Online 3D Bin Packing with Constrained Deep Reinforcement Learning

Hang Zhao[1], Qijin She[1], Chenyang Zhu[1], Yin Yang[2], Kai Xu[1*]
[1]National University of Defense Technology
[2]Clemson University

Abstract

We solve a challenging yet practically useful variant of 3D Bin Packing Problem (3D-BPP). In our problem, the agent has limited information about the items to be packed into a single bin, and an item must be packed immediately after its arrival without buffering or readjusting. The item's placement also subjects to the constraints of order dependence and physical stability. We formulate this *online 3D-BPP* as a constrained Markov decision process (CMDP). To solve the problem, we propose an effective and easy-to-implement constrained deep reinforcement learning (DRL) method under the actor-critic framework. In particular, we introduce a *prediction-and-projection* scheme: The agent first predicts a feasibility mask for the placement actions as an auxiliary task and then uses the mask to modulate the action probabilities output by the actor during training. Such supervision and projection facilitate the agent to learn feasible policies very efficiently. Our method can be easily extended to handle lookahead items, multi-bin packing, and item re-orienting. We have conducted extensive evaluation showing that the learned policy significantly outperforms the state-of-the-art methods. A preliminary user study even suggests that our method might attain a human-level performance.

Introduction

As a classic NP-hard problem, the bin packing problem (1D-BPP) seeks for an assignment of a collection of items with various weights to bins. The optimal assignment houses all the items with the fewest bins such that the total weight of items in a bin is below the bin's capacity c (Korte and Vygen 2012). In its 3D version i.e., 3D-BPP (Martello, Pisinger, and Vigo 2000), an item i has a 3D "weight" corresponding to its length, l_i, width w_i, and height h_i. Similarly, c is also in 3D including $L \geq l_i$, $W \geq w_i$, and $H \geq h_i$. It is assumed that $l_i, w_i, h_i, L, W, H \in Z^+$ are positive integers. Given the set of items $\mathcal{I}$, we would like to pack all the items into as few bins as possible. Clearly, 1D-BPP is a special case of its three dimensional counter part – as long as we constrain $h_i = H$ and $w_i = W$ for all $i \in \mathcal{I}$, a 3D-BPP instance can be relaxed to a 1D-BPP. Therefore, 3D-BPP is also highly NP-hard (Man Jr, Garey, and Johnson 1996).

Regardless of its difficulty, the bin packing problem turns out to be one of the most needed academic problems as many real-world challenges could be much more efficiently handled if we have a good solution to it. A good example is large-scale parcel packaging in modern logistics systems (Figure. 1), where parcels are mostly in regular cuboid shapes, and we would like to collectively pack them into rectangular bins of the standard dimension. Maximizing the storage use of bins effectively reduces the cost of inventorying, wrapping, transportation, and warehousing. While being strongly NP-hard, 1D-BPP has been extensively studied. With the state-of-the-art computing hardware, big 1D-BPP instances (with about $1,000$ items) can be exactly solved within tens of minutes (Delorme, Iori, and Martello 2016) using e.g., integer linear programming (ILP) (Schrijver 1998), and good approximations can be obtained within milliseconds. On the other hand 3D-BPP, due to the extra complexity imposed, is relatively less explored. Solving a 3D-BPP of moderate size exactly (either using ILP or branch-and-bound) is much more involved, and we still have to resort to heuristic algorithms (Crainic, Perboli, and Tadei 2008; Karabulut and İnceoğlu 2004).

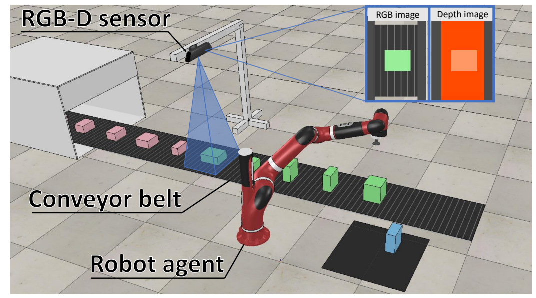

Figure 1: Online 3D-BPP, where the agent observes only a limited numbers of lookahead items (shaded in green), is widely useful in logistics, manufacture, warehousing etc.

Most existing 3D-BPP literature assumes that the information of all items is known while does not take physical stability into consideration, and the packing strategies allow backtracking i.e., one can always repack an item from the bin in order to improve the current solution (Martello, Pisinger, and Vigo 2000). In practice however, we do not know the information of all items. For instance see Figure 1, where a robot works beside a bin, and a conveyor forwards parcels sequentially. The robot may only have the vision of several upcoming items (similar to Tetris), and an item must be packed within a given time period after its arrival.

[*]Hang Zhao and Qijin She are co-first authors. Kai Xu is the corresponding author (kevin.kai.xu@gmail.com).

It is costly and inefficient if the robot frequently unloads and readjusts parcels in packed bins. Such constraints further complicate 3D-BPP in its real-world applications.

As an echo to those challenges, we design a deep reinforcement learning algorithm for 3D-BPP. To maximize the applicability, we carefully accommodate restrictions raised in its actual usage. For instance, we require item placement satisfying order dependence and not inducing instable stacking. An item is immediately packed upon its arrival, and no adjustment will be permitted after it is packed. To this end, we opt to formulate our problem as a constrained Markov decision process (CMDP) (Altman 1999) and propose a constrained DRL approach based on the on-policy actor-critic framework (Mnih et al. 2016; Wu et al. 2017).

