Machine Learning for Bin Packing Problem

1 Introduction

The Bin Packing Problem (BPP) is a well-known combinatorial optimization problem that seeks to pack a set of items with varying sizes into a finite number of bins or containers, such that the total number of bins used is minimized.

This problem arises in various fields such as logistics, manufacturing, resource allocation, and computer science, with applications ranging from cargo loading, cutting stock, and file storage, to memory allocation and VLSI circuit design.

The BPP is considered NP-hard, meaning that finding an exact solution becomes computationally infeasible for large instances, thus prompting the need for approximation and heuristic-based methods to solve real-world problems efficiently.

The 3D Bin Packing Problem (3D BPP) is a variant of the general BPP where the items to be packed and the containers are three-dimensional. Each item has specific dimensions in terms of length, width, and height, and the objective is to determine how to optimally place these items into containers while minimizing the total volume of the containers used.

The challenge lies not only in fitting the items within the containers but also in managing the three-dimensional aspect of the placement, as items can be rotated to fit more efficiently.

To address the 3D BPP, various methods are employed, each catering to different aspects of the problem. In this report, we explore policy network models, tree search algorithms and heuristic-based approaches to reach a near-optimal solution. These methods leverage machine learning techniques and human intuition to tackle the complexities of the 3D BPP and related optimization problems.

2 Policy learning for BPP problem

2.1 Policy Network Model

The policy network model is a type of reinforcement learning algorithm that learns a policy to map states to actions in order to maximize a reward signal.

Actor-critic algorithms are commonly used to train policy networks for combinatorial optimization problems like the BPP. These algorithms optimize the policy by updating the actor (policy) and critic (value function) networks based on the rewards received during training. In each iteration, the actor selects an action based on the state and the policy, and the environment provides a reward value that indicates how good the action was. The critic evaluates the quality of the action and provides feedback to the actor to improve its policy.

PPO (Proximal Policy Optimization) is a popular actor-critic algorithm that has been successfully applied to various combinatorial optimization problems, including the BPP. PPO uses a clipped surrogate objective to prevent large policy updates and stabilize training, making it suitable for problems with discrete action spaces like the BPP.

2.2 Method

2.2.1 Assumptions

To simplify the 3D BPP and make it more feasible for training a policy network model, we make the following assumptions:

- The items can only be placed orthogonally in the container, i.e., each item must be placed with its faces aligned to the faces of the container.
- Due to the previous assumption, the rotation of the item is limited to 6 possible orientations, which are (l, w, h), (l, h, w), (w, l, h), (w, h, l), (h, l, w), and (h, w, l).
- The items are placed in a single container with fixed length and width, but infinite height. The goal is to minimize the height of the stacked items in the container. With this assumption, the agent can always successfully pack all items into the container.

- The problem is discrete. That is, all dimensions of the items and the container are integers and the agent can only place the items at integer coordinates in the container.
- We use an encoder-decoder framework to encode the state of the environment and decode the action to take, treating the BPP as a sequence-to-sequence problem.

These assumptions helps to simplify the problem and make it more tractable for training the policy network model, while still capturing the essence of the 3D BPP.

2.2.2 Action Representation

The action space of the policy network model consists of three parts: the index of the item to pack, the rotation of the item, and the position to place the item in the container.

The index of the item is an integer ranging from 0 to the number of unpacked items, representing which item to pack next. The rotation of the item is an integer ranging from 0 to 5, as a 3D item has 6 possible rotations.

The position to place the item is a tuple of two integers representing the x and y coordinates of the item in the container. Here we do not use the z coordinate as the height of the item is can be inferred from the existing items in the container. The item will be automatically placed on the top of the stacked items.

Therefore the action space can be represented as $a = (a^s, a^r, a^p)$, where a^s is the index of the item, a^r is the rotation of the item, and a^p is the position to place the item.

This leads to a multi-discrete action space of size $n \times 6 \times L \times W$, where n is the number of items, L and W are the length and width dimensions of the container, respectively.

2.2.3 State Representation

The state, or observation space, of the policy network model consists of the state of the container and the state of the items. The state representation should provide enough information for the policy network

to make informed decisions about which action to take, while also use as few features as possible to reduce the complexity of the model. The design of the state representation is based on the given reference paper [1].

