Learning to Pack: A Data-Driven Tree Search Algorithm for Large-Scale 3D Bin Packing Problem

Qianwen Zhu*
Nanjing University
Nanjing, China
zhuqw@smail.nju.edu.cn

Xihan Li
University College London
London, The United Kingdom
xihan.li@cs.ucl.ac.uk

Zihan Zhang
Huawei Noah's Ark Lab
Shenzhen, China
zihan.zhang@huawei.com

Zhixing Luo
Nanjing University
Nanjing, China
luozx@nju.edu.cn

Xialiang Tong
Huawei Noah's Ark Lab
Shenzhen, China
tongxialiang@huawei.com

Mingxuan Yuan, Jia Zeng
Huawei Noah's Ark Lab
Shenzhen, China
{yuan.mingxuan,zeng.jia}@huawei.com

ABSTRACT

The 3-dimensional bin packing problem (3D-BPP) is not only fundamental in combinatorial optimization but also widely applied in real world logistics. In the modern logistics industry, the complexity of constraints, heterogeneity of cargoes and scale of orders are dramatically increased, leading to great challenges to devise packing plans up to standard. While the tree search algorithm is proved to be a successful paradigm to solve the 3D-BPP, it is too time-consuming to be applied in the aforementioned large-scale scenarios. To overcome the limitation, we propose a data-driven tree search algorithm (DDTS) to tackle the 3D-BPP. The solution space with complicated constraints is explored by a tree search algorithm, and a convolutional neural network trained with historical data guides pruning the tree so as to accelerate the search process. Computational experiments on real-world datasets show that our algorithm outperforms the state-of-the-art approach with a loading rate improvement of 2.47%. Moreover, the deep learning technique increases searching efficiency by 37.14% with only 0.04% performance loss. The algorithm has been deployed in Huawei Logistics System, which increases the loading rate by 3% and could reduce the logistics cost by millions of dollars per year. To the best of our knowledge, we are the first to embed pruning networks into tree search for the large-scale 3D-BPP.

CCS CONCEPTS

• **Applied computing** → **Operations research**; **Transportation**;
• **Mathematics of computing** → **Combinatorial optimization**;
• **Computing methodologies** → **Machine learning**.

*This work is done when Qianwen Zhu worked as an intern at Huawei Noah's Ark Lab.

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3481933

KEYWORDS

combinatorial optimization, deep learning, bin packing problem, container loading problem, tree search, convolutional neural network

ACM Reference Format:
Qianwen Zhu, Xihan Li, Zihan Zhang, Zhixing Luo, Xialiang Tong, and Mingxuan Yuan, Jia Zeng. 2021. 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 and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3481933

1 INTRODUCTION

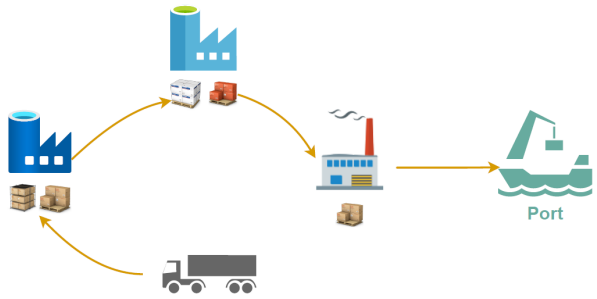


Figure 1: 3-dimensional bin packing problem

How to load cargoes into containers as full as possible is a core problem in logistics and manufacturing industries, which is considered in many scenarios, e.g., box packing, vehicle loading and warehouse stacking. As shown in Figure 1, a large number of cargoes of different types and shapes in different depots need to be loaded into a vehicle, considering many practical constraints. Only a slight improvement in loading rates could significantly reduce the logistics cost, resulting to a huge market value. Taking the logistics scenario of Huawei, a leading global provider of ICT infrastructures and smart devices, as an example, 1% loading rate increase could reduce the logistics cost by more than one million US dollars per year. Packing a set of 3D rectangular items into a 3D rectangular container under the specified constraints is known as the 3-dimensional bin packing problem (3D-BPP) or the 3-dimensional container loading problem (3D-CLP). The objective of the 3D-BPP is to maximize the loading rate of the container. It is a typical NP-hard problem[22]. There are two major challenges in the real world. **(1) Abundant**

constraints. The constraints in real scenarios are abundant and complex. For example, there are 30+ constraints in Huawei vehicle loading scenario. Whenever an item is packed into the container, all constraints need to be checked to ensure the feasibility of the loading solution. Manually doing this job is time-consuming and almost impossible. The fragmented solution space makes it difficult to find an acceptable solution in a reasonable time. **(2) Strong heterogeneity.** Strong heterogeneity refers to the variety of materials and shapes of cargoes. The size of the search space increases exponentially with the increase of heterogeneity, which makes it take much more time to obtain a feasible and satisfactory solution.

This paper proposes a data-driven tree search algorithm (DDTS) to tackle the real-world 3D-BPP. The whole packing process consists of a sequence of packing actions. In each packing action, we have to determine 1) which item to pack and 2) where it should be placed. To determine the best packing action, a lookahead mechanism is designed to search all possibility of items and their locations. For a item-location pair, we check all constraints to guarantee that the item could be placed at the location legally. Then the item-location pair is evaluated by a *simulation loading*. Specifically, in the simulation loading, after the item is placed at the corresponding location of the pair, the remaining items are packed afterwards by a heuristic method. Thus the pair is scored by the simulation loading rate of the container. Among all pairs, the pair with the highest score is selected. Finally, we place the item at the location of the selected pair. Since the process of constraint checking and simulation loading is time-consuming and the number of item-location pairs is large in each action, evaluating all pairs is inefficient and almost impossible. Therefore, to avoid using constraint checking and simulation loading to prune unpromising pairs, we train a pruning convolutional neural network instead. The historical information enables the network to help select the best item-location pair from all possibilities more efficiently in each packing action. Computational experiments present that our algorithm generates better solutions than the state-of-the-art approach, in which the pruning network acts well. The algorithm has been successfully online, which brings immediate commercial value.

The contribution of this paper is threefold.

(1) A pruning network is trained using historical data to empower the tree search algorithm, in which a CNN model extracts features of a container and items. To the best of our knowledge, we are the first to apply machine learning techniques to prune branches in optimization methods for the large-scale 3D-BPP. Speeding up with the pruning network saves 37.14% time with only 0.04% performance loss.

(2) We address a practical 3D-BPP with abundant constraints and a large number of strongly heterogeneous cargoes, which emerges as the rapidly rising demand and is rarely studied in previous researches. Then a data-driven tree search algorithm is proposed to solve the problem effectively.

(3) The numerical experiments on real-world datasets show that our algorithm outperforms the state-of-the-art approach with a loading rate improvement of 2.47%. The algorithm has been deployed in Huawei Logistics System, and the average loading rate has increased by 3% over the old system. It is conservatively estimated to save millions of dollars in logistics costs every year.