In particular, we introduce a *prediction-and-projection* scheme for the training of constrained DRL. The agent first predicts a feasibility mask for the placement actions as an auxiliary task. It then uses the mask to modulate the action probabilities output by the actor. These supervision and projection enable the agent to learn feasible policy very efficiently. We also show that our method is general with the ability to handle lookahead items, multi-bin packing, and item re-orienting. With a thorough test and validation, we demonstrate that our algorithm outperforms existing methods by a noticeable margin. It even demonstrates a human-level performance in a preliminary user study.

Related Work

1D-BPP is one of the most famous problems in combinatorial optimization, and related literature dates back to the sixties (Kantorovich 1960). Many variants and generalizations of 1D-BPP arise in practical contexts such as the cutting stock problem (CSP), in which we want to cut bins to produce desired items of different weights, and minimize the total number of bins used. A comprehensive list of bibliography on 1D-BPP and CSP can be found in (Sweeney and Paternoster 1992). Knowing to be strongly NP-hard, most existing literature focuses on designing good heuristic and approximation algorithms and their worst-case performance analysis (Coffman, Garey, and Johnson 1984). For example, the well-known greedy algorithm, the next fit algorithm (NF) has a linear time complexity of $O(N)$ and its *worst-case performance ratio* is 2 i.e. NF needs at most twice as many bins as the optimal solution does (De La Vega and Lueker 1981). The first fit algorithm (FF) allows an item to be packed into previous bins that are not yet full, and its time complexity increases to $O(N \log N)$. The best fit algorithm (BF) aims to reduce the residual capacity of all the non-full bins. Both FF and BF have a better worst-case performance ratio of $\frac{17}{10}$ than NF (Johnson et al. 1974). Pre-sorting all the items yields the off-line version of those greedy strategies sometimes also known as the *decreasing* version (Martello 1990). While straightforward, NF, FF, and BF form a foundation of more sophisticated approximations to 1D-BPP (e.g. see (Karmarkar and Karp 1982)) or its exact solutions (Martello and Toth 1990; Scholl, Klein, and Jürgens 1997; Labbé, Laporte, and Martello 1995; Delorme, Iori, and Martello 2016). We also refer the reader to `BPPLib` library (Delorme, Iori, and Martello 2018), which includes the implementation of most known algorithms for the 1D-BPP problem.

2D- and 3D-BPP are natural generalization of the original BPP. Here, an item does not only have a scalar-valued weight but a high-dimension size of width, height, and/or depth. The main difference between 1D- and 2D-/3D- packing problems is the verification of the feasibility of the packing, i.e. determining whether an accommodation of the items inside the bin exists such that items do not interpenetrate and the packing is within the bin size. The complexity and the difficulty significantly increase for high-dimension BPP instances. In theory, it is possible to generalize exact 1D solutions like MTP (Martello and Toth 1990) or branch-and-bound (Delorme, Iori, and Martello 2016) algorithms to 2D-BPP (Martello and Vigo 1998) and 3D-BPP (Martello, Pisinger, and Vigo 2000). However according to the timing statistic reported in (Martello, Pisinger, and Vigo 2000), exactly solving 3D-BPP of a size matching an actual parcel packing pipeline, which could deal with tens of thousand parcels, remains infeasible. Resorting to approximation algorithms is a more practical choice for us. Hifi et al. (2010) proposed a mixed linear programming algorithm for 3D-BPP by relaxing the integer constraints in the problem. Crainic et al. (2008) refined the idea of corner points (Martello, Pisinger, and Vigo 2000), where an upcoming item is placed to the so-called *extreme points* to better explore the un-occupied space in a bin. Heuristic local search iteratively improves an existing packing by searching within a neighbourhood function over the set of solutions. There have been several strategies in designing fast approximate algorithms, e.g., guided local search (Faroe, Pisinger, and Zachariasen 2003), greedy search (De Castro Silva, Soma, and Maculan 2003), and tabu search (Lodi, Martello, and Vigo 1999; Crainic, Perboli, and Tadei 2009). Similar strategy has also been adapted to Online BPP (Ha et al. 2017; Wang et al. 2016). In contrast, genetic algorithms leads to better solutions as a global, randomized search (Li, Zhao, and Zhang 2014; Takahara and Miyamoto 2005).

Deep reinforcement learning (DRL) has demonstrated tremendous success in learning complex behaviour skills and solving challenging control tasks with high-dimensional raw sensory state-space (Lillicrap et al. 2015; Mnih et al. 2015, 2016). The existing research can largely be divided into two lines: on-policy methods (Schulman et al. 2017; Wu et al. 2017) and off-policy ones (Mnih et al. 2015; Wang et al. 2015; Barth-Maron et al. 2018). On-policy algorithms optimize the policy with agent-environment interaction data sampled from the current policy. While lacking the ability of reusing old data makes them less data efficient, updates calculated by on-policy data lead to stable optimization. In contrast, off-policy methods are more data-efficient but less stable. In our problem, agent-environment interaction data is easy to obtain (in 2000FPS), thus data efficiency is not our main concern. We base our method on the on-policy actor-critic framework. In addition, we formulate online 3D-BPP as constrained DRL and solve it by projecting the trajectories sampled from the actor to the constrained state-action space, instead of resorting to more involved constrained policy optimization (Achiam et al. 2017).