The state of the container is a height map with supplementary information, with each cell containing the following features:

- The height of the cell
- The distance to the nearest higher cell in positive length direction
- The distance to the nearest higher cell in positive width direction
- The distance to the edge of the cell plane in positive length direction
- The distance to the edge of the cell plane in negative length direction
- The distance to the edge of the cell plane in positive width direction
- The distance to the edge of the cell plane in negative width direction

These supplementary features, called “plane features”, are used to help the policy network understand the spatial relationship between the items and the container, as well as the constraints of the packing problem. For example, the distance to the nearest higher cell indicates how much space is available so that the agent knows whether an item can fit in that location more easily.

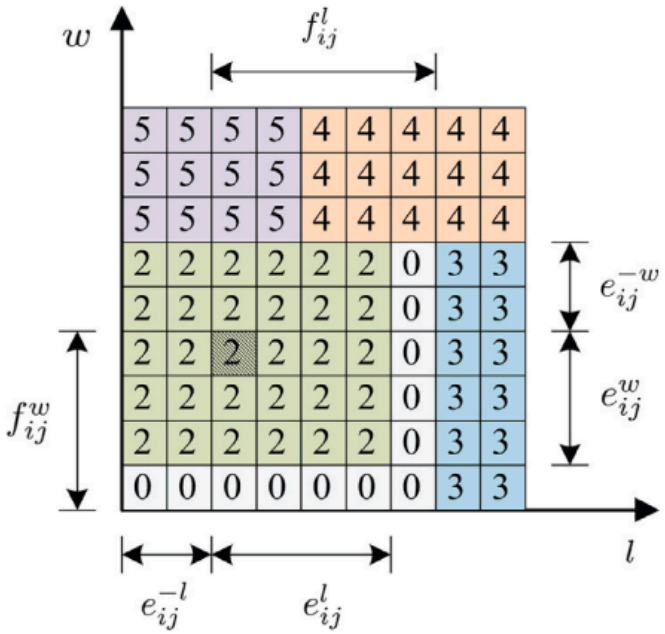

Figure 1: State representation of the container

Figure 1 from the paper [1] visualizes the state representation of the container. The supplementary features are respectively f_{ij}^{l}, f_{ij}^{w}, e_{ij}^{l}, e_{ij}^{-l}, e_{ij}^{w} and e_{ij}^{-w}. The numbers in the cell are the height of the cell, and the plane of a cell corresponds to the colored area in the figure.

The state of the items should a list of dimensions (l, w, h). A problem here is that the number of items may vary during the packing process, while the observation space of the policy network model should be of a fixed size.

To address this issue, we set a fixed number of items n that the agent can "see" in the observation space. If the number of items is less than n, padding mechanisms are used to fill the observation space with zeros. If the number of items exceeds n, the extra items are ignored. A mask is added to the observation space to help the agent distinguish between valid items and padding items, with 1 indicating a valid item and 0 indicating a padding item.

Therefore, the observation space has a total of $L \times W \times 7$ features for the container and $n \times 3$ features for the items, as well as a mask of size n. The three parts are packed into a dictionary and fed into the policy network model as the input.

2.2.4 Environment Design

A key component of training the policy network model is designing an environment that simulates the packing process and provides feedback on the quality of the packing. The environment should receive actions from the policy network, update the state based on the actions taken, and provide a reward signal to the policy network based on the quality of the packing.

In each step, the logic of the environment can be summarized as follows:

```
1: function Step(env, action)
2:     (index, rotation, position) ← decode(action)
3:     state ← env.state
4:     if index is invalid then
5:         return (state, reward: −1, done: False, info)
```

```
    if item does not fit in the container then
        return (state, reward: −1, done: False, info)
    items ← items - items[index]
    state ← encode(container, items)
    if items is empty then
        return (state, reward: 1, done: True, info)
    reward ← Calculate-Reward(container)
    return (state, reward, done: False, info)
```

The environment should provide a reward signal to the policy network based on the quality of the packing. The design of the reward function is crucial for training the policy network and guiding it towards the desired outcome.

Two common methods for calculating the reward are the height-based reward and the volume-based reward. The height-based reward penalizes the height of the container, while the volume-based reward penalizes the wasted space in the container. Here we use a combination of both rewards to encourage the policy network to increase the filling rate and reduce the height of the container. The formula for the reward is as follows:

$$reward = 1 + \frac{1}{V} \times \sum_{i=1}^{n} v_i - 0.1 \times H$$

where v_i is the volume of the i-th item, V is the volume of the container, and H is the height of the container.

2.3 Results and Discussion

The policy network model is trained using the PPO algorithm on the designed environment for a short period of time (< 10 minutes) to demonstrate the feasibility of the approach. The model fails to exhibit a good performance due to the limited training resources and time, but the results provide insights into the challenges and potential of training policy networks for the 3D BPP.

