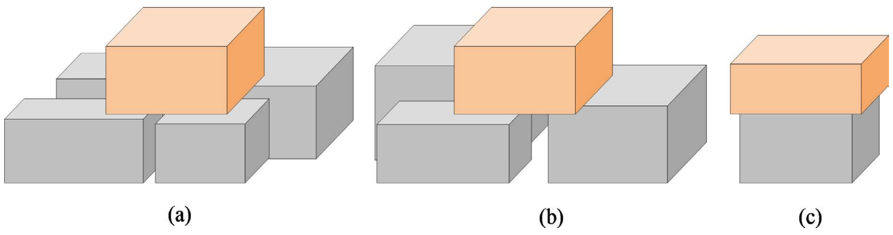

Figure 2. Box physical stability rules.

of the container is discretized into a grid of gated cells, where each cell contains a numerical value representing the current stacking height of the item. To represent the stacking height map in the online 3D bin packing problem, we employ a 2D matrix denoted as H_n, which has dimensions $L \times W$. Here, L denotes the length of the container and W denotes its width. The matrix H_n serves as a visual characterization of the height at each location within the container, illustrating the stacked items and their respective heights. Furthermore, the item currently being placed is represented as a three-dimensional vector $d_n = [l_n, w_n, h_n]^T \in \mathbb{Z}^3$, where l_n, w_n, and h_n denote the length, width, and height of item n, respectively. Consequently, the current state can be represented as $s_n = \{H_n, d_n, d_{n+1}, \cdots, d_{n+k-1}\}$, where k indicates the number of items that still need to be placed in the current state.

- **Action Space:** Based on the extensive conventions in previous literature, we define the packing action as the process in which the agent places the Front-Left-Bottom (FLB) corner of the item to be packed onto a discretized grid point at the bottom of the container. Therefore, for item n, the action a_n is defined as $a_n = Lx_n + y_n$, where L represents the length of the container, and x_n and y_n denote the coordinates of the target grid point. The action space A is defined as $A = \{0, 1, \cdots, L \times W - 1\}$, encompassing all the possible grid points in the container's bottom.
- **Reward:** We employ a binary reward system in our reinforcement learning approach to facilitate the agent's learning. Based on prior literature [3], we define an extrinsic reward as the volume of the packed item, while an intrinsic reward complements the extrinsic reward and is designed based on the evaluation of the packing strategy. Hence, our reward function is defined as $R_1 = 10 \times l_n \times w_n \times h_n / L \times W \times H$., where R_1 represents the incremental reward. At each step of the packing process, the agent is rewarded based on the volume of the item that is packed, with the reward being equivalent to the item's volume. If the agent fails to successfully place the item, it receives zero reward and the packing attempt ends immediately. In addition, we introduce a final-step reward, $R_2 = B_{ratio} \times 10$, where B_{ratio} denotes the utilization rate of the container. After successfully placing all items into the container, the agent receives a reward proportional to the utilization rate, whereas during incremental item placement, no reward is provided. With this reward design, we provide the agent with a dual perspective for solving the online 3D bin packing problem: a terminal perspective that evaluates the final result, and an incremental perspective that measures the packing process. This approach is designed to prevent the agent from getting trapped in the search for the black hole during the problem-solving process, ensuring that the packet planning remains on track.

3.2. Dual-Value Stratification Objectives

Commonly, the packing task can be divided into two stages: first, the generation of new states where box sizes and container height maps are determined, and second, the execution of strategies by the agent on the new states while satisfying real-world constraints [11]. The estimation of the value is affected differently in the two phases. In the first stage, object generation is stochastic, which introduces uncertainty into the volume-oriented reward design. This uncertainty increases the variability in the estimation of values for the same state or action, and as a result. In the second stage, practical considerations lead us to adopt hard constraints (based on a few simple rules) to ensure the legality of agent actions. However, these hard constraints often affect the agent's policy, further exacerbating the inaccuracy of the value estimates. Thus, the two stages of binning introduce different levels of variance in the value estimation.

Based on the heuristic reward configuration, we formulate our problem, where the reward of the agent depends primarily on the size of the current object. Additionally, the lower bound of rewards that the agent can obtain is determined by the arrangement of boxes within the container (as a failed packing attempt results in a reward of 0 and immediate termination of the current packing round). According to the general definition of reinforcement learning, the value, as expressed in Equation (2), represents the sum of discounted rewards that may be obtained in the future. In our problem, the discount factor γ is set to one to better take into account the long-term reward information. Due to the specific nature of the reward design, the prediction of the value is transformed into predicting the volume of future generated objects. Given the inherent uncertainty in the object generation process in online 3D binning, the estimation of the value is subject to variance in the value prediction. This variance has a significant impact on the value prediction, resulting in significant fluctuations in the predicted value of the value network for the same state or action. As a result, policy optimization becomes more challenging. Therefore, it is necessary to reduce the variance in the value estimate to improve the accuracy of the value prediction.

