上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Machine Learning for Bin Packing Problem

522031910652　许邦锐
522031910355　姬智昊
522031910030　陈升恒

2025 年 1 月 4 日

1 Introduction

The bin packing problem (BPP) is a well-known combinatorial optimization challenge that arises in various fields, including logistics, manufacturing, and computer science. The problem revolves around efficiently arranging objects of different sizes into a finite number of bins or containers, ensuring that no container is overfilled, and no items overlap. The primary objective is to minimize the number of bins used or, alternatively, maximize the utilization of available space within a fixed number of containers.

BPP has numerous real-world applications. In industries such as shipping and warehousing, it can determine how to load goods into trucks or storage units in a way that optimizes space while minimizing transportation costs. In manufacturing, it applies to cutting raw materials like steel or fabric to reduce waste. The problem is also prevalent in computer science, where it appears in memory allocation, data storage, and virtual machine placement in cloud computing environments.

Mathematically, BPP can be formulated as an NP-hard problem, indicating that there is no known polynomial-time algorithm that can solve all instances optimally. As the number of items and bins increases, the complexity of the problem grows exponentially, making exact solutions computationally prohibitive for large-scale cases. Consequently, heuristic, approximation, and metaheuristic algorithms are often employed to find near-optimal solutions within reasonable time frames. Techniques such as greedy algorithms, genetic algorithms, and simulated annealing are commonly used to tackle BPP.

Variants of the bin packing problem further expand its complexity and applicability. For instance, the 1D bin packing problem deals with packing items of varying lengths into linear bins, while the 2D and 3D bin packing problems involve packing objects into rectangular or cubic containers, reflecting real-world spatial constraints. Additional constraints such as item rotation, fragility, and weight distribution introduce even more practical relevance and challenge to the problem.

In this project, We focus on the 3D bin packing problem and attempt to obtain relatively optimal solutions by using a policy network, a search tree, and a combination of both.

2 Task1: Policy Network

In Task 1, we build a policy network that selects an object, its placement position, and rotation state based on the current container and object states.[1]

2.1 State Representation

The following describes how we model the states of containers and items.

For the container, we use a list with a size equal to its length and width, representing the top-down view of the container. Each position in this list corresponds to a 1x1 planar section. At each position, there is a 7-dimensional vector. The seven dimensions of this vector represent(Figure 1):

1.The current stacking height at this position.

2.The distance to the right where the height remains unchanged.

3.The distance to the left where the height remains unchanged.

4.The distance downward where the height remains unchanged.

5.The distance upward where the height remains unchanged.

6.The distance to the right to the next higher point.

7.The distance upward to the next higher point.

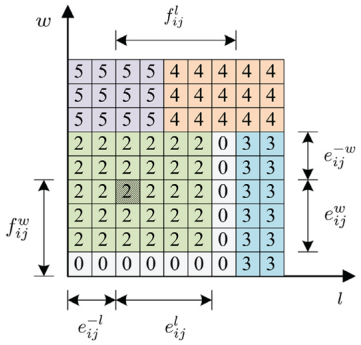


Figure 1 An example marking the value of plane features for a 9 × 9 container.

Each item is represented by a list of length 3, indicating its fixed length, width, and height. Multiple items form a larger list.

Using these two structures, we model the intuitive state of the current bin and items, as well as some hidden spatial relationships.

2.2 Policy Network

The policy network selects actions based on input:

1.Choose a position in the container.

2.Select an unpacked item.

3.Determine whether to rotate (there are six possible combinations of length, width, and height).

Our policy network uses a transformer architecture, with each of the three choices handled by the transformer. First, the modeling of the container and item states is encoded by the encoder(Figure 2).

In the position-selection decoder, the encoded description of the container state is used as the query (q), and the encoded item sequence is used as key (k) and value (v) to obtain the probability of each container position. Based on this probability, a position is selected, and the corresponding encoding for that position is retrieved.