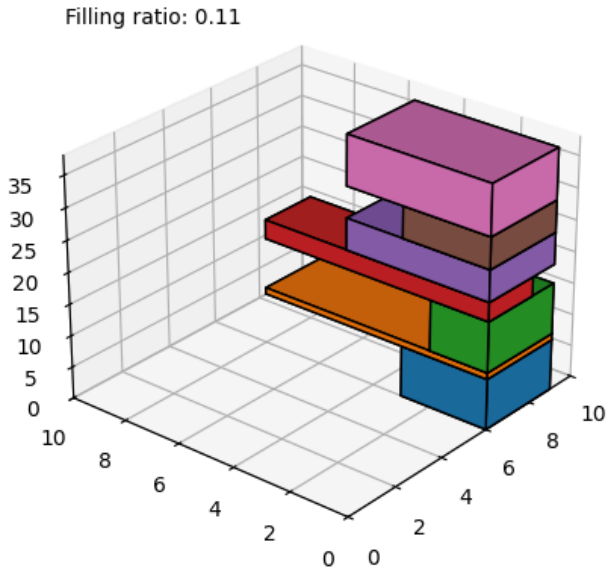

Figure 2: Packing result of the policy network model

In Figure 2, we can see that the model tries to rotate the items so that the height of the item is minimized, which is consistent with the goal to minimize total height. However, the model struggles to find a proper placement for the items, leading to suboptimal packing results. This highlights the challenges of training policy networks for combinatorial optimization problems like the 3D BPP, where the search space is vast and the optimal solution is hard to find.

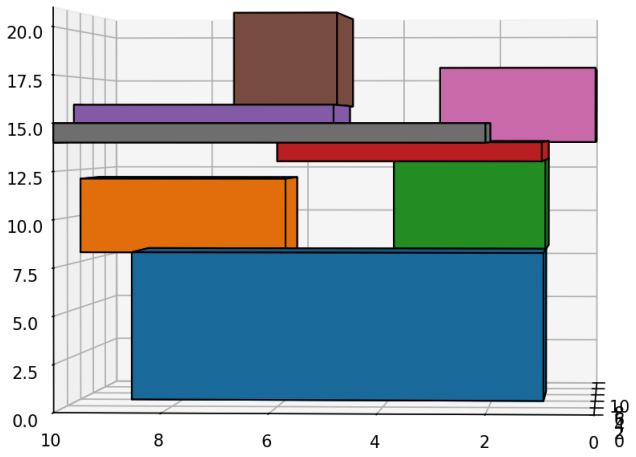

Figure 3: Packing result of the policy network model

With a little more training time, the model will start trying to locate the item more randomly, as is shown in Figure 3. This is because placing the item randomly has a higher chance of not increasing the overall height of the stacked items. Though this does lead to a better result, it is not a robust policy that can generalize to unseen data. The model needs more training time and resources to explore the search space more effectively and find a near-optimal solution.