To improve the effectiveness of policy optimization and reduce the variance in value estimation due to reward uncertainty, we design an intrinsic value network based on bagged policy evaluation. The intrinsic value network and the original value network estimate the value based on the rewards r_2 and r_1 respectively, and the value is re-evaluated by a weighted sum of the two. The intrinsic reward r_2 primarily addresses the problem of non-computable future rewards under reward r_1. In order to make the long-term reward stable and controllable, it is designed as an evaluation of the current arrangement of boxes within the container. When the reward r_1 exhibits high variance in future estimations, the determinism of the intrinsic reward r_2 compensates for the variance in the current value estimate. Ultimately, the weighted sum of the intrinsic value estimate and the original value estimate guarantees a stable and controllable current value estimate. Moreover, since the design of r_2 is based on terminal rewards, this means that the environment only provides reward feedback based on the packing results after the packing is completed. Consequently, when training the agent with a single reward r_2, it encounters another common problem in reinforcement learning: sparse rewards. In such cases, the agent struggles to obtain sufficient information to learn the policy. To address this, the value network with intrinsic reward r_2 shares the network backbone with the value network with intrinsic reward r_1, differing only in the output layer. This helps facilitate the learning of the intrinsic value network under the sparse reward r_2 condition.

$$V(s) = E\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots \mid S_t = s\right] \tag{2}$$

3.3. Network Architecture and Training Configurations

According to previous literature, the ACKTR (Actor-Critic using Kronecker-Factored Trust Region) algorithm demonstrates significant advantages over other reinforcement learning algorithms in the context of online 3D packing problems [3]. Our model is designed based on the ACKTR framework. To address the high variance in value estimation due to reward uncertainty, we introduce intrinsic rewards based on the packing strategy. Different rewards train separate state-value networks that generate the value function. The results of these networks are combined through weighted summation to obtain the final value predictions, which guide the actor in learning a policy network to assist the agent in placing items at each Load Point (LP). ACKTR is an advanced online model-free reinforcement learning technique that builds upon the A2C (Advantage Actor-Critic) algorithm. It incorporates enhancements to improve sampling efficiency, such as utilizing natural gradient descent instead of stochastic gradient descent, effectively addressing the scale arbitrariness prevalent within the parameter space. Furthermore, ACKTR introduces a substitution for the conjugate gradient component found in TRPO (Trust Region Policy Optimization) by employing K-FAC (Kronecker-Factored Approximated Curvature). This substitution optimizes the storage and inversion operations associated with the Fisher matrix, resulting in improved performance [24].

- **State input:** To simplify the process, a Convolutional Neural Network (CNN) is employed to encode the initial state of the Bin Packing Problem (BPP). The box information, represented by d_n, is expanded into a three-channel tensor called $D_n \in \mathbb{Z}^{L\times W\times 3}$. Each d_n is a 2D matrix of dimensions $L\times W$, which captures the length, width, and height details of the respective box. Thus, the state s_n is a combination of the stacked height map H_n and the tensor D_n. The resulting state, $s_n = \{H_n, D_n\}$, is transformed into a 2D matrix with dimensions $L\times W\times 4$, as depicted in **Figure 3**.
- **Model structure: Figure 4** provides an illustrative representation of the comprehensive framework of the model. Our model contains three separate multi-layer perceptron modules. In addition to the actor and critic networks under the actor-critic framework, an intrinsic reward-based intrinsic critic network is also introduced. To help train the intrinsic critic network and critic network to share the same network architecture, only the output layers of the two networks differ. The estimate of the value is determined by the weighted prediction value of the intrinsic-critic and critic networks. The