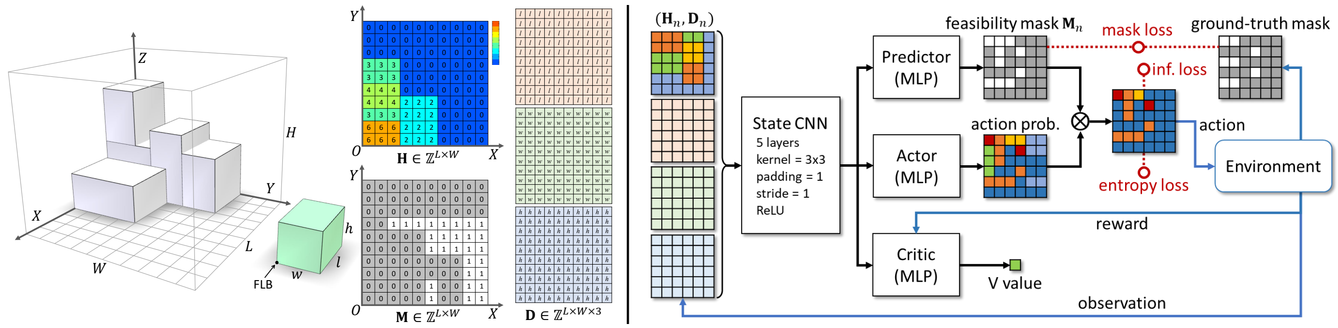

Figure 2: Left: The environment state of the agent includes the configuration of the bin (the grey boxes) and the size of the next item to be packed (green box). The bin configuration is parameterized as a height map $\mathbf{H}$ over a $L \times W$ grid. The feasibility mask $\mathbf{M}$ is a binary matrix of size $L \times W$ indicating the placement feasibility at each grid cell. The three dimensions of the next item are stored into a $L \times W \times 3$ tensor $\mathbf{D}$. Right: The network architecture (the three losses other than the standard actor and critic losses are shown in red color).

RL for combinatorial optimization has a distinguished history (Gambardella and Dorigo 1995; Zhang and Dietterich 2000) and is still an active direction with especially intensive focus on TSP (Bello et al. 2016). Early attempts strive for heuristics selection using RL (Nareyek 2003). Bello et al. (2016) combined *RL pretraining* and *active search* and demonstrated that RL-based optimization outperforms supervised learning framework when tackling NP-hard combinatorial problems. Recently, Hu et al. (2017) proposed a DRL solution to 3D-BPP. Laterre et al. (2018) introduced a rewarding strategy based on self-play. Different from ours, these works deal with an offline setting where the main goal is to find an optimal sequence of items inspired by the Pointer Network (Vinyals, Fortunato, and Jaitly 2015).

Method

In online 3D-BPP, the agent is agnostic on l_i, w_i or h_i of all the items in $\mathcal{I}$ – only immediately incoming ones $\mathcal{I}_o \subset \mathcal{I}$ are observable. As soon as an item arrives, we pack it into the bin, and no further adjustment will be applied. As the complexity of BPP decreases drastically for bigger items, we further constrain the sizes of all items to be $l_i \leq L/2$, $w_i \leq W/2$, and $h_i \leq H/2$. We start with our problem statement under the context of DRL and the formulation based on constrained DRL. We show how we solve the problem via predicting action feasibility in the actor-critic framework.

Problem Statement and Formulation

The 3D-BPP can be formulated as a Markov decision process, which is a tuple of $(\mathcal{S}, \mathcal{A}, P, R)$. $\mathcal{S}$ is the set of environment states; $\mathcal{A}$ is the action set; $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function; $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition probability function. $P(s'|s, a)$ gives the probability of transiting from s to s' for given action a. Our method is model-free since we do not learn $P(s'|s, a)$. The policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a map from states to probability distributions over actions, with $\pi(a|s)$ denoting the probability of selecting action a under state s. For DRL, we seek for a policy π to maximize the accumulated discounted reward, $J(\pi) = E_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$. Here, $\gamma \in [0, 1]$ is the discount factor, and $\tau = (s_0, a_0, s_1, \ldots)$ is a trajectory sampled based on the policy π.

The environment state of 3D-BPP is comprised of two parts: the current configuration of the bin and the coming items to be placed. For the first part, we parameterize the bin through discretizing its bottom area as a $L \times W$ regular grid along length (X) and width (Y) directions, respectively. We record at each grid cell the current height of stacked items, leading to a *height map* $\mathbf{H}_n$ (see Figure 2). Here, the subscript n implies n is the next item to be packed. Since all the dimensions are integers, $\mathbf{H}_n \in \mathbb{Z}^{L\times W}$ can be expressed as a 2D integer array. The dimensionality of item n is given as $\mathbf{d}_n = [l_n, w_n, h_n]^\top \in \mathbb{Z}^3$. Working with integer dimensions helps to reduce the state/action space and accelerate the policy learning significantly. A spatial resolution of up to 30×30 is sufficient in the real scenarios. Putting together, the current environment state can be written as $s_n = \{\mathbf{H}_n, \mathbf{d}_n, \mathbf{d}_{n+1}, ..., \mathbf{d}_{n+k-1}\}$. We first consider the case where $k = |\mathcal{I}_o| = 1$, and name this special instance as BPP-1. In other words, BPP-1 only considers the immediately coming item n i.e., $\mathcal{I}_o = \{n\}$. We then generalize it to BPP-k with $k > 1$ afterwards.

In BPP-1, the agent places n's front-left-bottom (FLB) corner (Figure 2 (left)) at a certain grid point or the loading position (LP) in the bin. For instance, if the agent chooses to put n at the LP of (x_n, y_n). This action is represented as $a_n = x_n + L \cdot y_n \in \mathcal{A}$, where the action set $\mathcal{A} = \{0, 1, \ldots, L \cdot W - 1\}$. After a_n is executed, $\mathbf{H}_n$ is updated by adding h_n to the maximum height over all the cells covered by n: $\mathbf{H}'_n(x, y) = h_{\max}(x, y) + h_n$ for $x \in [x_n, x_n + l_n], y \in [y_n, y_n + w_n]$, with $h_{\max}(x, y)$ being the maximum height among those cells. The state transition is deterministic: $P(\mathbf{H}|\mathbf{H}_n, a_n) = 1$ for $\mathbf{H} = \mathbf{H}'_n$ and $P(\mathbf{H}|\mathbf{H}_n, a_n) = 0$ otherwise.