In the item-selection decoder, the encoded item sequence is used as the query (q), and the encoding of the position obtained in the first step is used as key (k) and value (v) to compute the probability for each item. Based on this probability, an item is selected, and its six possible rotation states are generated, followed by encoding through the encoder.

In the final rotation-selection step, the encoded rotated item from the second step serves as the query (q), and the encoded position from the first step is used as key (k) and value (v) to compute the probability of each rotation. Based on this probability, a rotation is selected.

Ultimately, the selections from the three decoders are combined into a complete placement action, with the final probability calculated as the product of the three probabilities(Figure 3).

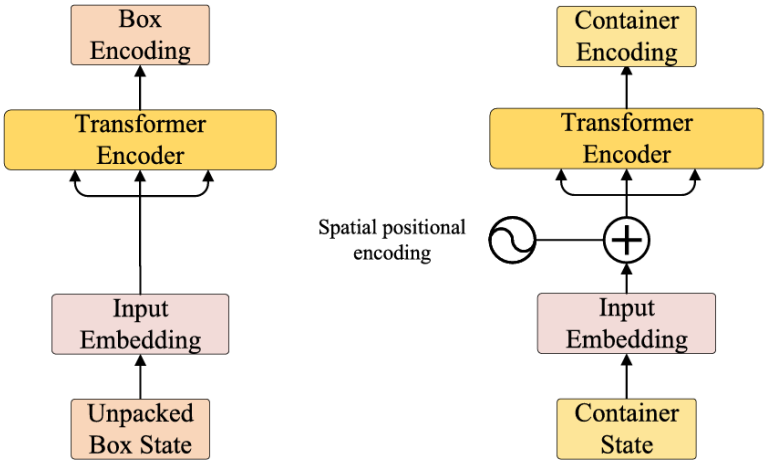
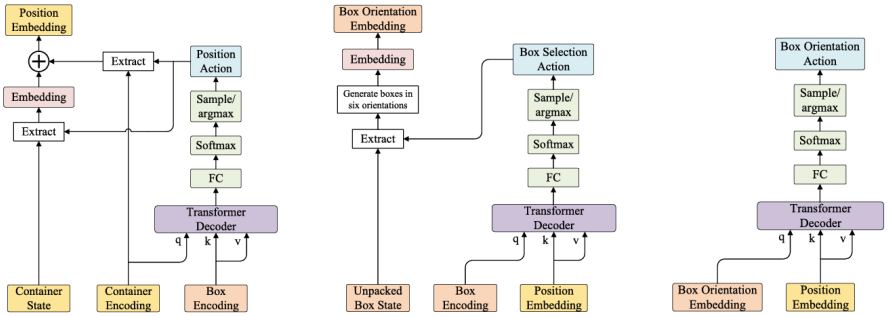


Figure 2 Two types of encoders



(a) Position decoder (b) Selection decoder (c) Orientation decoder

Figure 3 Three types of decoders

2.3 Training Method

After the policy network outputs its selected actions, the container state is updated according to the action, and the packed item is removed from the item list. The item is simulated as being placed into the container, and the new state is passed to the policy network for the next action.

The reward for each step is defined as:

Waste space in the current step - Waste space in the previous step.Where waste space = (maximum packing height) * (container length) * (container width) - (total volume of packed items).

During this process, the legality of the output actions must also be checked, ensuring that the item does not exceed the container’ s boundaries or overlap with other items. This check is done by calculating the three-dimensional lengths of the packed item and certain properties of the container state. If an error occurs, no update is made, and a penalty is applied. If the stacking height exceeds the container’ s capacity, the loop is terminated, marking the end of one training iteration.

2.4 Data Generation

Training data is generated by randomly slicing a large cube into smaller pieces, using Algorithm 1.

2.5 Loss Calculation and Parameters Update

In training, a reward is computed each time an item is placed. At the end of each training session, the rewards obtained are used to calculate the advantages through the Generalized Advantage Estimation (GAE) function, which are then used to compute the loss function.