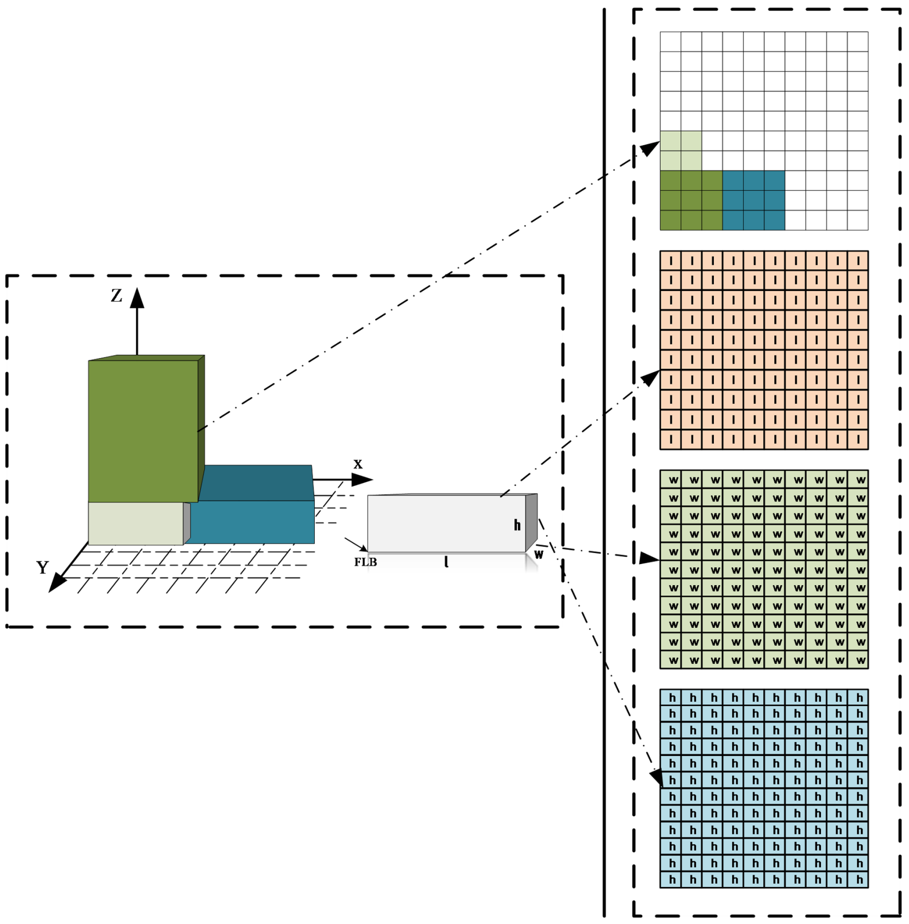

Figure 3. An indication of the status during the packing process.

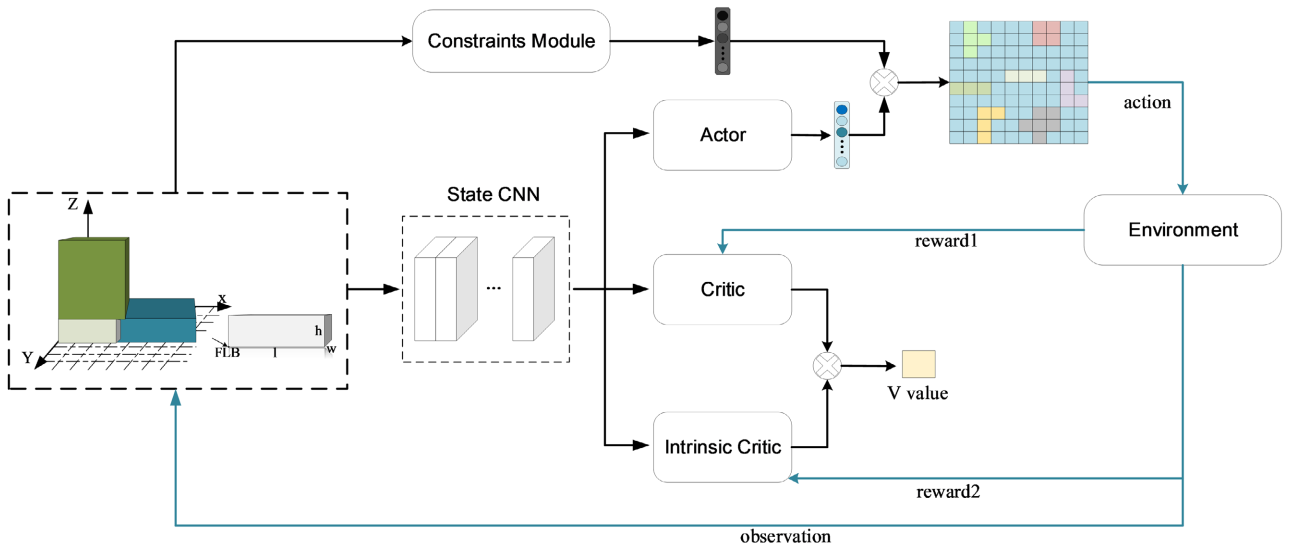

Figure 4. The packaging model structure under the guidance of double value.

actor computes the action vector $\pi\left(a_t \mid s_t\right)$, which is guided by the value to update its strategy. Under the hard constraint, we use action masking to project the policy onto the legitimate action space, ensuring that the agent samples actions from a valid space. The training strategy for the actor network parameters is based on the policy gradient formula shown in Equation (3), where θ_{actor} represents the adjustable parameters for different actors. The discount factor γ is set to 1, while the hyperparameters σ and ρ are set to 0.7 and 0.3, respectively.

$$\begin{aligned}&\nabla\theta_{actor}\\&=\left(\sigma\cdot\left(r_n^1+\gamma V_1(s_{n+1})-V_1(s_n)\right)+\rho\cdot\left(r_n^2+\gamma V_2(s_{n+1})-V_2(s_n)\right)\right)\nabla\log P_{actor}(a_n|s_n)\end{aligned} \tag{3}$$