During packing, the agent needs to secure enough space in the bin to host item n. Meanwhile, it is equally important to have n statically equilibrated by the underneath at the LP so that all the stacking items are physically stable. Evaluating the physical stability at a LP is involved, taking into account of n's center of mass, moment of inertia, and rotational stability (Goldstein, Poole, and Safko 2002). All of them are normally unknown as the mass distribution differs among items. To this end, we employ a conservative and simplified criterion. Specifically, a LP is considered *feasible* if it not only provides sufficient room for n but also satisfies any of following conditions with n placed: 1) over 60% of n's bottom area and all of its four bottom corners are supported by existing items; or 2) over 80% of n's bottom area and three out of four bottom corners are supported; or 3) over 95% of n's bottom area is supported. We store the feasibility of all the LPs for item n with a *feasibility mask* $\mathbf{M}_n$, an $L \times W$ binary matrix (also see Figure 2).

Since not all actions are allowed, our problem becomes a constrained Markov decision processes (CMDP) (Altman 1999). Typically, one augments the MDP with an auxiliary cost function $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ mapping state-action tuples to costs, and require that the expectation of the accumulated cost should be bounded by c_m: $J_C(\pi) = E_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma_C^t C(s_t, a_t)] \leq c_m$.

Several methods have been proposed to solve CMDP based on e.g., algorithmic heuristics (Uchibe and Doya 2007), primal-dual methods (Chow et al. 2017), or constrained policy optimization (Achiam et al. 2017). While these methods are proven effective, it is unclear how they could fit for 3D-BPP instances, where the constraint is rendered as a discrete mask. In this work, we propose to exploit the mask $\mathbf{M}$ to guide the DRL training to enforce the feasibility constraint without introducing excessive training complexity.

Network Architecture

We adopt the actor-critic framework with Kronecker-Factored Trust Region (ACKTR) (Wu et al. 2017). It iteratively updates an actor and a critic module jointly. In each iteration, the actor learns a policy network that outputs the probability of each action (i.e., placing n at the each LP). The critic trains a state-value network producing the value function. We find through experiments that on-policy methods (such as ACKTR) lead to better performance than off-policy ones like SAC (Haarnoja et al. 2018); *see a comparison in the supplemental material.*

In the original ACKTR framework, both actor and critic networks take the raw state directly as input. In our implementation however, we devise a CNN, named *state CNN*, to encode the raw state vector into features. To facilitate this, we "stretch" $\mathbf{d}_n$ into a three-channel tensor $\mathbf{D}_n \in \mathbb{Z}^{L\times W\times 3}$ so that each channel of $\mathbf{d}_n$ spans a $L \times W$ matrix with all of its elements being l_n, w_n or h_n, respectively (also see Figure 2). Consequently, state $s_n = (\mathbf{H}_n, \mathbf{D}_n)$ becomes a $L \times W \times 4$ array (Figure 2 (right)).

We define a simplistic step-wise reward as the volumetric occupancy introduced by the current item: $r_n = 10 \times l_n \cdot w_n \cdot h_n/(L \cdot W \cdot H)$ for item n. When the current item is not placeable, its reward is zero and the episode ends. While the feasibility mask saves the efforts of exploring invalid actions, this step-wise reward directs the agent to place as many items as possible. We find through comparison that this step-wise reward is superior than a termination one (e.g. the final space utilization); *see supplemental material.*

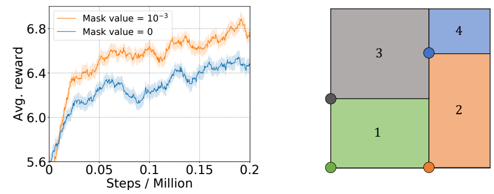

Figure 3: Left: Softening the mask-based modulation improves the training convergence. Right: A toy example of 2D cutting stock.

We devise a prediction-and-projection mechanism to enforce feasibility constraints. First, we introduce an independent multilayer perceptron module, namely the *mask predictor* (Figure 2 (right)), to predict the feasibility mask $\mathbf{M}_n$ for the item n. The predictor takes the state CNN features of the current state as the input and is trained with the ground-truth mask as the supervision. Next, we use the predicted mask to modulate the output, i.e., the probability distribution of the actions. In theory, if the LP at (x, y) is infeasible for n, the corresponding probability $P(a_n = x + L \cdot y|s_n)$ should be set to 0. However, we find that setting P to a small positive quantity like $\epsilon = 10^{-3}$ works better in practice – it provides a strong penalty to an invalid action but a smoother transformation beneficial to the network training (Figure 3 (left)). To further discourage infeasible actions, we explicitly minimize the summed probability at all infeasible LPs: $E_{inf} = \sum P(a_n = x + L \cdot y|s_n), \forall(x, y)|\mathbf{M}_n(x, y) = \epsilon$, which is plugged into the final loss function for training.

Our loss function is defined as:

$$L = \alpha \cdot L_{actor} + \beta \cdot L_{critic} + \lambda \cdot L_{mask} + \omega \cdot E_{inf} - \psi \cdot E_{entropy}. \tag{1}$$

Here, L_{actor} and L_{critic} are the loss functions used for training the actor and the critic, respectively. L_{mask} is the MSE loss for mask prediction. To push the agent to explore more LPs, we also utilize an action entropy loss $E_{entroy} = \sum_{\mathbf{M}_n(x,y)=1} -P(a_n|s_n) \cdot \log\big(P(a_n|s_n)\big)$. Note that the entropy is computed only over the set of all feasible actions whose LP satisfies $\mathbf{M}_n(x, y) = 1$. In this way, we stipulate the agent to explore only feasible actions. We find through experiments that the following weights lead to consistently good performance throughout our tests: $\alpha = 1$, $\beta = \lambda = 0.5$, and $\omega = \psi = 0.01$.