2 RELATED WORK

2.1 3D bin packing problem

The bin packing problem (BPP) is one of the most important problems in combinatorial optimization and theoretical computer science. The BPP is useful in practice and finds numerous applications in scheduling, routing and resource allocation problems. According to the number of dimensions, the BPP can be classified into 1D, 2D and 3D. Effective implementations of the 3D-BPP are extremely in demand for the logistics industry and manufacturing industry, considering the practicality and difficulty of the problem. The traditional methods to deal with the 3D-BPP are exact algorithms, heuristics and meta-heuristics. Exact algorithms, i.e., mixed integer linear programming models and branch & bound methods, have been proposed [12, 13, 18]. However, these algorithms have high time complexity especially when a large number of heterogeneous cargoes are involved. Heuristics and meta-heuristics are viable to obtain acceptable solutions in a reasonable time. Heuristics aims to pack items according to some fixed rules, such as First Fit [6], Best Fit [21]. These methods are fast, but the quality of obtained solutions is relatively poor. Existing meta-heuristics methods apply search algorithm to this problem, including Biased Random-Key Genetic Algorithm [8], Particle Swarm Optimization [5], Variable Neighborhood Search[20], Greedy Randomized Adaptive Search Procedure [19], etc. Nowadays, the most successful approaches for 3D-BPPs are the block-building-based tree search methods. Blocks are the combinations of items, which simplifies the search process. [28] identified the 6 key elements to all block building approaches and proposed the greedy 2-step lookahead algorithm. Then [1] presented a state-of-the-art beam search approach embedded with a new evaluation function for ranking items, which will serve as one of our comparison methods in subsequent experiments. Moreover, there are some related surveys. [27] reviewed the design and implementation of solution methodologies for solving the 3D-BPP. [2] studied practically-relevant constraints of the BPP. Traditional methods have not accumulated and utilized historical experience to raise the efficiency and quality of solutions, which is a potential direction for research.

In terms of problem scale, existing researches generally consider 4 or 5 common constraints, and the quantity of items is close to 130, and the maximum number of categories is 100, which is put forward in [4]. The scale of 3D-BPPs in the actual scenarios we investigated includes: 1) 30 items and 5 item types per order with 5 constraints [3]; 2) 307.13 items and 94.68 item types per order with 10 constraints [17]; 3) 64 items and 2 basic constraints [26]. The number of cargoes and constraints in the existing research can hardly reach the scale in our article (61 types of 658 items with 30+ constraints).

2.2 Learning for combinatorial optimization

Recently, machine learning has shown potential in operation research. Some studies have applied learning technologies to the 3D-BPP. [17] extracted features manually and trained regression models to predict loading rates for 3D-BPPs to make time for routing. [3] solved a multi-level bin packing problem in real-life scenarios using manual historical packing records by fuzzy-matching algorithms and dynamic programming approaches. Besides, many researches

attempt to employ deep reinforcement learning methods to solve variants of the 3D-BPP, such as the 3D flexible BPP [7, 11, 16] and the online 3D-BPP [23, 26]. Among them, [26] and [16] applied a Monte Carlo Tree Search (MCTS). [26] combined tree search with deep learning for the online 3D-BPP from the perspective of value estimation to speed up the node scoring process. [16] developed a deep neural network to estimate a policy and a value function, as well as MCTS for policy improvement to solve the 3D flexible BPP. However, as far as we know, for the large-scale offline 3D-BPP, no research has tried the combination of tree search and deep learning.

Besides 3D-BPPs, many NP-hard problems have been resolved inspired by learning-guided fashions. [9, 14, 15] proposed learning technologies in a branch and bound search for mixed-integer linear programming. Specifically, [9] learned a node selection policy and a node pruning policy using imitation learning. [14] learned the Strong Branching strategy for variable selection. [15] learned to decide which heuristic should be run at every node of the search tree to help select nodes and variables. For other problems, [24] presented a learning-based A* algorithm for recovering the graph edit distance (GED). They developed a graph edit neural network to predict similarity scores in replacement of manually designed heuristics in traditional A*. [25] proposed machine learning-driven upper bounds for the container relocation problem (CRP). They manually designed problem-specific features and used traditional classification methods to obtain a classifier to help calculate tighter upper bounds with only a little extra time. Furthermore, the new upper bounds were injected into an exact branch-and-bound algorithm and a beam search method to enhance their respective performance. [10] proposed a deep learning heuristic tree search to tackle the container pre-marshalling problem (CPMP). They integrated deep neural networks to decide which branch to choose next and to estimate a bound for pruning the search tree.

In conclusion, recently various studies have attempted to embed learning mechanisms into tree search by replacing time-consuming modules or guiding the search process, thus making the solving more efficient. For specific problems, there are two vital concerns: 1) methods to capture problem features such as encoders and neural networks; 2) appropriate modules in optimization algorithms to be modified. Furthermore, to the best of our knowledge, we are the first to embed deep learning technologies into optimization algorithms for the large-scale 3D-BPP from the perspective of pruning to cut down the nodes that need to be scored.

3 PROBLEM DESCRIPTION

Formally, given a container Γ and a index set of items $\mathcal{I} = \{1, \ldots, i, \ldots, |\mathcal{I}|\}$, the objective of the 3D-BPP is to pack items from $\mathcal{I}$ into Γ, maximizing the loading rate of Γ. The dimensions of Γ and item i are denoted by (L, W, H) and (L_i, W_i, H_i). Each item is located at a certain depot. The objective of our problem is:

$$\max \frac{\sum_{i=1}^{N} q_i \cdot L_i \cdot W_i \cdot H_i}{L \cdot W \cdot H} \tag{1}$$

of which q_i is a boolean variable indicating whether item i has been packed in Γ. An objective corresponds to a packing plan $\Omega = \{(i, x, y, z, o) \mid i \in \mathcal{I}\}$. Ω gives the position (x, y, z) and the orientation o for each item i s.t. $q_i = 1$. There are 30+ constraints in our practical industrial scenario. The problem is subject to the following constraints according to the classification scheme in [2].

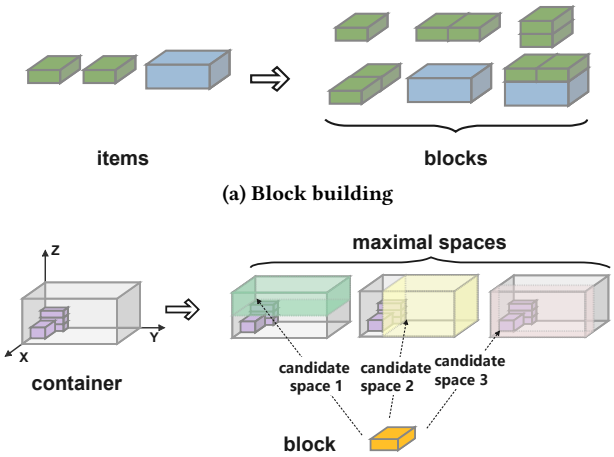
Container-related constraints. (1) Shape constraint. A container must be a cuboid. (2) Weight and volume limits. A container can only be loaded with items as long as the weight and volume limits are not exceeded.