The calculation of the loss function and backpropagation primarily follows the principles of the PPO algorithm, which consists of three components: policy loss, value loss, and entropy loss. First, the log probabilities of the current policy are calculated using the action probabilities output by the policy network, and the importance sampling ratio r(θ) is computed by comparing it with the old policy probabilities. This ratio, combined with a clipping mechanism, limits the magnitude of policy updates to ensure stability. The value loss is measured by the mean squared error (MSE) between the value network’s output and the target advantage values, optimizing the value network. The entropy loss is used to increase the randomness of the policy, en-

算法 1 Bin Packing Problem Generator

Initialize the items list $I \leftarrow \{(100, 100, 100)\}$
Sample N from $[10, 50]$
while $|I| < N$ **do**
    Pop an item randomly from I by the item' s volume
    Choose an axis randomly by the length of edge
    Choose a position randomly on the axis by the distance to the center of edge
    Split the item into two items
    To randomly rotate the newly generated objects
    Add them into I. The relative position in the container and the rotation can be saved as the label for training
end while
return I

hancing exploration. The total loss is obtained by combining these three components. After accumulating the loss across each time step, backpropagation is performed, and network parameters are optimized, gradually improving the performance of both the policy and value networks.

3 Task2: Searching Tree

We build a Monte Carlo Tree Search for the 3D packing problem. To be compatible with the policy network, the state representation, action contents, and legal action detection here are essentially the same as in Task 1.[2]

3.1 Basic Introduction

In the basic tree, we perform the main loop based on the current state, searching downwards from the current node of the tree. During the main loop, we iterate over all positions, all items, and all rotations, exhaustively enumerating all legal actions based on the current state and executing them. For each state resulting from these actions, multiple random simulation loops are conducted.

In each simulation loop, a currently legal action is randomly executed, and the depth is recorded until no more actions are available. Finally, based on the average depth of all subsequent simulation loops after each action is performed, we decide which action to execute at the current node of the main loop. This action is executed, recorded, and the process proceeds to the next node. When no legal actions remain in the main loop, we return the action history along the path down the tree to the current node, as well as the list of items that have not yet been packed.

3.2 Simplification and Integration with Policy Network

However, brute-force exhaustive search can result in significant time costs, so we introduce several simple methods for pruning.

3.2.1 Introducing Quantity Dimension

In addition to the original three-dimensional description of items, we add a fourth dimension representing quantity. This allows us to merge multiple identical items into one, reducing the number of items considered during exhaustive search. When updating the item state, we simply decrement the quantity dimension by 1. If the quantity reaches zero, the item is removed.

3.2.2 Initial Action Optimization

In the first step, we place the largest item in the bottom-left corner of the box. This aligns with the general human approach to bin packing and significantly reduces subsequent search iterations.

3.2.3 Simulation Action Optimization

During random simulation loops, when iterating over all positions to enumerate legal actions, we set a step size, avoiding all possible positions. Additionally, smaller items are prioritized.

3.2.4 Integration with Policy Network

In the later steps of the main loop, instead of exhaustively enumerating and executing legal actions, we use the policy network trained in task1. The network generates a certain number of actions, which are checked for legality and added to the list for execution.

In the end, we obtain a model that combines a search tree and a policy network for the final task.

4 Task3: E-commerce packaging problem

Finally, we address the e-commerce bin packing problem, which builds on task1 and task2 by considering multiple batches of orders and multiple box sizes to choose from

4.1 Information Processing

We read data from task3.csv to gather all orders. For each order, all items to be packed are organized into a two-dimensional list, where the first dimension represents all items, and the second dimension contains the item's length, width, height, and quantity.Since we assumed that the object dimensions are integers in the previous construction, we round up the object dimensions here. However, when calculating the volume of the packed object, we still use the original values.