- **Loss function:** Our model is trained using a composite loss, defined by Equation (4) and specifically designed as Equation (5), the loss function L_{actor} serves as the actor network's loss function to guide the network's update of policy parameters. This loss consists of two components: a policy objective function and an entropy regularization term. The policy objective function measures the performance of the current policy, which is the expected return obtained under the current policy. We design the loss based on the actor-critic framework, so that the policy objective function exists as an advantage function. The entropy regularization term can improve the exploratory nature of the policy and prevent the policy from getting stuck in local optima. L_{critic} and $L_{intrinsic\text{-}critic}$ evaluate the returns of taking actions under rewards r_1 and r_2, respectively. The loss of the value network is usually in the form of MSE to make the estimated value function as close as possible to the true value, thus better guiding the actor network to output the packing policy. Moreover, $E_{entropy}$ is the entropy loss function used to penalize the stochasticity of the policy, which allows the distribution of the output packing policy of the actor network to spread more widely and avoid getting stuck in local optima due to insufficient exploration. E_{inf} starts to avoid illegal actions from the generation of policy network parameters based on the computed action masks, that is, it tries to penalize illegal actions as much as possible during the network generation of the policy.

$$L=\alpha\cdot L_{actor}+\beta\cdot L_{critic}+\delta\cdot L_{intrinsic\text{-}critic}+\omega\cdot E_{inf}+\varphi\cdot E_{entropy} \tag{4}$$

$$\begin{cases}L_{actor}=\left(\sigma\cdot\left(r_n^1+\gamma V_1(s_{n+1})-V_1(s_n)\right)+\rho\cdot\left(r_n^2+\gamma V_2(s_{n+1})-V_2(s_n)\right)\right)\log P_{actor}(a_n|s_n)\\ L_{critic}=\left(r_n^1+\gamma V_1(s_{n+1})-V_1(s_n)\right)^2\\ L_{intrinsic\text{-}critic}=\left(r_n^2+\gamma V_2(s_{n+1})-V_2(s_n)\right)^2\\ E_{inf}=\sum\limits_{M_{n,o}(x,y)=0}P_{actor}(a_n|s_n)\\ E_{entropy}=\sum\limits_{M_{n,o}(x,y)=1}-P_{actor}(a_n|s_n)\cdot\log\left(P_{actor}(a_n|s_n)\right)\end{cases} \tag{5}$$

4. Experiments

We conducted our experiments on a desktop computer running Windows 10, equipped with a 12th Gen Intel(R) Core(TM) i9-12900KF CPU @ 3.20 GHz, 64 GB of RAM, and an Nvidia RTX A6000 GPU with 48 GB of memory. The code is written in Python and all network models are implemented using PyTorch. The spatial resolution of the bottom of the disputed container was set to 10 × 10.

4.1. Training and Test Set

Regarding the dataset, we followed the settings of [14] and set the length, width, and height of the container to $L = W = H = 10$. The item information is generated via three different settings. To avoid overly simplified scenarios, the size range of items is defined as follows: $2 \le l_i \le L/2$, $2 \le w_i \le W/2$, $2 \le h_i \le H/2$. The concrete form is shown in Figure 5. In the initial setting, RS (Random