Item-related constraints. (1) Shape constraint. An item must be a cuboid. (2) Parallel constraint. The item faces must be parallel to the container faces. (3) Last-In-First-Out constraint. All items must be directly loaded and unloaded through a sequence of straight movements parallel to the length of the container without any repositioning of other items. (4) Non-overlapping constraint. Items cannot overlap with any other item. (5) Orientation constraint. Allowed orientations for each item are restricted. (6) Fragility constraint. Non-fragile items can not be placed on fragile ones. (7) Support area constraint. i) The underside of each item must be supported by the items below or the bottom of the container at least by a certain percentage. ii) Items above a certain height can only be supported by a single item. iii) Some items can only be supported by the bottom of the container. (8) Load bearing constraint. i) When items with different bottom sizes are stacked, the weight of the items above can not exceed the maximum bearing weight of the items below. ii) When items with the same bottom sizes are stacked, the stacked layers of items cannot exceed the maximum allowed layers of items. (9) Height constraint. Items within a certain distance from the door can not exceed the door height. (10) Packing material constraint. Woven bags can not be placed on cartons.

Route-related constraints. (1) Adjacent constraint. The overlap of items in adjacent depots in the length direction of vehicles can not exceed a certain value. (2) Chronological constraint. The items at the later visited depots can not be loaded before the items at the former visited depots.

4 DATA-DRIVEN TREE SEARCH ALGORITHM

4.1 Preliminaries on tree search algorithm



(a) Block building

(b) Space generating

Figure 2: Spaces and blocks

At each packing step, we have to determine 1) which item to pack and 2) where it should be placed. To overcome these problems we first build blocks and spaces.

Block building. To simplify the process of packing items, we build blocks [28] that are cuboids made up of items in $\mathcal{I}$, and then pack blocks instead of items into the container. As shown in Figure 2(a), a block $b = \{(i, x, y, z, o) \mid i \in \mathcal{I}',\ \mathcal{I}' \subseteq \mathcal{I}\}$ consists of items. At each packing step, considering the number of the unpacked items, a set of all possible blocks $\mathcal{B}$ will be generated as candidates for the next packing action. The block building algorithm is provided in Appendix A.

Space generating. A space is a cuboid within the container where blocks could be placed. The initial space of an empty container is the container itself. At each step, a block and a space will be selected. After the selected block being packed in the selected space, the new empty spaces within the original one will be generated. The empty cuboid spaces that cannot be extended in any dimension in the original space are called maximal spaces [1], which will be the candidates for the next packing action. At most three new overlapping maximal spaces will be generated after a block being packed at a corner of the original space. The space generating algorithm is provided in Appendix B.

Tree search algorithm. Algorithm 1 describes the tree search algorithm used in our problem, which is called Greedy LookAHead algorithm (GLAH). We will explain the details of pseudocode in the following. In line 4, The pair (s, b) is feasible means block b can be packed into space s satisfying all constraints. Spaces in $\mathcal{S}$ are sorted by the distance from the origin, blocks in $\mathcal{B}$ are sorted by the size. Then select feasible pairs up to max_p greedily. In line 5, the lookahead strategy aims to select the best block-space pair from all candidates, which is shown in Figure 3(a). The state in our tree search algorithm consists of two parts, one is a container that has already loaded some items, the other is a set of items outside the container to be loaded. The colorful circles represent the states. After some items are packed, the current state transforms to a new state. The set of dark blue circles is a list of states $\mathcal{T}$ transformed from the light blue circle by $\mathcal{P}$ with parameter $max\_p = 3$, which is called the greedy width. We lookahead every state in $\mathcal{T}$ to score them respectively. Detailed lookahead procedure with parameter $m = 4$ is shown by orange circles. To score a dark blue circle $t \in \mathcal{T}$, we virtually execute m different feasible packing actions, and then get dark orange circles $\mathcal{T}' = \{t'_1, \ldots, t'_m\}$. Afterwards, we obtain m loading rates $\{l(t'_1), \ldots, l(t'_m)\}$ by simulation loading all elements in $\mathcal{T}'$. Finally we take $\max\{l(t'_1), \ldots, l(t'_m)\}$ as the score of t. m is called the lookahead width. In line 7, the origin designated in the container with coordinates, and examples of the corner selection for each candidate space are shown in Figure 2(b). In line 8, all i and o can be obtained according to b. Besides, Figure 3(b) shows the pruning in lookahead which will be explained immediately.

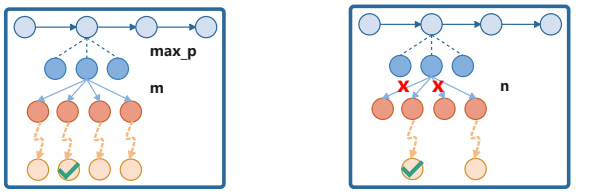
4.2 Deep learning guidance

The lookahead procedure is frequently called when running the algorithm and the simulation loading is time-consuming, considering numerous cargoes and constraints. Hence, accelerating the lookahead process will save considerable time. In our DDTS, we apply deep learning technologies to decrease $|\mathcal{T}'|$ from m to n in lookahead procedures, as shown in Figure 3(b), where n is the branch size after pruning. It helps the total computational time acceptable with a little loss of performance. It is worth mentioning that we check all constraints in the tree search algorithm and then input feasible states into machine learning methods, thus avoiding the problem that the end-to-end learning needs to guarantee the feasibility of solutions.

Algorithm 1 Greedy LookAHead

Input: A container Γ; A item index set $\mathcal{I}$.
Output: A packing plan Ω.

1: Generate a block list $\mathcal{B}$ with $\mathcal{I}$, $|\mathcal{B}| \leq max\_bk$;
2: Initialize a space list $\mathcal{S}$ with Γ;
3: **while** $\mathcal{S} \times \mathcal{B} \neq \emptyset$ **do**
4: Select a list of feasible pairs $\mathcal{P} = \{p = (s, b) \mid s \in \mathcal{S}, b \in \mathcal{B}\}$, $|\mathcal{P}| \leq max\_p$;
5: Select a pair p with maximum loading rate from $\mathcal{P}$ according to **lookahead** mechanism;
6: **if** p exists **then**
7: Place b at the corner of s closest to the origin;
8: Update Ω by adding (i, x, y, z, o) for each item i in b;
9: Update $\mathcal{S}$ and $\mathcal{B}$;
10: **else**
11: Break;
12: **end if**
13: **end while**
14: **return** Ω.



(a) Lookahead before pruning (b) Lookahead after pruning

Figure 3: Lookahead before and after pruning

4.2.1 State representation. In our algorithm, we need to select the appropriate pair for execution. We do not model the candidate pair but model the state after the corresponding pair is executed. In this section, we use a CNN-based network to represent the state in the tree search. The main component of our network consists of two branches, one is to capture container features, the other is to retain features of a set of items.