If any item is found to be invalid during this process—meaning it is too large to fit into any available box—the order is discarded, and a corresponding warning message is output. If the item is valid, its volume is calculated and added to the total volume of items for that order.

4.2 Container Selection

Before using the previously developed model to perform packing, we need to select an appropriate box. Our strategy is to choose a container that has dimensions (length, width, height) larger than the largest remaining item, but with a volume that is closest to the current total volume of items to be packed.

The goal is to place an item in a box that is not excessively large, leaving the remaining space for smaller items. This prevents selecting a box that is too large without adequately filling it. This approach also aligns with the prioritization of smaller items during the simplified search tree simulation loop.

4.3 Iterative Packing

Once a suitable container is selected, the previously developed model is used to perform packing. If there are leftover items after the current packing, the process continues iteratively. A new container is selected based on the remaining items, and the model is invoked for packing again. After each container selection, its volume is calculated and added to the total volume of containers.

4.4 Computing Results

At the end of each packing iteration, all actions taken during that round are output for manual inspection. After completing the packing for an entire order, the ratio of total item volume to total container volume is calculated and output as a performance metric for the model.

5 Results and Analysis

5.1 Result of Task1

We set the PPO clipping range to epsilon = 0.2, the entropy regularization coefficient to beta = 0.01, and the value loss weight to c1 = 1.0. We then trained the model on the test set generated using the data generation algorithm. Below is a visualization of one of the results from the data generation algorithm(Figure 4).

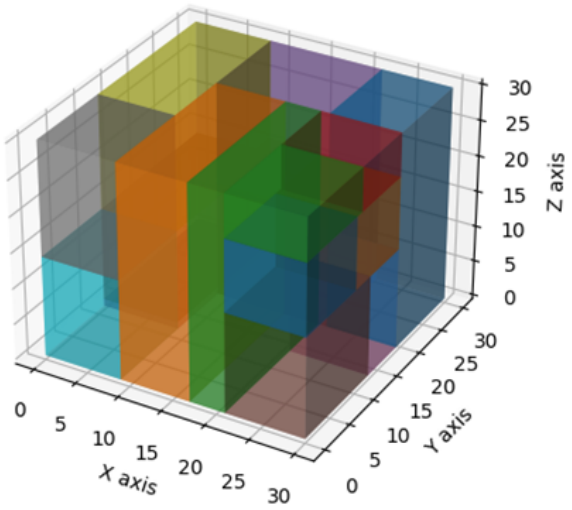


Figure 4 A example result of data generation

The model's output after completing a training set is shown in the figure below(Figure 5).

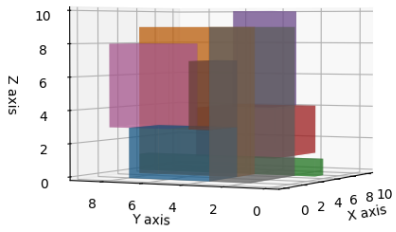


Figure 5 Result of task1

5.2 Result of Task2

We set the random search parameters with roll = 3, the step size for traversing positions as step = 5, and the number of attempts for the network to generate a valid action as try_count = 10. The packing results on the generated set are shown in the figure below(Figure6). Although the results are reasonably good, the loop complexity remains high before introducing the policy network to obtain valid actions, which makes it unsuitable for large-scale testing in Task 3.