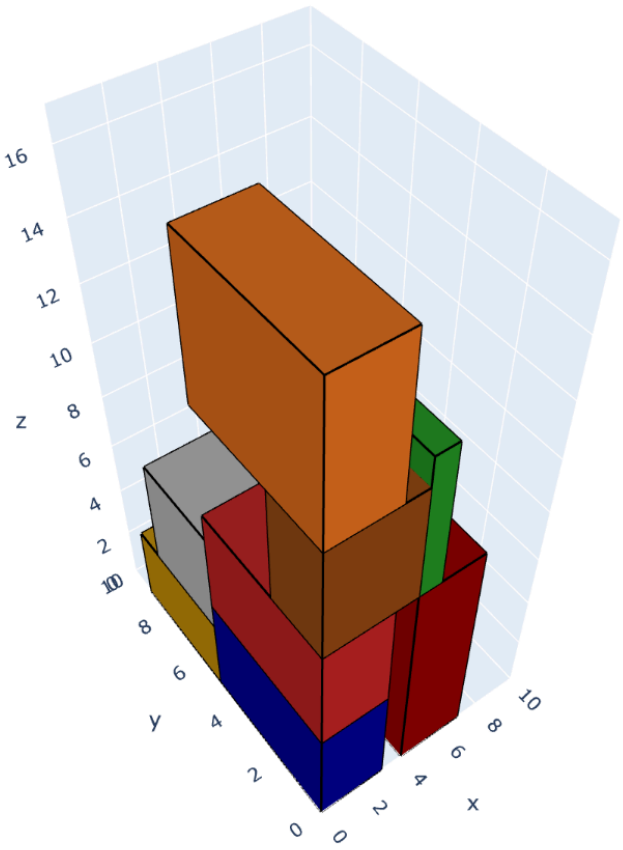

Figure 4: Packing result of the policy network model

Figure 4 shows the packing result of the policy network model with a different reward function. If we increase the penalty when the item exceeds the boundary of the container, the agent will choose a more conservative packing strategy. We can see that the model prefers to place the items in coordinate $(0, 0)$, because this is the safest position that will not exceed the container. This shows that the reward function plays a crucial role in shaping the behavior of the policy network and guiding it towards the desired outcome.

The results demonstrate the potential of policy network models for solving the 3D BPP, but also highlight the challenges of training such models effectively. The main difficulties lies in these aspects:

- A general PPO model may not be suitable for the 3D BPP, as the action space and the observation space are more complex than typical reinforcement learning problems.
- The reward function is crucial for training the policy network, and different reward functions can lead to very different packing strategies.
- The severe lack of training resources and time makes it difficult to explore the full potential of the policy network model.

3 Neural guided search algorithm

3.1 Tree Search Algorithm

Tree search algorithms are a class of algorithms that explore the search space by constructing a tree of possible states and actions, evaluating the quality of each state, and selecting the best action to take based on the evaluation. In the context of the BPP, tree search algorithms can be used to find an optimal or near-optimal solution by exhaustively searching through the space of possible packings.

However, it is impractical to perform an exhaustive search for large instances of the BPP due to the combinatorial explosion of the search space. Therefore, the policy network model can be used to guide the tree search algorithm by providing insights on which actions to explore and which solutions to prioritize. This approach, known as neural guided search, combines the strengths of tree search algorithms and policy networks to efficiently solve combinatorial optimization problems.

3.2 Method

A typical tree search algorithm for the BPP is the branch-and-bound algorithm, which recursively partitions the search space into branches and prunes unpromising branches based on lower bounds and upper bounds of the objective function. The algorithm explores the tree in a breadth-first manner, evaluating the quality of each state and selecting the best action to take at each step.

This method can be enhanced by incorporating a policy network model to guide the search process. The policy network can predict the best action to take at each step, providing a estimate of the quality of the state and guiding the search towards promising regions of the search space. By combining the tree search algorithm with the policy network model, the neural guided search algorithm can efficiently explore the search space and find high-quality solutions to the BPP.

As the policy network we trained in the previous section gives an actual action, instead of a estimate of the quality of the state, we can directly use the policy network to generate several optimal actions at each step to split the search space into several branches. This creates a breadth-first search tree, where each branch represents a possible packing solution. [2]

To restrict the search space, we can set a maximum width k for the tree search algorithm. After each iteration, only the best k environments are selected to continue the search, while the rest are pruned. This helps to focus the search on the most promising regions of the search space and avoid exploring unpromising branches.

Denote the number of actions to generate at each step as m. The neural guided search algorithm can be summarized as follows:

```
function NEURAL-GUIDED-SEARCH(initial-env, k, m)
    environments ← [initial-env]
    while environments are not done do
        new-environments ← []
        for env in environments do
            for i ∈ {1, 2, ..., m} do
                new-env ← copy of env
                action ← Policy-Network(new-env)
                new-env ← Step(new-env, action)
                Append new-env to new-environments
        environments ← Select-K-Best(new-environments, k)
    return Select-Best(environments)
```

3.3 Results and Discussion

Here we choose parameters $k = 5$ and $m = 2$ to demonstrate the neural guided search algorithm. The algorithm runs on the same environment as the policy network model, and the results are compared to evaluate the performance of the algorithm.