Container representation. CNN has achieved great performance in processing images, video, speech and text for extracting features. For 3D-BPPs, recent researches attempt to apply deep learning technologies, and the CNN model has been used to extract the features of containers. In this work, we continue to employ CNN to capture the container features. Before constructing CNN, we first need to encode the raw information of a container into an input format for neural networks. To this end, we use a method called "height map" to encode the container. The height map is of size $L \times W$ indicating the height of stacked items in the container at each cell, which is shown in Figure 4. After encoding the raw container information

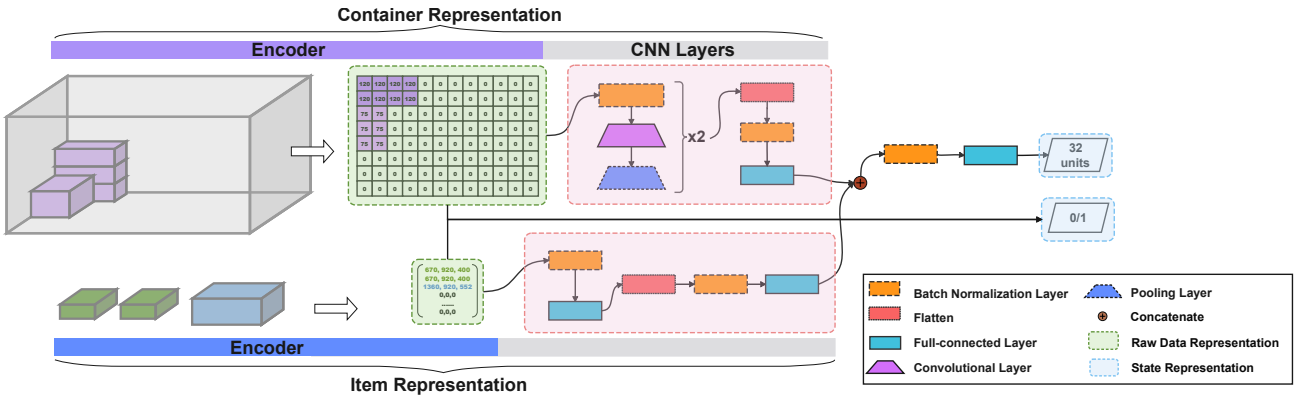


Figure 4: State representation

into a height map, we construct a CNN to convert the height map into features. In the following, we explain the details of our CNN. Firstly, we apply a batch normalization layer that makes the heat map to a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. Then we use a 2D convolution layer to produce a tensor of outputs. The number of output filters in the convolution is 1, and the height and width of the 2D convolution window are both 3. We use ReLU as the activation function. Then we downsample the input along its spatial dimensions (height and width) by taking the maximum value over an input window of 2×2 for each channel of the input. Afterwards repeat the above three layers, and flatten and normalize the output. Finally, a fully-connected layer with the ReLU activation function is applied to obtain a 32-dimensional tensor.

Item representation. In our large-scale 3D-BPP, the number of items is 658 on average. As far as we know, the number of items in the existing deep learning related studies of 3D-BPPs is small, such as 1 to 8 [23, 26]. Therefore, for the large-scale 3D-BPP in this paper, we design an encoder for items as $[L_i, W_i, H_i] \in \mathbb{Z}^3$, $i \in \mathcal{I}'$, $\mathcal{I}' \subseteq \mathcal{I}$, where $\mathcal{I}'$ is the set of items to be packed. For the size of neural networks needs to be specified, we set a threshold of the maximum number of items called *max_item*. If the number of items in the state is less than this threshold, then the last vectors are filled with $[0, 0, 0]$. This item encoder is effective in our problem and is shown in Figure 4. After encoding the items, first we normalize the input and then apply a fully-connected layer with ReLU as the activation function to obtain a 128-dimensional tensor. Then flatten and normalize the output. Finally, a fully-connected layer with the ReLU activation function is used to obtain a 32-dimensional tensor.

State representation. Having known the representation of a container and items, first we concatenate the two tensors, then normalize the output, finally we apply a fully-connected layer with the ReLU activation function to obtain a 32-dimensional tensor. Moreover, we use a boolean mask value to indicate whether the state is empty or not. 0 for empty and 1 for not. If there are less than m candidate states, for the network size is fixed, we set the last states as 32-dimensional tensors whose elements are all 0 and set their corresponding mask values to 0. In brief, the representation of a state consists of two parts: a 32-dimensional tensor and a boolean mask value, as shown in the blue shaded area in Figure 4.

4.2.2 Pruning network. The raw input of the pruning network is m states. The output is a vector of scores between 0 and 1. Under the guidance of the output, the tree search can only retain the valuable branches with the top n scores and explore their loading rates. The network architecture is shown in Figure 5. For our pruning network, on one hand, we concatenate m 32-dimensional tensors from states and then normalize them. Afterwards, we apply a fully-connected layer with the Sigmoid activation function to obtain a m-dimensional tensor. Value in each dimension is the score of the state at the corresponding position. On the other hand, we concatenate m masks. Finally, we multiply masks with the m-dimensional tensor to invalidate the score of dummy states and avoid selecting them. The optimizer used in our model is RMSprop. Our model uses binary cross entropy as the loss function and binary accuracy as the metric.

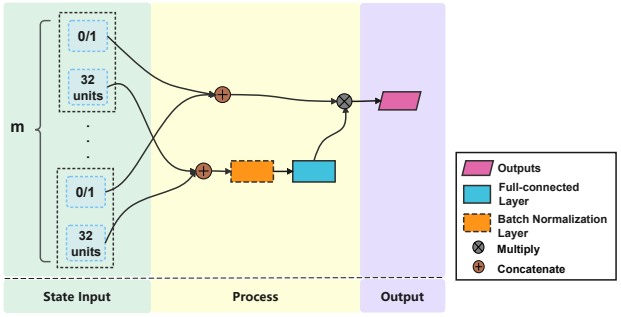


Figure 5: The architecture of pruning network

4.2.3 Workflow. We have two types of orders: historical orders and new orders. Historical orders are solved by a time-consuming tree search algorithm with lookahead width m. During the search, we collect branch decision records. In more detail, for each lookahead procedure, we create a branch example $(\mathbf{x}, \mathbf{y})$. Feature $\mathbf{x}$ records the information of a list of m states $\mathcal{T}'$, of which $\mathcal{T}'$ is the set of state t' and is introduced in Section 4.1. That is, $\mathbf{x} = \{t'_1, \cdots, t'_i, \cdots, t'_m\}$. Label $\mathbf{y}$ is a boolean vector of length m. $\mathbf{y} = \{y_1, \cdots, y_i, \cdots, y_m\}$, where $y_i = 1$ only when $l(t'_i)$ is large enough to be selected, otherwise, $y_i = 0$, $\sum_{i=1}^{m} y_i = n$. After finishing all historical orders, we accumulate a large number of instances. Instances are split into training and test sets. Then a pruning network is learned in a supervised way and saved for later usage. With a well-trained network, we do not need to calculate loading rates for all m branches in lookahead procedures. We could only explore the promising n branches according to the output of the pruning network. In brief, a data-driven tree search algorithm means learning from historical

branching choices to make future choices with fewer detours. Recall the example shown in Figure 3, $m = 4$, $n = 2$, that is, for each state (dark blue circle) to be evaluated, we reduce $m - n = 2$ times of simulation loading.