BPP-k with $k = |\mathcal{I}_o| > 1$

In a more general case, the agent receives the information of $k > 1$ *lookahead* items (i.e., from n to $n + k - 1$). Obviously, the additional items inject more information to the environment state, which should be exploited in learning the policy $\pi(a_n|\mathbf{H}_n, \mathbf{d}_n, ..., \mathbf{d}_{n+k-1})$. One possible solution is to employ sequential modeling of the state sequence $(\mathbf{d}_n, ..., \mathbf{d}_{n+k-1})$ using, e.g., recurrent neural networks. We found that, however, such state encoding cannot well inform the agent about the lookahead items during DRL training and yields limited improvement. Alternatively, we propose a search-based solution leveraging the height map $\mathbf{H}$ update and feasibility mask prediction.

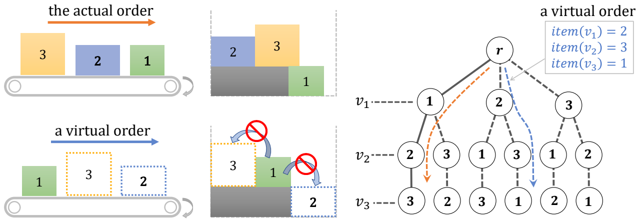

Figure 4: The permutation tree for $\mathcal{I}_o = \{1, 2, 3\}$. To find the best packing for item 1, our method explores different virtual placing orders satisfying the order dependence constraint, e.g., 1 cannot be placed on top of virtually placed 2 or 3.

The core idea is to condition the placement of the current item n on the next $k-1$ ones. Note that the actual placement of the k items still follows the order of arrival. To make the current placement account for the future ones, we opt to "hallucinate" the placement of future items through updating the height map accordingly. Conditioned on the virtually placed future items, the decision for the current item could be globally more optimal. However, such virtual placement must satisfy the *order dependence constraint* which stipulates that the earlier items should never be packed on top of the later ones. In particular, given two items p and q, $p < q$ in $\mathcal{I}_o$, if q is (virtually) placed before p, we require that the placement of p should be spatially independent to the placement of q. It means p can never be packed at any LPs that overlap with q. This constraint is enforced by setting the height values in $\mathbf{H}$ at the corresponding LPs to H, the maximum height value allowed: $\mathbf{H}_p(x, y) \leftarrow H$, for all $x \in [x_q, x_q + l_q]$ and $y \in [y_q, y_q + w_q]$. Combining explicit height map updating with feasibility mask prediction, the agent utilizes the trained policy with the order dependence constraint satisfied implicitly.

We opt to search for a better a_n through exploring the permutations of the sequence $(\mathbf{d}_n, ..., \mathbf{d}_{n+k-1})$. This amounts to a permutation tree search during which only the actor network test is conducted – no training is needed. Figure 4 shows a k-level permutation tree: A path $(r, v_1, v_2, ..., v_k)$ from the root to a leaf forms a possible permutation of the placement of the k items in $\mathcal{I}_o$, where r is the (empty) root node and let $item(v_i)$ represent the i-th item being placed in the permutation. Given two items $item(v_i) < item(v_j)$ meaning $item(v_i)$ arrives before $item(v_j)$ in the actual order. If $i > j$ along a permutation path, meaning that $item(v_j)$ is virtually placed before $item(v_i)$, we block the LPs corresponding to $item(v_j)$'s occupancy to avoid placing $item(v_i)$ on top of $item(v_j)$.

Clearly, enumerating all the permutations for k items quickly becomes prohibitive with an $O(k!)$ complexity. To make the search scalable, we adapt the Monte Carlo tree search (MCTS) (Silver et al. 2017) to our problem. With MCTS, the permutation tree is expanded in a priority-based fashion through evaluating how promising a node would lead to the optimal solution. The latter is evaluated by sampling a fixed number of paths starting from that node and computing for each path a value summing up the accumulated reward and the critic value ("reward to go") at the leaf (k-th level) node. After search, we choose the action a_n corresponding to the permutation with the highest path value. *Please refer to the supplemental material for more details on our adaptions of the standard MCTS.* MCTS allows a scalable lookahead for BPP-k with a complexity of $O(km)$ where m is the number of paths sampled.

MP	MC	FE	Space uti.	# items
✗	✗	✗	7.82%	2.0
✓	✗	✓	27.9%	7.5
✓	✓	✗	63.7%	16.9
✗	✓	✓	63.0%	16.7
✓	✓	✓	**66.9%**	**17.5**

Table 1: This ablation study compares the space utilization and the total number of packed items with different combinations of MP, MC and FE, on the CUT-2 dataset.

Experiments

We implement our framework on a desktop computer (`ubuntu 16.04`), which equips with an `Intel Xeon Gold 5115` CPU @ 2.40 GHz, 64G memory, and a `Nvidia Titan V` GPU with 12G memory. The DRL and all other networks are implemented with `PyTorch` (Paszke et al. 2019). The model training takes about 16 hours on a spatial resolution of 10×10. The test time of BPP-1 model (no lookahead) is less than 10 ms.