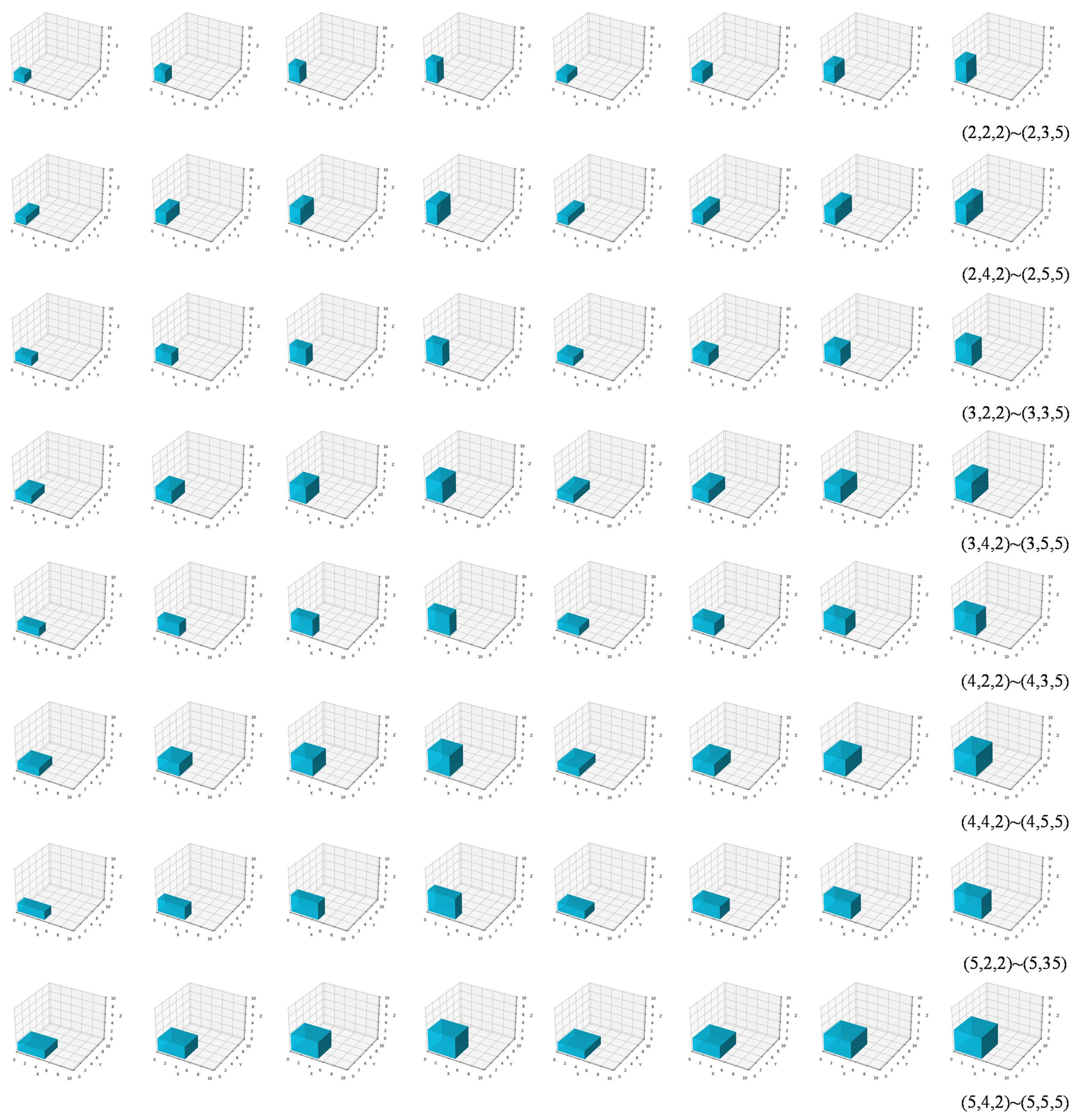

Figure 5. Predefined project visualization.

Sequence), the bins are combined in a random fashion, which poses a challenge in evaluating the algorithm's performance due to the uncertainty surrounding the optimal placement order. However, in some logistics centers and transfer stations, such randomly generated, unknown-order situations are more common and realistic. Although it is difficult to accurately measure the algorithm's learning performance relative to the optimal solution in this dataset, if the algorithm wins in a comparison of general performance (such as comparing their average number of wins in a packing game), it demonstrates its practicality. From an engineering point of view, the generality and practicality of the algorithm are particularly important for such practical scenarios. In the second data collection method, CUT-1, we obtain the data set by cutting a complete container. Within the scope of the set, the bins can be divided into several sub-boxes, and the results are sorted from top to bottom based on the Z-coordinate of the FLB of each sub-box, forming the CUT-1 dataset. In the third cut method, CUT-2, boxes are sorted according to their stacking dependence after the cut, and a box is added to the sequence only if all of its support items are present. In both the second and third methods, we can easily obtain the optimal placement order, so the performance of the algorithm is mainly focused on these two methods.

4.2. Performance Comparison

Table 1 shows the performance comparison across the three datasets, where our performance on the RS dataset is close to that of the offline loading scenario. Figure 6 shows partial visual results of our model on three datasets. Figure 7 shows the distribution of the three models on 100 test cases. It can be observed that our model performs best in obtaining the optimal policy for the CUT-1 dataset, achieving the highest box recovery rate for the cut boxes. On the CUT-2 dataset, our model demonstrates superior performance in terms of overall placement utilization and the number of items placed. On the RS dataset, our model maintains an equilibrium performance. Similar to [3], we also evaluate the impact of BPP-K on performance. BPP-K involves the addition of one or more sensing devices to the pre-placed boxes in a forward-looking manner without affecting the placement sequence, with the goal of finding the optimal placement for the projected item. The performance of BPP-K in our model is depicted

Table 1. Presents the performance comparison of our method on the three datasets.

Method	Items/% Space Uti.		
	RS	CUT-1	CUT-2
Boundary Rule (Online)	8.7/34.9%	10.8/41.2%	11.1/40.8%
BPH (Online)	8.7/35.4%	13.5/51.9%	13.1/49.2%
LBP (Offline)	12.9/54.7%	14.9/59.1%	15.2/59.5%
Zhao *et al.* (Online)	12.2/50.5%	19.1/73.4%	17.5/66.9%
Our (Online)	13.2/54.4%	19.2/74.0%	18.3/69.7%