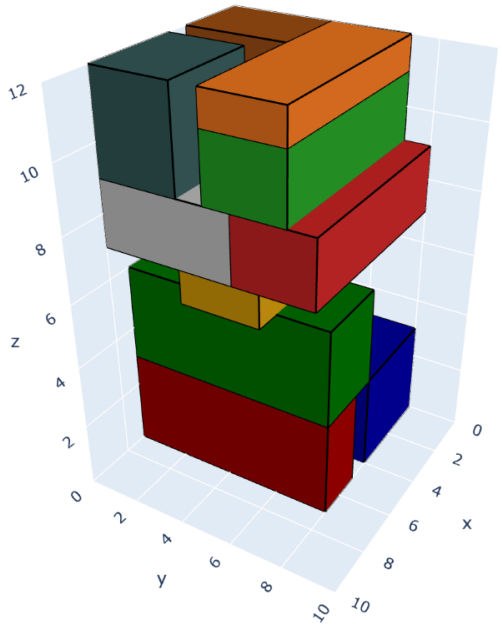

Figure 5: Packing result of the neural guided search algorithm

Even though the policy network we trained in the previous section has a poor performance, the neural guided search algorithm can still find a better packing result. This is because the tree search algorithm can explore a wider range of solutions. In each step, only the best environments can survive, increasing the chance of finding a better solution. However, the quality of the final result heavily depends on the initial actions.

The results demonstrate the potential of combining tree search algorithms with policy network models to solve combinatorial optimization problems like the 3D BPP. The neural guided search algorithm can effectively explore the search space and find near-optimal solutions by leveraging the strengths of both approaches.

4 E-commerce packaging problem

4.1 Problem Statement

The E-commerce packaging problem is a variant of the BPP that arises in the context of online retail and logistics. In this problem, the goal is to pack a set of items into a containers of different sizes, in order to minimize the total volume of the containers used and reduce the shipping costs. The challenge lies in wisely selecting which container size to use and how to choose items from a large inventory to maximize the packing efficiency.

The dataset `task3.csv` contains information about the items to be packed, including the `sta_code`, `sku_code`, `length`, `width`, `height`, and `quantity` of each item. The available container sizes are (35, 23, 13), (37, 26, 13), (38, 26, 13), (40, 28, 16), (42, 30, 18), (42, 30, 40), (52, 40, 17), (54, 45, 36). As the total volume of the items is fixed, the goal is to minimize the total volume of the containers used, thus maximizing the filling rate of the containers.

4.2 Method

4.2.1 Heuristic Approach

As the policy network model and the neural guided search algorithm do not have an ideal performance, we resort to a heuristic-based approach to solve the E-commerce packaging problem. The heuristic approach involves designing rules and strategies to pack items into containers efficiently, without the need for complex optimization algorithms.

In this case, we consider packing a chosen set of items into some chosen containers. The approach will first sort the items in descending order of volume, and then pack the items one by one into the containers based on the following rules:

- Try to add the item into the current container. If the item fits, add it to the container and go to the next item.
- After all items are tried to be added into the container, if there are still unpacked items, set the next container as the current container and go to the first step.
- If no containers is available, return all unpacked items.

The algorithm to pack the items into containers is adapted from a relevant paper [3]. The algorithm is as follows:

```
function Try-Add-Item-At-Position(container, item, position)
    for rotation in 6 possible rotations of the item do
        if No-Overlap(container, item, position, rotation) then
            item.position ← position
            item.rotation ← rotation
            Add item to container
            return True
    return False
```

```
function Try-Add-Item-To-Container(container, item)
    if container is empty then
        return Try-Add-Item-At-Position(container, item, (0, 0, 0))
    for axis = {x, y, z} do
        for container-item ∈ container.items do
            length, width, height ← container-item.dimensions
            if axis = x then
                offset ← (length, 0, 0)
            else if axis = y then
                offset ← (0, width, 0)
            else
                offset ← (0, 0, height)
            return Try-Add-Item-At-Position(container, item,
            container-item.position + offset)
    return False
```

```
function Pack-Items-Into-Containers(items, containers)
    unpacked ← Sort items in descending order of volume
    for container in containers do
        container.items ← []
        for item in unpacked do
            if Try-Add-Item-To-Container(container, item) then
                unpacked ← unpacked - item
    return unpacked
```

This algorithm can provide a reasonable packing solution for the E-commerce packaging problem. The heuristic approach can be visualized as follows:

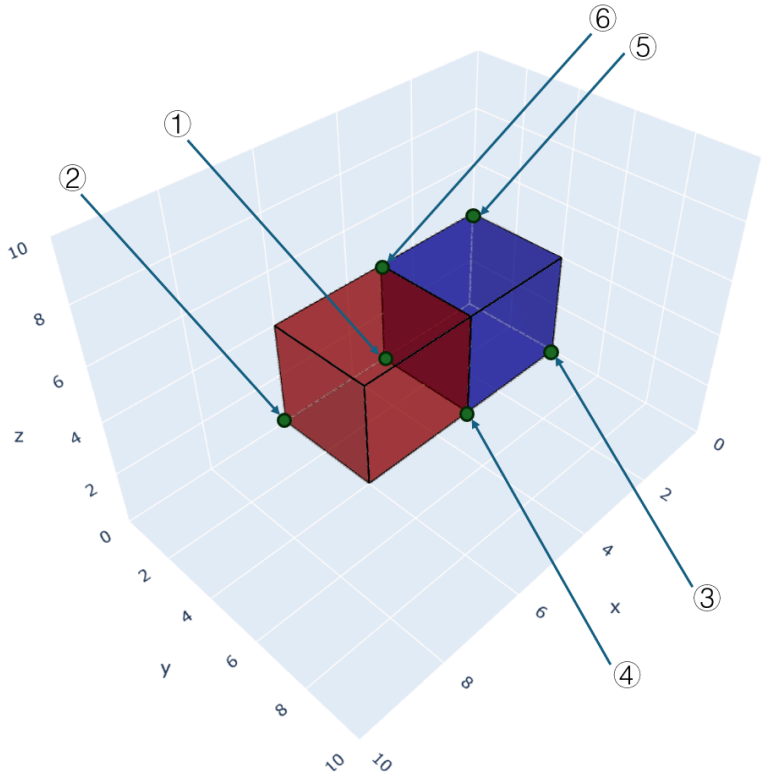

Figure 6: Visualization of heuristic algorithm

Figure 6 shows the packing process of the heuristic approach. For a new item, the algorithm sequentially tries position 1 to 6. This is consistent with human intuition, as we tend to try first pack the items into a row, then a layer, and finally stack the items on top of each other. The heuristic approach is simple and effective, providing a reasonable packing solution for the E-commerce packaging problem.

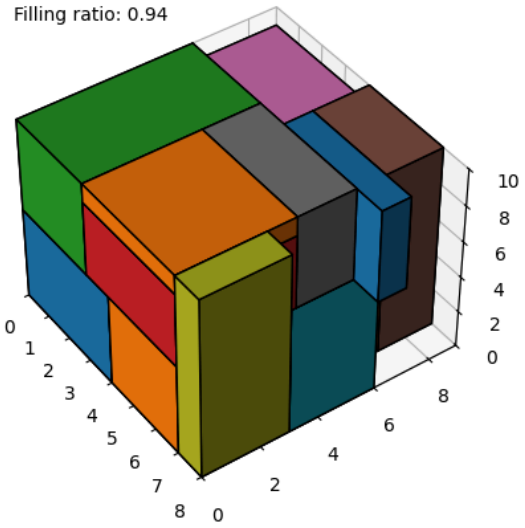

Figure 7: Packing result of the heuristic approach

The heuristic method can reach a relatively high filling rate, as is shown in Figure 7. However, the heuristic approach is not optimal and may not always find the best solution. Besides, the time complexity of the algorithm is around $O(n^3)$, which is not efficient for large instances of the problem.

4.2.2 E-commerce Packer

Before we can apply the heuristic approach to the E-commerce packaging problem, we need to do some preprocessing to the dataset.

The first problem is that not all items can be packed into the containers. Some items are too large to fit into any available containers, if placed orthogonally. We need to discard these items and they are not considered in the packing process. Note that in the final result, these impossible items are taken into consideration. There are a total of 221 impossible items out of 21500 items, so this will not affect the final result significantly.

Another problem is that the dimensions of the items are not all integers, which is a requirement of the algorithm. To increase the accuracy of the algorithm, we multiply all dimensions by 10 and round them to the nearest larger integer. The filling rate is still calculated based on the original dimensions of the items.

To tackle the E-commerce packaging problem, we have to make some modifications to the algorithm to handle multiple containers.

A packer now receives only a size of a container. When the packer is called, it will pack the items into the container based on the heuristic approach. If the current container can not hold all the items, a new container will be created and the remaining items will be packed into the new container. This process will continue until all items are packed and there will be no items left. The packer will return a list of containers that are filled with items.

As there are 8 available container sizes, we need to create 8 packers, each corresponding to a different container size. The packers will pack the items into the containers based on the heuristic approach. The final result will be the packer with the highest filling rate.

4.2.3 Optimization Ideas

The packer described above has the ability to pack the items into the containers efficiently. However, it is practically impossible to run the algorithm due to its time complexity. Besides, the effectiveness of the packer can be further improved by optimizing the algorithm. To improve the efficiency and effectiveness of the packer, we propose several optimization ideas.