5 COMPUTATIONAL EXPERIMENT

5.1 Experimental setup

The experiments are conducted on a Linux server of Ubuntu 18.04.4 LTS with 4 Intel(R) Xeon(R) Platinum 8180M CPUs@ 2.50GHz and 1T Memory. The tree search algorithm is mainly coded in Java and network models are implemented using TensorFlow v2.4.0.

Dataset. Our dataset consists of real-world large-scale packing orders from September to December 2020 in Huawei logistics. It contains 1784 orders. Each order has 500 to 1000 items. Among all orders, the average number of item categories is 61, the average number of items is 658, and the average number of depots is 5. The total number of item categories is 4697, and the total number of depots is 71. Orders are similar to a certain extent. The length, width and height of our vehicle are 4.1 meters, 2.3 meters and 2.27 meters respectively. The dataset is randomly divided into the training set and test set with the order number of 1249 : 535. We run the tree search algorithm without pruners using all orders in the training set, and extract 25419 branch decision records, then use these branch samples to train and test the pruning network.

Compared Methods. We implement one state-of-the-art algorithm, and other algorithms are conducted to observe the profit of modules, such as simulation loading to evaluate states and the lookahead procedure.

- Random: Select the block and space randomly under constraint conditions.
- Constructive Heuristic: Select the block and space according to heuristic rules such as selecting the largest block that can be put into the space satisfying all constraints.
- Greedy Search: Algorithm 1 without lookahead procedures. Select the block and space according to the simulation loading rates directly.
- Beam Search [1]: The state-of-the-art algorithm that was proposed to solve the 3D-BPP.

Metrics. We use 3 evaluation metrics in this paper.

- Loading rate: The objective of 3D-BPPs illustrated in Eq. 1.
- Time: Calculation time for each order in seconds.
- Best loading: The ratio of best packing plans obtained by the current method.

Parameter settings. The size of the pruning network is $L = 41$, $W = 23$, $max\_item = 1000$. We empirically set the learning rate as 5e-5. The maximum number of training epochs is set as 500. The early stopping mechanism is applied to our CNN models. The monitored quantity is the loss of the validation set. The early stopping patience is set as 20 epochs. We restore model weights from the epoch with the best value of the monitored quantity. We reduce the learning rate when the loss of the validation set has stopped improving for 10 epochs. The factor by which the learning rate will be reduced is 0.5. The lower bound on the learning rate is 1e-6. For our DDTS, $m = 12$, $n = 4$, $max\_bk = 5000$, $max\_p = 12$. For Beam Search, the beam width is 150.

5.2 Results

Table 1: Comparison results

	Loading rate (%)	Time (s)	Best loading (%)[2]
Random	53.80	1.42	1.88
Constructive Heuristic	72.91	1.92	43.44
Greedy Search	74.04	**0.61**	60.00
Beam Search	74.84	30.18	61.88
DDTS	**76.69**	27.32	**97.19**
Comp[1]	2.47%	-10.47%	36.12%

[1] Comp = (DDTS - Beam Search) / Beam Search.
[2] A best packing plan may be obtained by more than one method, so its sum of all methods may be over 100%.

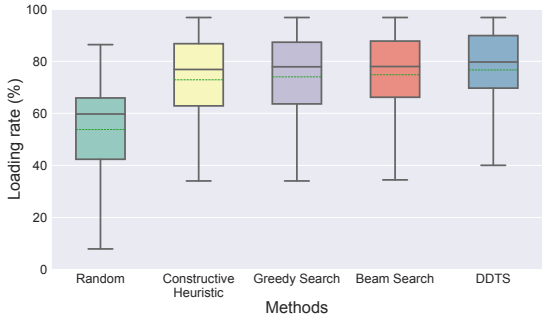


Figure 6: Comparison of loading rates for all 5 algorithms, over 500 test orders

The test accuracy of the trained pruning model is 0.89. Computational results are presented in Table 1. "Comp" means the comparison between our algorithm and the state-of-the-art algorithm, i.e., (DDTS - Beam Search) / Beam Search. In the following, the effects of each module in DDTS are analyzed. Not surprisingly, the loading rate of Random demonstrates that there is still much room for improvement. That is, although constraints are numerous, different methods will lead to a great difference in the quality of solutions. The loading rate of Constructive Heuristic is over 1% lower than those methods using simulation loading procedures to select actions, which illustrates the importance of proper selection with simulation loading procedures. Our new proposed DDTS obtained a loading rate of 76.69%, which is 2.47% higher than the state-of-the-art algorithm in a shorter time. Moreover, DDTS obtained 97.19% best packing plans for all orders, which explains the stability of DDTS on various orders, instead of performing extremely well only on some orders to obtain a higher average loading rate. It is crucial for practical use. The box whiskers plot shown in Figure 6 illustrates that DDTS has higher median loading rates and average loading rates than other algorithms, and the distribution is more centralized. At last, considering the time and the quality of the solution, Greedy Search is recommended when acceptable solutions are needed in extremely limited time (The reason why Greedy Search takes less time than Random and Constructive Heuristic is because that Greedy Search needs fewer steps to finish the loading).

5.3 Ablation Study

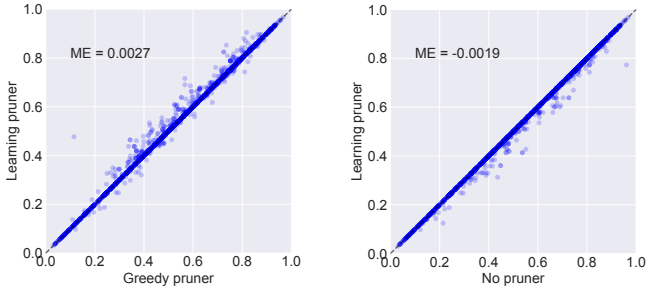
5.3.1 Does the CNN modeling methods work well? To find out how well the CNN works for modeling states, we design a simplified network to estimate loading rates of states. The simplified network is almost the same as the design in the state representation, as shown in Figure 4, but the network structures following the concatenate layer are different. For the simplified network, after concatenating tensors, we normalize the output which is followed by a fully-connected layer of 32 units with the ReLU activation. Then we continue to normalize the output and use a fully-connected layer of 1 unit with the Sigmoid activation to output a loading rate value. Besides, the simplified network does not need the mask output. The workflow of sample collecting and model training is similar to the pruning network. A sample consists of a state as features and a calculated loading rate as its label. During the algorithm, a time-consuming method is designed to calculate near-optimal loading rates for states. The states and their corresponding near-optimal loading rates are collected as samples. The input of the simplified network is a state and the output is a decimal between 0 and 1 which means the loading rate of the input state. We use the mean squared error as the loss function and the root mean squared error as the metric. Other compiling settings are the same as the pruning network. The prediction results of states in the test set are shown in Figure 7, in which the X axis represents the near-optimal loading rate (i.e., the ground truth), and the Y axis is the loading rate predicted by the network or calculated by simulation loading. For simulation loading is almost impossible to get better loading rates than near-optimal values as a *lower bound*, the green points in Figure 7(a) do not appear in the upper left. We evaluate the performance of simulation loading and network estimation by Root Mean Squared Error (RMSE) compared with the ground truth, respectively. Given the ground truth y_t and the predicted loading rate $\hat{y_t}$ for state t in the test set Q, RMSE calculates the standard deviation of the residuals over all states, as shown in Eq. 2.