Training and Test Set

We set $L = W = H = 10$ in our experiments with 64 pre-defined item dimensions ($|\mathcal{I}| = 64$). We also set $l_i \le L/2$, $w_i \le W/2$ and $h_i \le H/2$ to avoid over-simplified scenarios. The training and test sequence is synthesized by generating items out of $\mathcal{I}$, and the total volume of items should be equal to or bigger than bin's volume. We first create a benchmark called **RS** where the sequences are generated by sampling items out of $\mathcal{I}$ randomly. A disadvantage of the random sampling is that the optimality of a sequence is unknown (unless performing a brute-force search). Without knowing whether the sequence would lead to a successful packing, it is difficult to gauge the packing performance with this benchmark.

Therefore, we also generate training sequences via *cutting stock* (Gilmore and Gomory 1961). Specifically, items in a sequence are created by sequentially "cutting" the bin into items of the pre-defined 64 types so that we understand the sequence may be perfectly packed and restored back to the bin. There are two variations of this strategy. **CUT-1**: After the cutting, we sort resulting items into the sequence based on Z coordinates of their FLBs, from bottom to top. If FLBs of two items have the same Z coordinate, their order in the sequence is randomly determined. **CUT-2**: The cut items are sorted based on their stacking dependency: an item can be added to the sequence only after all of its supporting items are there. A 2D toy example is given in the Figure 3 (right) with FLB of each item highlighted. Under CUT-1, both $\{1, 2, 3, 4\}$ and $\{2, 1, 3, 4\}$ are valid item sequences. If we use CUT-2 on the other hand, $\{1, 3, 2, 4\}$ and $\{2, 4, 1, 3\}$

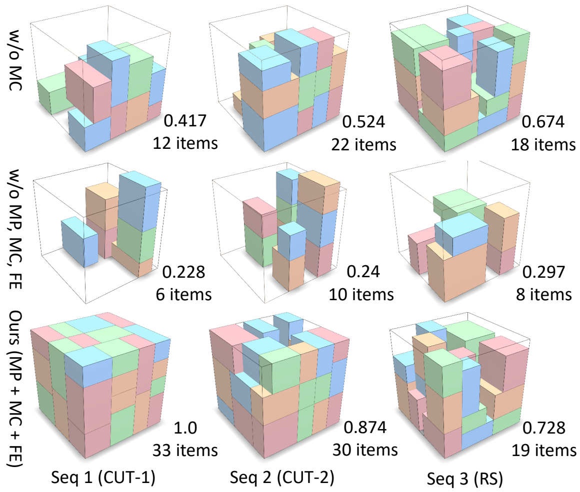

Figure 5: Packing results in the ablation study. The numbers beside each bin are *space uti.* and *# items*.

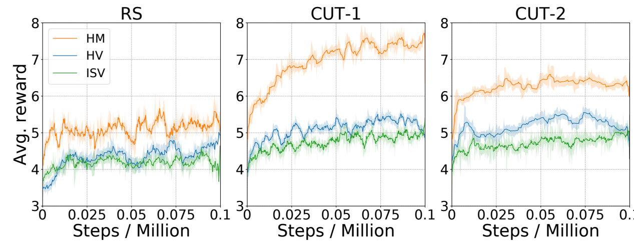

Figure 6: HM shows a clear advantage over vector-based height parameterizations (HV and ISV).

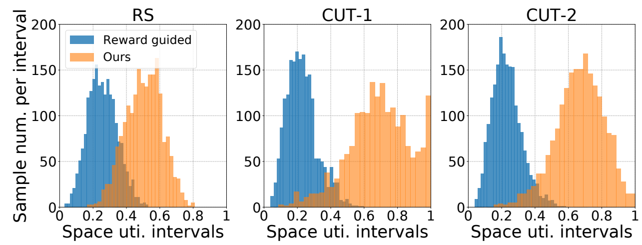

Figure 7: Comparison to DRL with reward tuning. Our method obtains much better space utilization.

would also be valid sequences as the placement of 3 or 4 depends on 1 or 2. For the testing purpose, we generate 2,000 sequences using RS, CUT-1, and CUT-2 respectively. The performance of the packing algorithm is quantitated with space utilization (*space uti.*) and the total number of items packed in the bin (*# items*).

Ablation Study and Evaluation

Table 1 reports an ablation study, we found that the packing performance drops significantly if we do not incorporate the feasibility mask prediction (MP) during the training. The performance is impaired if the mask constraint (MC) is not enforced with our projection scheme. The feasibility-based entropy (FE) is also beneficial for both the training and final performance. Figure 5 demonstrates the packing results visually for different method settings.

Next, we show that the environment parameterization using the proposed 2D height map (HM) (i.e., the $\mathbf{H}$ matrix) is necessary and effective. To this end, we compare our method using HM against that employing two straightforward 1D alternatives. The first competitor is the height vector (HV), which is an $L \cdot W$-dimensional vector stacking columns of $\mathbf{H}$. The second competitor is referred to as the item sequence vector (ISV). The ISV lists all the information of items currently packed in the bin. Each packed item has 6 parameters corresponding to X, Y, and Z coordinates of its FLB as well as the item's dimension. From our test on CUT-1, HM leads to 16.0% and 19.1% higher space utilization and 4.3 and 5.0 more items packed than HV and ISV, respectively. The plots in Figure 6 compare the average reward received using different parameterizations, which shows that 2D height map (HM) is an effective way to describe the state-action space.

In DRL training, one usually discourages low-profile moves by tuning the reward function. We found this strategy is less effective than our constraint-based method. In Figure 7, we compare to an alternative method which uses a negative reward to penalize unsafe placements. Constraint-based DRL seldom predicts invalid moves (predicted placement are 99.5% legit).