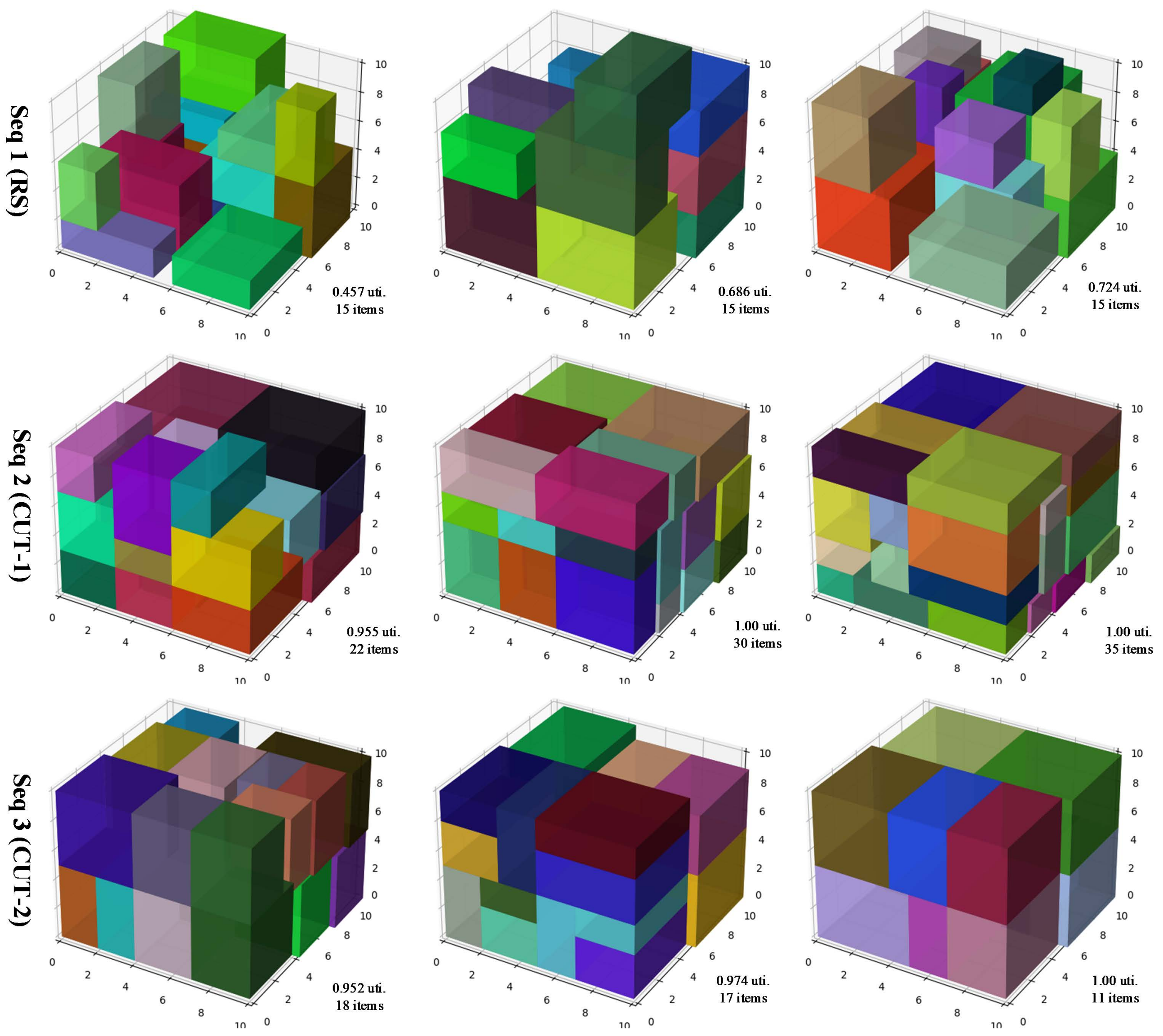

Figure 6. Visualization of palletizing results.