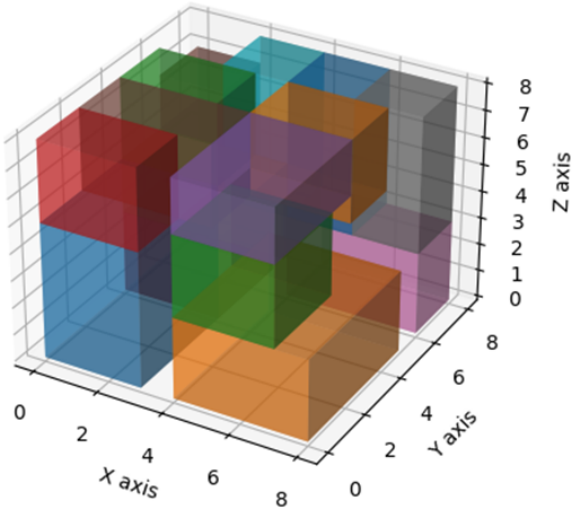


Figure 6 Result of task2

5.3 Result of Task3

We used the model trained in Task 1 and combined it with the search algorithm from Task 2 to test on the entire task3.csv. Ultimately, we achieved a ratio of 45.02%.

Next, we will conduct a case analysis on some specific results (with object lengths rounded up).

5.3.1 Satisfactory results

In order BSIN2309040417, the model selected a container (52, 40, 17) and executed the following actions: ['position': [0, 0, 0], 'item': [34, 22, 12, 1], 'rotation': 1, 'position': [0, 0, 12], 'item': array([50, 38, 2, 1]), 'rotation': 0, 'position': [0, 0, 14], 'item': array([26, 13, 1, 1]), 'rotation': 1]. The three items in the order were packed into a single box.

It can be observed that our box selection algorithm successfully chose a box that could accommodate all the items without being too large. After placing the largest item [34, 22, 12, 1], the remaining two items, [50, 38, 2, 1] and [26, 13, 1, 1], which had lower heights, could be stacked on top, fully utilizing the remaining space. Since the order contains an item with a side length of 50, and (52, 40, 17) is the smallest box that can fit it, this is the optimal solution.

In another order BSIN2309068936, the model first selected a container (37, 26, 13) and executed ['position': [0, 0, 0], 'item': [35, 24, 13, 1], 'rotation': 0]. Then, it selected another container (37, 26, 13) and executed ['position': [0, 0, 0], 'item': array([37, 26, 3, 1]), 'rotation': 0, 'position': [0, 0, 3], 'item': array([21, 15, 4, 1]), 'rotation': 1].

It can be observed that our model packed [35, 24, 13, 1] into (37, 26, 13), which is the smallest box that can fit it. After that, due to the remaining space not being enough to accommodate any other items, a second container (37, 26, 13) was chosen to pack the remaining two items. This container is also the smallest box that can fit [37, 26, 3, 1], so this represents a local optimal solution for both steps, and is likely the global optimal solution as well.

5.3.2 Unsatisfactory results

1.In order BSIN2309070818, the model selected four containers (40, 28, 16) and packed three items: [34, 21, 12, 3] and [30, 28, 5, 1]. In fact, to pack [34, 21, 12, 3], a (35, 23, 13) box would suffice. However, when selecting the box, the model considered that the width of [30, 28, 5, 1] is 28, so it chose a box with a width of 28. In reality, after packing [34, 21, 12, 3] into the (40, 28, 16) box, it was no longer able to fit [30, 28, 5, 1], resulting in wasted space in the box.

In order ODO1694210527514757, the model selected two containers (37, 26, 13) and packed [31, 22, 12, 1] and [37, 26, 4, 1]. Similarly, a (35, 23, 13) box would have been sufficient for packing [31, 22, 12, 1], but due to the influence of [37, 26, 4, 1], a larger box was chosen. Furthermore, if a slightly larger box (42, 30, 18) had been selected, both items could have been packed into a single box.

5.4 Analysis

The case analysis highlights the strengths and limitations of our model when applied to different packing scenarios. A clear pattern emerges when the items to be packed vary significantly in size—some being substantially larger while others are relatively small. In such situations, the model demonstrates an impressive ability to identify and select containers that efficiently accommodate the larger objects first. By doing so, it maximizes the use of available space by strategically arranging smaller items in the remaining gaps. This approach not only optimizes space utilization but also reflects the model' s capacity to handle heterogeneous

item distributions effectively, balancing the need for container efficiency with practical constraints.