Scalability and Versatility

With the capability of lookahead, it is expected that the agent better exploits the remaining space in the bin and delivers a more compact packing. On the other hand, due to the NP-hard nature, big k values increase the environment space exponentially. Therefore, it is important to understand if MCTS is able to effectively navigate us in the space at the scale of $O(k!)$ for a good packing strategy. In Figure 8(a,b), we compare our method with a brute-force permutation search, which traverses all $k!$ permutations of k coming items and chooses the best packing strategy (i.e., the global optimal). We also compare to MCTS-based action search with k lookahead items in which no item per-

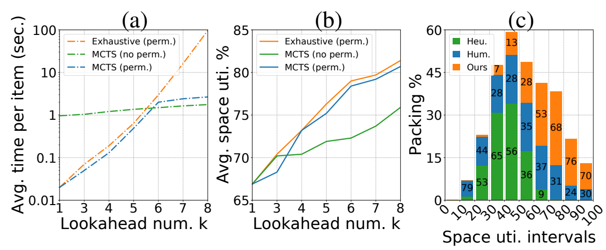

Figure 8: (a): Our permutation based MCTS maintains good time efficiency as the number of lookahead items increases. (b): The performance of our MCTS based BPP-k model achieves similar performance (avg. space utility) as the brute-force search over permutation tree. (c): The distribution of space utilization using boundary rule (Heu.), human intelligence (Hum.), and our BPP-1 method (Ours).

# bins	Space uti.	# items per bin	# total items	Decision time
1	67.4%	17.6	17.6	2.2×10^{-3} s
4	69.4%	18.8	75.2	6.3×10^{-3} s
9	72.1%	19.1	171.9	1.8×10^{-2} s
16	75.3%	19.6	313.6	2.8×10^{-2} s
25	77.8%	20.2	505.0	4.5×10^{-2} s

Table 2: Multi-bin packing tested with the CUT-2 dataset.

mutation is involved. We find that our MCTS-based permutation tree search yields the best results – although having slightly lower space utilization rate ($\sim 3\%$), it is far more efficient. The search time of brute-force permutation quickly surpasses 100s when $k = 8$. Our method takes only 3.6s even for $k = 20$, when permutation needs hours. A larger k makes the brute-force search computationally intractable.

Our method is versatile and can be easily generalized to handle different 3D-BPP variants such as admitting *multiple bins* or allowing *item re-orientation*. To realize multi-bin 3D-BPP, we initialize multiple BPP-1 instances matching the total bin number. When an item arrives, we pack it into the bin in which the item introduces the least drop of the critic value given by the corresponding BPP-1 network. *More details can be found in the supplemental material.* Table 2 shows our results for varying number of bins. More bins provide more place options which leads to better performance. Both time (decision time per item) and space complexities grow linearly with the number of bins.

We consider only horizontal, axis-align orientations of an item, which means that each item has two possible orientations. We therefore create two feasibility masks for each item, one for each orientation. The action space is also doubled. The network is then trained to output actions in the doubled action space. In our test on the RS dataset, we find allowing re-orientation increases the space utilization by 11.6% and the average items packed by 3, showing that our network handles well item re-orientation.

Comparison with Non-Learning Methods and Human Intelligence

Existing works mostly study offline BPP and usually adopt non-learning methods. We compare to two representatives with source code available. The first is a heuristic-based online approach, BPH (Ha et al. 2017) which allows the agent to select the next best item from k lookahead ones (i.e., BPP-k with re-ordering). In Table 3, we compare to its BPP-1 version to be fair. In Figure 9, we compare *online BPH* and our method under the setting of BPP-k. The second method is the *offline LBP* method (Martello, Pisinger, and Vigo 2000) which is again heuristic based. In addition, we also design a heuristic baseline which we call *boundary rule* method. It replicates human's behavior by trying to place a new item side-by-side with the existing packed items and keep the packing volume as regular as possible.

From the comparison in Table 3, our method outperforms all alternative online methods on all three benchmarks and even beats the offline approach on CUT-1 and CUT-2. Through examining the packing results visually, we find that our method automatically learns the above "boundary rule" even without imposing such constraints explicitly. From Figure 9, our method performs better than *online BPH* consistently with varying number of lookahead items even though BPH allows re-ordering of the lookahead items. We also conducted a preliminary comparison on a real robot test of BPP-1. Over 50 random item sequences, our method achieves averagely 66.3% space utilization, much higher than *boundary rule* (39.2%) and *online BPH* (43.2%).

Method	# items / % Space uti. RS	CUT-1	CUT-2
Boundary rule (Online)	8.7 / 34.9%	10.8 / 41.2%	11.1 / 40.8%
BPH (Online)	8.7 / 35.4%	13.5 / 51.9%	13.1 / 49.2%
LBP (Offline)	**12.9 / 54.7%**	14.9 / 59.1%	15.2 / 59.5%
Our BPP-1 (Online)	12.2 / 50.5%	**19.1 / 73.4%**	**17.5 / 66.9%**

Table 3: Comparison with three baselines including both online and offline approaches.

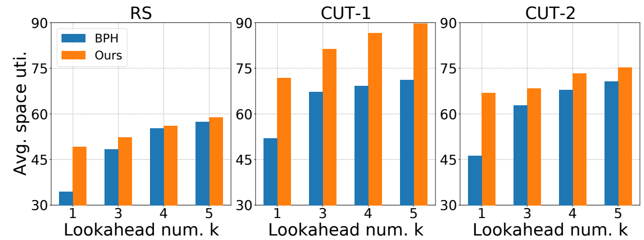

Figure 9: Comparison with the online BPH method (Ha et al. 2017) on BPP-k. Note that BPH allows lookahead item reordering while ours does not.