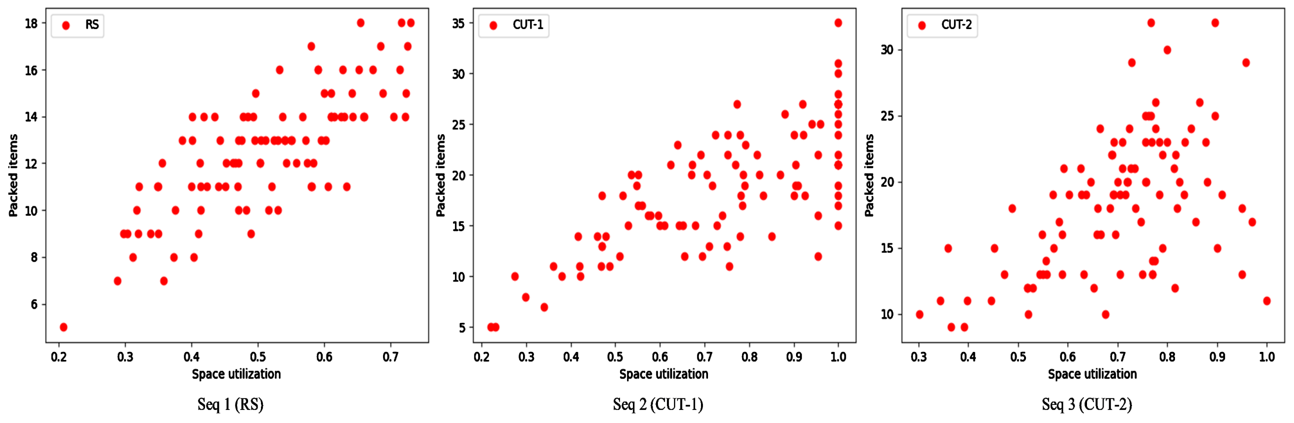

Figure 7. Distribution map of test cases.

in Figure 8.

In our model testing, we investigated the impact of box rotation on performance, and the results on the three datasets are presented in Table 2. For computational efficiency, we considered only two rotation angles: 1) vertical angle and 2) horizontal angle. In reality, each box can have six rotation directions, namely: direction 1 (l, w, h), direction 2 (w, l, h), direction 3 (h, w, l), direction 4 (l, h, w), direction 5 (w, h, l), and direction 6 (h, l, w), where l, w, and h represent the length, width, and height of the box, respectively. Increasing the number of rotation angles poses challenges in maintaining the action mask representation. As the resolution of the action space increases, the storage and computation requirements rise dramatically. Considering that our stability estimation method still relies on the original ergodic approach, introducing all six rotation angles makes the model more difficult to solve and results in a decrease in accuracy. Thus, only two rotation angles are considered in this study. In the future, we will further investigate the effect of all six rotation angles on the model performance by optimizing the stability estimation method.

5. Conclusion

We design an online 3D packing model that is described as a sequential decision process and solved using deep reinforcement learning methods. To meet real-world requirements, we introduce sequential dependencies and physical stability, and impose hard constraints on the action space in the form of action masks. We develop a two-valued network model based on the actor-critic framework, which includes both primitive and intrinsic value networks. We find that the problem of high variance in value estimation due to reward uncertainty can be corrected by an intrinsic reward mechanism, enabling the network to realistically perform long-term prediction and planning. To accelerate learning, we share the network heads of the original and intrinsic value networks. In the future,

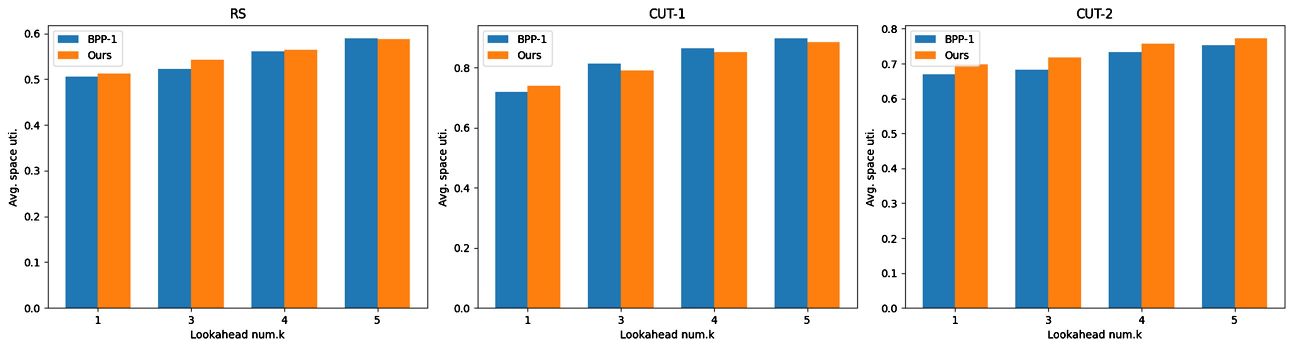

Figure 8. BPP-K performance graph.

Table 2. Rotation performance comparison.

Rotation	RS	CUT-1	CUT-2
w Orientation	13.2/54.4%	18.6/74.0%	18.3/69.7%
w/o Orientation	15.2/62.3%	18.7/75.9%	18.9/72.5%

we would like to carry out more studies on this issue. For example, more box rotation angles are introduced to improve packing efficiency; to further improve the performance of the model, it is important to explore the reasons why the agent autonomously discriminates and causes high variance in the value estimates.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Li, W. and Wang, S.Q. (2021) Research Status and Prospect of 3D Packing Problem. *Computer Knowledge and Technology*, **17**, 204-205, 222.

[2] Zhu, Q., Li, X., Zhang, Z., *et al.* (2021) Learning to Pack: A Data-Driven Tree Search Algorithm for Large-Scale 3D Bin Packing Problem. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, Gold Coast, 1-5 November 2021, 4393-4402. https://doi.org/10.1145/3459637.3481933

[3] Zhao, H., She, Q., Zhu, C., *et al.* (2021) Online 3D Bin Packing with Constrained Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35, 741-749. https://doi.org/10.1609/aaai.v35i1.16155

[4] Huang, S. and Ontañón, S. (2020) Action Guidance: Getting the Best of Sparse Rewards and Shaped Rewards for Real-Time Strategy Games. ArXiv: 2010.03956.