Conversely, when the objects to be packed exhibit similar overall volumes but possess distinctive, prominent dimensions (e.g., elongated or irregularly shaped along one axis), the model's performance shows noticeable inefficiencies. In these cases, the model tends to select containers that are slightly larger than the immediate requirement, leading to scenarios where objects are packed individually rather than grouped efficiently. This one-by-one packing approach often results in underutilized container space, contributing to noticeable wastage. The inability to exploit the full capacity of the container stems from the model's preference for immediate feasibility over complex spatial optimization, which may lead to an accumulation of voids or inefficient use of available space.

A deeper examination of this behavior suggests that the root cause lies in the structural design of the model. To enhance computational efficiency and reduce operational time, the search scope within both the policy network and the search tree is intentionally limited. While this reduction in search iterations accelerates decision-making, it concurrently narrows the exploration of potentially superior configurations. Consequently, the model may settle on suboptimal solutions that could have been avoided if a broader, more exhaustive search process had been conducted.

Moreover, the model incorporates a degree of greediness in its container selection process, favoring solutions that satisfy immediate constraints rather than those that consider long-term spatial efficiency. This myopic approach can produce mixed outcomes. In certain scenarios, selecting the most straightforward solution can serendipitously align with the global optimum, resulting in efficient packing. However, in other cases, the same greedy strategy may lead to fragmented packing arrangements that diverge from the global optimal solution, ultimately diminishing overall performance.

In conclusion, while the model proves highly effective in scenarios involving heterogeneous object sizes, it faces notable challenges when dealing with similarly sized objects characterized by prominent dimensions. The trade-off between computational speed and packing efficiency underscores the need for refinement. Enhancing the model's capacity to balance exploration and exploitation—potentially through adaptive search strategies or hybrid approaches—could mitigate its tendency toward myopia, ensuring more consistent attainment of globally optimal solutions across diverse packing scenarios.

6 Possible Improvement

Based on the analysis of the results, we propose some areas for optimization。

6.1 Policy Network

Our policy network is trained on containers of fixed size and the objects cut from them. Due to time constraints, the training scale and batch size are limited. If the network could be trained more thoroughly based on the specific containers and objects in Task 3, it might be able to output more accurate and effective actions, thus helping the tree search make better choices.

6.2 Tree Search

Due to time constraints, we limited the exhaustive step size and random branching tree in the tree search. If these parameters could be increased, the search would be better at selecting the current action and improving myopic behavior.

6.3 Container Selection

Currently, our box selection is based solely on basic mathematical operations and comparisons, without considering the deeper mathematical relationships between the object dimensions and the container size. If a neural network could be trained for decision-making in this step, the performance might be improved.

6.4 State Representation

We use integer length division of the top-down view to represent the container state. However, when applying it to Task 3, we encountered issues with handling decimals. Additionally, the top-down view has limitations in representing vertical information. For example, if an object exists at position [5, 5, 5], the height information at [5, 5] is updated to 5, even if the space at [5, 5, 0:4] is empty and another object could still be placed there, leading to some inefficiency. This could be solved by subdividing the spatial blocks more finely and changing the state representation to three-dimensional, but this would require more time and space for data processing.

References

[1] Que, Quanqing, Fang Yang, and Defu Zhang. Solving 3d packing problem using trans former network and reinforcement learning. *Expert Systems with Applications*, 2023.

[2] Pejic, Igor, and Daan van den Berg. Monte carlo tree search on perfect rectangle packing problem instances. *Proceedings of the 2020 genetic and evolutionary computation conference companion*, 2020.

A The source code of our model

You can get the code of our model form https://github.com/Vegetable-bird10086/Machine-Learning-for-Bin-Packing-Problem.git.

B Contribution of each member

Name	Student ID	Score	Work
许邦锐	522031910652	40%	code train report
陈升恒	522031910030	30%	train code
姬智昊	522031910355	30%	train report