$$RMSE = \sqrt{\frac{1}{|Q|}\sum_{t \in Q}(y_t - \hat{y_t})^2} \tag{2}$$

According to the RMSE metric, networks perform similarly to the simulation loading when predicting loading rates. But networks do not need to explicitly construct packing plans, which is time-saving. In brief, the comparison results show that our CNN modeling method can represent states in tree search algorithms well.

(a) Simulation loading **(b) Network estimation**

Figure 7: Prediction accuracy of simulation loading and network estimation



(a) Greedy pruner **(b) No pruner**

Figure 8: Prediction accuracy of the learning pruner

Table 2: Comparison results of tree search with different pruners

	Loading rate (%)	Time (s)	Best loading(%)[2]
Greedy pruner	75.05	**24.74**	79.69
Learning pruner	76.69	27.32	98.44
No pruner	**76.72**	43.46	**98.75**
Gap[1]	0.04%	37.14%	0.31%

[1] Gap = (No pruner - Learning pruner) / No pruner.
[2] A best packing plan may be obtained by more than one method, so its sum of all methods may be over 100%.

5.3.2 Does the pruning network bring benefits? We believe that the more proper we pick out states in the intermediate process, the better the final solution performs. Therefore, to observe the effect of the pruner network thoroughly, we need to not only check the final solution performance of tree search algorithms but also the effect of inner selection with different pruners. When facing the same m states in the lookahead process, we record the maximum loading rate in the three groups of selected states. The results are reported in Table 2. "Gap (%)" represents the gap between tree search algorithms with learning pruner and without pruner, i.e., (No pruner - Learning pruner) / No pruner. The methods involved in comparison are as follows.

- Greedy pruner: The tree search algorithm with a greedy pruner. Spaces are sorted by the distance from the origin, blocks are sorted by size. It selects top n states satisfying all constraints.
- Learning pruner: The tree search algorithm with a learning pruner, i.e., DDTS proposed in this paper. It selects n states guided by the pruning network.
- No pruner: The tree search algorithm with a "perfect" pruner. It explores all m states to guarantee the state with the best loading rate can be selected.

When facing the same m states in lookahead process, we record the maximum loading rate of each selected state group by each pruner. We call the m states in lookahead process as a lookahead state h. We evaluate the performance of the learning pruner by Mean Error (ME) compared with the greedy pruner and the no pruner. We denote the best loading rate by the learning pruner as

y_h, and the loading rate by greedy pruner or with pruner as y'_h for h in the test set $\mathcal{H}$. ME calculates the mean error over all lookahead states in $\mathcal{H}$, as shown in Eq. 3.

$$ME = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} (y_h - y'_h) \tag{3}$$

If $ME > 0$, the learning pruner can capture more critical states than the pruner corresponding to y'_h when facing the lookahead state h. As shown in Figure 8(a), the learning pruner can capture states with greater loading rates than the greedy pruner. Compared with no pruner in Figure 8(b), the learning pruner misses few best states, which is because our pruning network is not "perfect". In addition to observing the effect from the inner lookahead procedures, we also compare the final loading rates. As shown in Table 2, the algorithm with learning pruners can achieve almost the same performance as the algorithm without pruners but in only half computational time. Algorithms with greedy pruners and learning pruners take about the same time, but the latter performs much better. In summary, speeding up with learning pruners can save 37.14% time with only 0.04% performance loss compared to no pruner version.

5.3.3 Does the constraint scale affect the algorithm? The large-scale constraint is one of the most important challenges in the practical scene, and here we conduct the effect of constraint scale on DDTS. As given in Table 3, we reduce the number of constraints to 20 by ignoring the packing material constraint, orientation constraint, weight constraint, etc. Compared with the results of the full constraint version in Table 1, the loading rates of all methods have increased by about 7%. Beam Search is closed to Greedy Search in full constraint version, which is far worse than DDTS. However, in the partial constraint version, Beam Search is closed to DDTS and far higher than Greedy Search. It illustrates that with the constraint scale increases, Beam Search deteriorates rapidly, while DDTS can still perform extremely well. The comparison loading rate results of the two versions are presented in lines of Figure 9, and the box whiskers plot shows the loading rate distribution of all orders for the partial constraint version.

Table 3: Comparison results for partial constraint

	Loading rate (%)	Time (s)	Best Loading (%)[2]
Random	61.37	8.38	0
Constructive Heuristic	79.88	12.21	32.48
Greedy Search	81.43	**4.35**	46.15
Beam Search	83.13	134.77	69.23
DDTS	**83.17**	50.13	**78.63**
Comp[1]	0.05%	-62.80%	13.58%

[1] Comp = (DDTS - Beam Search) / Beam Search.
[2] A best packing plan may be obtained by more than one method, so its sum of all methods may be over 100%.

5.3.4 Does the item scale affect the algorithm? All previous experiments were conducted on large-scale orders. In this section, we involve in small-scale and medium-scale orders to explore the effect of item scale on algorithms. The small-scale dataset is of 0 to 200 items per order and the medium-scale dataset is of 200 to 500 items per order. The loading rate results of all methods are provided in Figure 10. Except for Random, other algorithms have almost the same trend on three datasets. That is, our DDTS has good generalization ability on different item scales.