In order to reduce the time complexity of the algorithm, we can select a batch of items from the inventory and feed them to the packers in each iteration. This will reduce the number of items to be packed and speed up the packing process. The batch size can be tuned to balance the trade-off between efficiency and effectiveness.

By intuition, a batch should contain a mix of large, medium, and small items. This is because large items can take up a lot of space and should be packed first, while small items can fill the remaining space efficiently. By selecting a mix of items in each batch, we can ensure that the containers are filled efficiently and the number of containers used is minimized.

Specifically, we can set a ratio for the number of large, medium, and small items in each batch. For example, we can set the ratio to be 16:32:96, meaning that we select 16 largest items, 96 smallest items and 32 medium

items in each batch. This ratio can also be tuned to improve the performance.

With batched items, we can pack them into all 8 packers and choose the packer with the highest filling rate as the final result. This strategy can help to find a better packing solution.

Another idea is to set a minimum threshold for the filling rate of the container. If the filling rate of the container is below the threshold, the container will be discarded and the items will be returned to the inventory. This prevents the packer from creating too empty containers. However, there may not be a feasible solution if the threshold is set too high.

The process of the E-commerce packaging problem can be summarized as Figure 8.

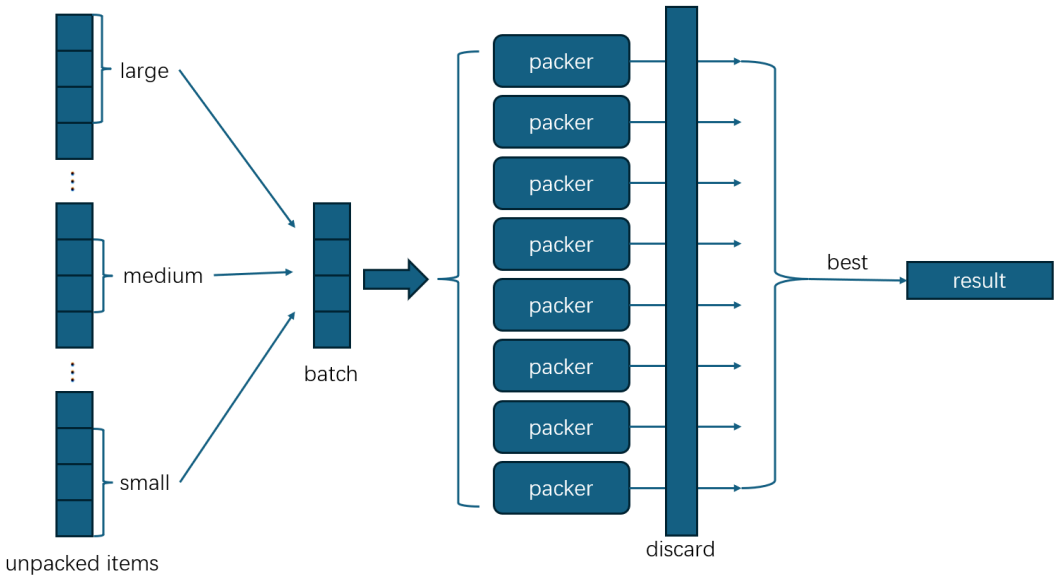

Figure 8: Logic of E-commerce packer

4.3 Results and Discussion

The performance of the E-commerce packaging problem depends on the hyperparameters chosen, including the threshold for the filling rate, the batch size for the items, and the small, medium, and large item ratios of a batch. By tuning these hyperparameters, we reach a final 79.99% filling rate, which is a reasonable result for the E-commerce packaging problem.

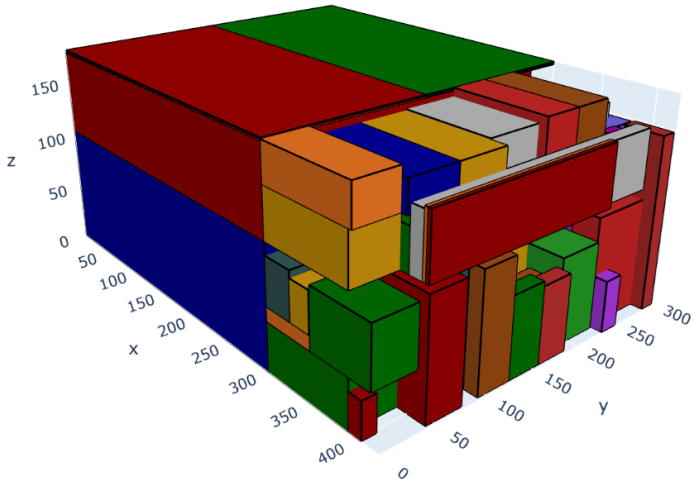

Figure 9: First container of the E-commerce packaging problem

Figure 9 illustrates the first container of the final result. The large items are placed first, followed by the medium items, and finally the small items fill any remaining space. This strategy is similar to the common strategy of human workers, who tend to pack large items first and then fill the gaps with smaller items.