[5] Huang, Y., Lai, L., Li, W., *et al.* (2022) A Differential Evolution Algorithm with Ternary Search Tree for Solving the Three-Dimensional Packing Problem. *Information Sciences*, **606**, 440-452. https://doi.org/10.1016/j.ins.2022.05.063

[6] Ntanjana, A. (2018) Two and Three-Dimensional Bin Packing Problems: An Efficient Implementation of Evolutionary Algorithms. Master's Thesis, Durban University of Technology, Durban, 24-60.

[7] Zhao, C., Jiang, L. and Teo, K.L. (2020) A Hybrid Chaos Firefly Algorithm for Three-Dimensional Irregular Packing Problem. *Journal of Industrial & Management Optimization*, **16**, 409-429. https://doi.org/10.3934/jimo.2018160

[8] Zhang, L., Li, D., Jia, S. and Shao, H.B. (2022) Brain-Inspired Experience Reinforcement Model for Bin Packing in Varying Environments. *IEEE Transactions on Neural Networks and Learning Systems*, **33**, 2168-2180. https://doi.org/10.1109/TNNLS.2022.3144515

[9] Zhao, H., Yu, Y. and Xu, K. (2021) Learning Efficient Online 3D Bin Packing on Packing Configuration Trees. *International Conference on Learning Representations*, Vienna, 3-7 May 2021, 1-18.

[10] Yuan, J., Zhang, J. and Yan, J. (2022) Towards Solving Industrial Sequential Decision-Making Tasks under Near-Predictable Dynamics via Reinforcement Learning: An Implicit Corrective Value Estimation Approach. https://openreview.net/forum?id=UawwAryavZI

[11] Zhang, J., Zi, B. and Ge, X. (2021) Attend2Pack: Bin Packing through Deep Reinforcement Learning with Attention. ArXiv: 2107.04333.

[12] Verma, R., Singhal, A., Khadilkar, H., *et al.* (2020) A Generalized Reinforcement Learning Algorithm for Online 3D Bin-Packing. ArXiv: 2007.00463.

[13] Li, D., Ren, C., Gu, Z., *et al.* (2020) Solving Packing Problems by Conditional Query Learning. https://openreview.net/forum?id=BkgTwRNtPB

[14] Kaelbling, L.P., Littman, M.L. and Moore, A.W. (1996) Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, **4**, 237-285. https://doi.org/10.1613/jair.301

[15] Mnih, V., Kavukcuoglu, K., Silver, D., *et al.* (2015) Human-Level Control through Deep Reinforcement Learning. *Nature*, **518**, 529-533. https://doi.org/10.1038/nature14236

[16] Van Hasselt, H., Guez, A. and Silver, D. (2016) Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30, 2094-2100. https://doi.org/10.1609/aaai.v30i1.10295

[17] Silver, D., Huang, A., Maddison, C.J., *et al.* (2016) Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, **529**, 484-489. https://doi.org/10.1038/nature16961

[18] Silver, D., Hubert, T., Schrittwieser, J., *et al.* (2018) A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play. *Science*, **362**, 1140-1144. https://doi.org/10.1126/science.aar6404

[19] Mazyavkina, N., Sviridov, S., Ivanov, S. and Burnaev, E. (2021) Reinforcement Learning for Combinatorial Optimization: A Survey. *Computers & Operations Research*, **134**, Article ID: 105400. https://doi.org/10.1016/j.cor.2021.105400

[20] Festa, P. (2014) A Brief Introduction to Exact, Approximation, and Heuristic Algorithms for Solving Hard Combinatorial Optimization Problems. 2014 16*th International Conference on Transparent Optical Networks* (*ICTON*), Graz, 6-10 July 2014, 1-20. https://doi.org/10.1109/ICTON.2014.6876285

[21] Vinyals, O., Fortunato, M. and Jaitly, N. (2015) Pointer Networks. 2015 *Advances in Neural Information Processing Systems*, Montreal, 7-12 December 2015, 2692-2700.

[22] Bello, I., Pham, H., Le, Q.V., Norouzi, M. and Bengio, S. (2016) Neural Combinatorial Optimization with Reinforcement Learning. ArXiv: 1611.09940.

[23] Achiam, J. and Sastry, S. (2017) Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning. ArXiv: 1703.01732.

[24] Wu, Y., Mansimov, E., Grosse, R.B., *et al.* (2017) Scalable Trust-Region Method for Deep Reinforcement Learning Using Kronecker-Factored Approximation. 2017 *Advances in Neural Information Processing Systems*, Long Beach, 4-9 December 2017, 5285-5294.