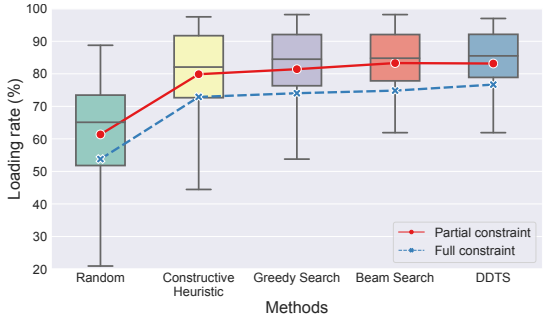


Figure 9: Comparison of loading rates for partial constraint and full constraint, on 5 algorithms, over 500 test orders

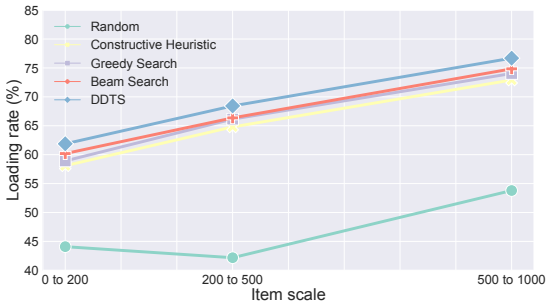


Figure 10: Comparison of loading rates on datasets of 0 to 200 items, 200 to 500 items and 500 to 1000 items

6 DEPLOYMENT

Our algorithm has been deployed in Huawei Logistics System with distributed mode, which is used in Huawei Global Supply Centers and warehouses. Since the algorithm went online in March 2021, it has got huge economic value and improved customer satisfaction. The average container loading rate has increased by 3%, which could save millions of dollars every year. The time consumption was reduced by 20%, making it more convenient for users to use in real time. Also, The accuracy rate of the loading plan has increased from 80% to 95%, which means 95% of the loading plans are consistent with the final implementation, with no need for manual modification.

7 DISCUSSION AND CONCLUSION

7.1 Lessons learned

To make sure the effectiveness of the deep learning guidance embedded inside the tree search algorithm, we have to ensure that 1) the improvement of the internal modules could lead to a better overall outcome; 2) both the internal modules and the overall outcome are improved. Quick preliminary experiments to explore the improvement in intermediate procedures are necessary, as conducted in Section 5.3.1 and Section 5.3.2.

We use a pruning network to reduce the number of branches to be evaluated in the lookahead process. However, [26], who also used lookahead strategies to score and select states, used a critic network to speed up the process of evaluating branches. It is not practical to our problem due to the large scale and complicated constraints. Although from Figure 7(b) we know that the loading rate could be estimated well by our simplified network, it is hard to distinguish the states at the same level for we collect data during the whole tree search without paying attention to distinguish the states at the same level. Thus, the simulation loading is still necessary to evaluate branches accurately in our problem. Our experiments indicate that the pruning network can capture the difference among the states at the same level by inputting them together and reduce the candidate states, and the simulation loading further selects the best one. For application, the idea of the pruning network can be used in other problems that also need to reduce the number of branches for acceleration, especially when the constraint scale is large and the direct evaluation is hard. The tree search algorithm is widely known, such as Beam Search, Branch and Bound and Monte Carlo Tree Search, as a powerful weapon to solve various decision-making problems in practical settings. Our idea can be tried in these scenarios.

Moreover, we have tried to involve other features when encoding items, but these three features in our paper are the most pivotal and convenient for this problem.

7.2 Conclusion and future Work

We address the practical 3D-BPP in Huawei and propose a data-driven tree search algorithm (DDTS) to solve it. Block building and space generation algorithms are used to deal with numerous cargoes. A lookahead tree search algorithm is used to explore diverse solutions. A CNN-based pruning network trained with historical records is embedded into the tree search to guide pruning. Computational experiments on real-world datasets show that DDTS outperforms the state-of-the-art approach with a loading rate improvement of 2.47%. Moreover, speeding up with pruning networks saves 37.14% time with only 0.04% performance loss. Since DDTS was deployed in Huawei logistics system, the average loading rate of orders has increased by 3%, saving significant costs. To the best of our knowledge, we are the first to embed pruning networks into tree search for the large-scale 3D-BPP. In the future, we will improve our current algorithm and try applying it to other scenarios.

8 ACKNOWLEDGMENTS

This research was supported by the Young Elite Scientists Sponsorship Program by China Association for Science and Technology (Grants No. 2019QNRC001).

REFERENCES

[1] Ignacio Araya, Keitel Guerrero, and Eduardo Nuñez. 2017. VCS: A new heuristic function for selecting boxes in the single container loading problem. *Computers & Operations Research* 82 (June 2017), 27–35. https://doi.org/10.1016/j.cor.2017.01.002

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

[3] Lei Chen, Xialiang Tong, Mingxuan Yuan, Jia Zeng, and Lei Chen. 2019. A Data-Driven Approach for Multi-level Packing Problems in Manufacturing Industry. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. Association for Computing Machinery, New York, NY, USA, 1762–1770. https://doi.org/10.1145/3292500.3330708

[4] A.Paul Davies and Eberhard E. Bischoff. 1999. Weight distribution considerations in container loading. *European Journal of Operational Research* 114, 3 (May 1999), 509–527. https://doi.org/10.1016/S0377-2217(98)00139-8

[5] B. M. Domingo, S.G. Ponnambalam, and G. Kanagaraj. 2012. Particle Swarm Optimization for the single container loading problem. In *2012 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE, Coimbatore, India, 1–6. https://doi.org/10.1109/ICCIC.2012.6510262

[6] György Dósa and Jiri Sgall. 2013. First Fit bin packing: A tight analysis. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 20)*, Natacha Portier and Thomas Wilke (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 538–549. https://doi.org/10.4230/LIPIcs.STACS.2013.538

[7] Lu Duan, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Jiangwen Wei, and Yinghui Xu. 2019. A Multi-task Selected Learning Approach for Solving 3D Flexible Bin Packing Problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 1386–1394.

[8] José Fernando Gonçalves and Mauricio G. C. Resende. 2012. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research* 39, 2 (Feb. 2012), 179–190. https://doi.org/10.1016/j.cor.2011.03.009

[9] He He, Hal Daume III, and Jason M. Eisner. 2014. Learning to search in branch and bound algorithms. *Advances in neural information processing systems* 27 (2014), 3293–3301.

[10] André Hottung, Shunji Tanaka, and Kevin Tierney. 2020. Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research* 113 (Jan. 2020), 104781. https://doi.org/10.1016/j.cor.2019.104781

[11] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. 2017. Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method. *arXiv:1708.05930 [cs]* (Aug. 2017). http://arxiv.org/abs/1708.05930 arXiv: 1708.05930.

[12] Leonardo Junqueira, Reinaldo Morabito, and Denise Sato Yamashita. 2012. Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers and Operations Research* 39, 1 (Jan. 2012), 74–85. https://doi.org/10.1016/j.cor.2010.07.017

[13] Leonardo Junqueira, Reinaldo Morabito, and Denise Sato Yamashita. 2012. MIP-based approaches for the container loading problem with multi-drop constraints. *Annals of Operations Research* 199, 1 (Oct. 2012), 51–75. https://doi.org/10.1007/s10479-011-0942-z

[14] Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. 2016. Learning to branch in Mixed Integer Programming. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, Phoenix, Arizona, 724–731.

[15] Elias B. Khalil, Bistra Dilkina, George L. Nemhauser, Shabbir Ahmed, and Yufen Shao. 2017. Learning to run heuristics in tree search. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press, Melbourne, Australia, 659–666.