There are still many improvements that can be made to the E-commerce packer. For example, we can further optimize the algorithm to reduce the time complexity and improve the efficiency of the packer. We can also explore different strategies for packing the items into the containers and find the best combination of hyperparameters to maximize the filling rate.

Another idea is to use machine learning techniques mentioned in the previous sections to train a policy network model or a neural guided search algorithm for the E-commerce packaging problem. The main difficulty lies in the large observation space and action space of the problem, which requires a more sophisticated model and training process. But combined with the heuristic approach, machine learning techniques may provide a more effective solution to the E-commerce packaging problem. For example, the policy network can choose from several heuristic algorithms to place one item and choose another algorithm to place the next. This can be a good way to combine the advantages of machine learning and heuristic algorithms.

Moreover, physical constraints can be added to the algorithm. For example, an item can not float in the air, and the stability of the stacked items should be considered. These constraints can be added to the algorithm to make the packing process more realistic and practical.

The detailed statistics of the final result are as follows:

- Total containers: 6965
- Total items: 21279
- Total volume of containers: 148 616 283
- Total volume of items: 118 880 654 (approximate)
- Filling rate: 79.9917% (approximate)
- Unpacked items: 221

5 Conclusion

The Bin Packing Problem is a challenging combinatorial optimization problem that arises in various real-world applications, including logistics, manufacturing, and resource allocation. In this report, we have explored different approaches to solving the 3D Bin Packing Problem and the E-commerce packaging problem using machine learning techniques and heuristic-based approaches.

We first trained a policy network model using the PPO algorithm to learn a policy for packing items into containers efficiently. The policy network model showed some promising results but struggled to find an optimal solution due to the complexity of the problem and the limited training resources. We then proposed a neural guided search algorithm that combines tree search algorithms with policy networks to efficiently explore the search space and find near-optimal solutions and demonstrated its effectiveness.

Finally, we addressed the E-commerce packaging problem using a heuristic-based approach that leverages human intuition to pack items into containers efficiently. The heuristic approach provided a reasonable packing solution for the problem, but the time complexity of the algorithm was a limiting factor. We then proposed several optimization ideas to improve the efficiency and effectiveness of the packer and achieved a 79.99% filling rate for the E-commerce packaging problem.

In conclusion, the 3D Bin Packing Problem and the E-commerce packaging problem are challenging optimization problems that require a balance between exploration and exploitation. By combining machine learning techniques with heuristic-based approaches, we can effectively solve these problems and find high-quality solutions that minimize the total volume of the containers used and reduce the shipping costs.

Acknowledgement

We would like to thank the authors of the reference papers and the dataset providers for their valuable contributions to the field of combinatorial optimization. We would also like to acknowledge the support of our team members and the guidance of our instructors throughout the project.

The dataset and code repository used in this report can be found at the following link: GitHub Repository.

To verify the results and reproduce the experiments, please refer to the code implementation in the repository.

Some figures in this report are adapted from the reference papers and the dataset, and are used for illustrative purposes only. Part of the implementation is based on this repository: 3D Bin Packing.

The contributions of each team member are as follows:

Name	Student ID	Score	Work
方一帆	522031910485	25%	Literature review, Parameter tuning
王昕宇	522031910835	30%	Methodology of Task 2, Literature review, Parameter tuning
万凌龙	522031910830	45%	Methodology of Task 1 and Task 3, Code implementation, Report writing

References

[1] Q. Que, F. Yang, and D. Zhang, "Solving 3D packing problem using Transformer network and reinforcement learning," *Expert Systems with Applications*, vol. 214, p. 119153–119154, 2023, doi: https://doi.org/10.1016/j.eswa.2022.119153.

[2] Q. Zhu *et al.*, “Learning to Pack: A Data-Driven Tree Search Algorithm for Large-Scale 3D Bin Packing Problem,” in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, in CIKM '21. Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021, pp. 4393–4402. doi: 10.1145/3459637.3481933.

[3] E. Dube, L. Kanavathy, L. K@i, and O. Za, “Optimizing Three-Dimensional Bin Packing Through Simulation,” 2006, p. .