[16] Alexandre Laterre, Yunguan Fu, Mohamed Khalil Jabri, Alain-Sam Cohen, David Kas, Karl Hajjar, Hui Chen, Torbjørn S. Dahl, Amine Kerkeni, and Karim Beguir. 2019. Ranked Reward: Enabling Self-Play Reinforcement Learning for Bin packing. (2019).

[17] Xijun Li, Mingxuan Yuan, Di Chen, Jianguo Yao, and Jia Zeng. 2018. A Data-Driven Three-Layer Algorithm for Split Delivery Vehicle Routing Problem with 3D Container Loading Constraint. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. Association for Computing Machinery, New York, NY, USA, 528–536. https://doi.org/10.1145/3219819.3219872

[18] Silvano Martello, David Pisinger, and Daniele Vigo. 2000. The Three-Dimensional Bin Packing Problem. *Operations Research* 48, 2 (April 2000), 256–267. https://doi.org/10.1287/opre.48.2.256.12386

[19] Ana Moura and Jose Fernando Oliveira. 2005. A GRASP approach to the container-loading problem. *IEEE Intelligent Systems* 20, 4 (2005), 50–57. Publisher: IEEE.

[20] F. Parreño, R. Alvarez-Valdes, J. F. Oliveira, and J. M. Tamarit. 2010. Neighborhood structures for the container loading problem: a VNS implementation. *Journal of Heuristics* 16, 1 (Feb. 2010), 1–22. https://doi.org/10.1007/s10732-008-9081-3

[21] F. Parreño, R. Alvarez-Valdes, J. M. Tamarit, and J. F. Oliveira. 2008. A Maximal-Space Algorithm for the Container Loading Problem. *INFORMS Journal on Computing* 20, 3 (Aug. 2008), 412–422. https://doi.org/10.1287/ijoc.1070.0254

[22] Guntram Scheithauer. 1992. Algorithms for the container loading problem. In *Operations Research Proceedings 1991*. Springer, 445–452.

[23] Richa Verma, Aniruddha Singhal, Harshad Khadilkar, Ansuma Basumatary, Siddharth Nayak, Harsh Vardhan Singh, Swagat Kumar, and Rajesh Sinha. 2020. A Generalized Reinforcement Learning Algorithm for Online 3D Bin-Packing. *arXiv preprint arXiv:2007.00463* (2020).

[24] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. 2021. Combinatorial learning of graph edit distance via dynamic embedding. In

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 5241–5250.

[25] Canrong Zhang, Hao Guan, Yifei Yuan, Weiwei Chen, and Tao Wu. 2020. Machine learning-driven algorithms for the container relocation problem. *Transportation Research Part B: Methodological* 139 (2020), 102–131. Publisher: Elsevier.

[26] Hang Zhao, Qijin She, Chenyang Zhu, Yin Yang, and Kai Xu. 2021. Online 3D Bin Packing with Constrained Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 1 (May 2021), 741–749. https://ojs.aaai.org/index.php/AAAI/article/view/16155 Number: 1.

[27] Xiaozhou Zhao, Julia A. Bennell, Tolga Bektaş, and Kath Dowsland. 2016. A comparative review of 3D container loading algorithms. *International Transactions in Operational Research* 23, 1-2 (2016), 287–320. Publisher: Wiley Online Library.

[28] Wenbin Zhu, Wee-Chong Oon, Andrew Lim, and Yujian Weng. 2012. The six elements to block-building approaches for the single container loading problem. *Applied Intelligence* 37, 3 (2012), 431–445. Publisher: Kluwer Academic Publishers Norwell, MA, USA.

A BLOCK BUILDING ALGORITHM

A block is a cuboid that consists of items. The block set $\mathcal{B}$ consists of all possible combinations of the remaining items $\mathcal{I}$. Given the remaining item set $\mathcal{I}$, the block building algorithm outputs the corresponding block set $\mathcal{B}$. The pseudocode is given in Algorithm 2.

Algorithm 2 Block Building

Input: The remaining item index set $\mathcal{I}$; All permitted orientations O_i for each item $i \in \mathcal{I}$.
Output: Block set $\mathcal{B}$.

Let $\mathcal{B} = \{(item_i, o_i) | i \in \mathcal{I}, o_i \in O_i\}$;
Let $\mathcal{B}' = \mathcal{B}$;
while $|\mathcal{B}| < max\_bk$ **do**
  Let $\mathcal{N} = \emptyset$;
  for all block b_1 in $\mathcal{B}'$ **do**
    for all block b_2 in $\mathcal{B}$ **do**
      for all axis in $\{X, Y, Z\}$ **do**
        Combine b_1 and b_2 along the axis to obtain b_3;
        if b_3 is cuboid and $b_3 \notin \mathcal{N}$ **then**
          Add b_3 to $\mathcal{N}$;
        end if
      end for
    end for
  end for
  if $\mathcal{N} = \emptyset$ **then**
    Break;
  end if
  Let $\mathcal{B} = \mathcal{B} \cup \mathcal{N}$;
  Let $\mathcal{B}' = \mathcal{N}$;
end while
return $\mathcal{B}$.

B SPACE GENERATING ALGORITHM

After an item is packed at a corner of a space, at most three new maximal spaces are generated [1]. To generate maximal spaces, we first need to find the smallest cuboid that can encase all packed items in the container. The smallest cuboid has three faces that do not coincide with the container. We cut along one face at a time to get a space, and three overlapping space cuboids are generated, as shown in Figure 2(b). The pseudocode is provided in Algorithm 3.

Algorithm 3 Space Generating

Input: Original space s with maximum and minimum coordinates $(x^s_{min}, x^s_{max}, y^s_{min}, y^s_{max}, z^s_{min}, z^s_{max})$; Index set $\mathcal{I}$ of the packed item in s; The corresponding maximum and minimum coordinates of the items $\left\{(x^i_{min}, x^i_{max}, y^i_{min}, y^i_{max}, z^i_{min}, z^i_{max}) | i \in \mathcal{I}\right\}$.
Output: New space set $\mathcal{S}$.

1: Generate cuboid C, with the corresponding maximum and minimum coordinates: $x^C_{min} = x^s_{min}$, so do y^C_{min}, z^C_{min}; $x^C_{max} = \min\left\{\max\left\{x^i_{max} | i \in \mathcal{I}\right\}, x^s_{max}\right\}$, so do y^C_{max}, z^C_{max};
2: Generate the largest free space along three dimensions, which might mutually overlaps. Along x-axis, the new free space s_x has maximum and minimum coordinates $(x^{s_x}_{min}, x^{s_x}_{max}, y^{s_x}_{min}, y^{s_x}_{max}, z^{s_x}_{min}, z^{s_x}_{max}) = (x^C_{max}, x^s_{max}, y^s_{min}, y^s_{max}, z^s_{min}, z^s_{max})$; Generate new spaces generated along y-axis s_y and z-axis s_z in similar way;
3: Add new spaces s_x, s_y, s_z in $\mathcal{S}$;
4: **return** $\mathcal{S}$.