The strongest competitor to all heuristic algorithms may be human intuition. To this end, we created a simple Sokoban-like app (*see the supplemental material*) and asked 50 human users to pack items manually vs. AI (our method). The winner is the one with a higher space utilization rate. 15 of the users are palletizing workers and the rest are CS-majored undergraduate/graduate students. We do not impose any time limits to the user. The statistics are plotted in Figure 8(c). To our surprise, our method outperforms human players in general ($1,339$ AI wins vs. 406 human wins and 98 evens): it achieves 68.9% average space utilization over $1,851$ games, while human players only have 52.1%.

Conclusion

We have tackled a challenging online 3D-BPP via formulating it as a constrained Markov decision process and solving it with constrained DRL. The constraints include order dependence and physical stability. Within the actor-critic framework, we achieve policy optimization subject to the complicated constraints based on a height-map bin representation and action feasibility prediction. In realizing BPP with multiple lookahead items, we adopt MCTS to search the best action over different permutations of the lookahead items. In the future, we would like to investigate more relaxations of the problem. For example, one could lift the order dependence constraint by adding a buffer zone smaller than $|\mathcal{I}_o|$. Another more challenging relaxation is to learn to pack items with irregular shape.

Acknowledgments

We thank the anonymous PC, AC and extra reviewers for their insightful comments and valuable suggestions. We are also grateful to the colleagues of SpeedBot Robotics for their help on real robot test. Thanks also go to Chi Trung Ha for providing the source code of their work (Ha et al. 2017). This work was supported in part by the National Key Research and Development Program of China (No. 2018AAA0102200), the NSFC (62002376, 62002375, 61532003, 61572507, 61622212) and NUDT Research Grants(No.ZK19-30).

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 22–31. JMLR. org.

Altman, E. 1999. *Constrained Markov decision processes*, volume 7. CRC Press.

Barth-Maron, G.; Hoffman, M. W.; Budden, D.; Dabney, W.; Horgan, D.; Muldal, A.; Heess, N.; and Lillicrap, T. 2018. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617* .

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* .

Chow, Y.; Ghavamzadeh, M.; Janson, L.; and Pavone, M. 2017. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research* 18(1): 6070–6120.

Coffman, E. G.; Garey, M. R.; and Johnson, D. S. 1984. Approximation algorithms for bin-packing—an updated survey. In *Algorithm design for computer system design*, 49–106. Springer.

Crainic, T. G.; Perboli, G.; and Tadei, R. 2008. Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing* 20(3): 368–384.

Crainic, T. G.; Perboli, G.; and Tadei, R. 2009. TS2PACK: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research* 195(3): 744–760.

De Castro Silva, J.; Soma, N.; and Maculan, N. 2003. A greedy search for the three-dimensional bin packing problem: the packing static stability case. *International Transactions in Operational Research* 10(2): 141–153.

De La Vega, W. F.; and Lueker, G. S. 1981. Bin packing can be solved within 1+ ε in linear time. *Combinatorica* 1(4): 349–355.

Delorme, M.; Iori, M.; and Martello, S. 2016. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research* 255(1): 1–20.

Delorme, M.; Iori, M.; and Martello, S. 2018. BPPLIB: a library for bin packing and cutting stock problems. *Optimization Letters* 12(2): 235–250.

Faroe, O.; Pisinger, D.; and Zachariasen, M. 2003. Guided local search for the three-dimensional bin-packing problem. *Informs journal on computing* 15(3): 267–283.

Gambardella, L. M.; and Dorigo, M. 1995. Ant-Q: A reinforcement learning approach to the traveling salesman problem. In *Machine Learning Proceedings 1995*, 252–260. Elsevier.

Gilmore, P. C.; and Gomory, R. E. 1961. A linear programming approach to the cutting-stock problem. *Operations research* 9(6): 849–859.

Goldstein, H.; Poole, C.; and Safko, J. 2002. Classical mechanics.

Ha, C. T.; Nguyen, T. T.; Bui, L. T.; and Wang, R. 2017. An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the Physical Internet. In *European Conference on the Applications of Evolutionary Computation*, 140–155. Springer.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290* .

Hifi, M.; Kacem, I.; Nègre, S.; and Wu, L. 2010. A linear programming approach for the three-dimensional bin-packing problem. *Electronic Notes in Discrete Mathematics* 36: 993–1000.

Hu, H.; Zhang, X.; Yan, X.; Wang, L.; and Xu, Y. 2017. Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930* .

Johnson, D. S.; Demers, A.; Ullman, J. D.; Garey, M. R.; and Graham, R. L. 1974. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on computing* 3(4): 299–325.

Kantorovich, L. V. 1960. Mathematical methods of organizing and planning production. *Management science* 6(4): 366–422.

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

Karmarkar, N.; and Karp, R. M. 1982. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 312–320. IEEE.

Korte, B.; and Vygen, J. 2012. Bin-packing. In *Kombinatorische Optimierung*, 499–516. Springer.

Labbé, M.; Laporte, G.; and Martello, S. 1995. An exact algorithm for the dual bin packing problem. *Operations Research Letters* 17(1): 9–18.

Laterre, A.; Fu, Y.; Jabri, M. K.; Cohen, A.-S.; Kas, D.; Hajjar, K.; Dahl, T. S.; Kerkeni, A.; and Beguir, K. 2018. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. *arXiv preprint arXiv:1807.01672* .

Li, X.; Zhao, Z.; and Zhang, K. 2014. A genetic algorithm for the three-dimensional bin packing problem with heterogeneous bins. In *IIE Annual Conference. Proceedings*, 2039. Institute of Industrial and Systems Engineers (IISE).

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* .

Lodi, A.; Martello, S.; and Vigo, D. 1999. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research* 112(1): 158–166.

Man Jr, E. C.; Garey, M.; and Johnson, D. 1996. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems* 46–93.

Martello, S. 1990. Knapsack problems: algorithms and computer implementations. *Wiley-Interscience series in discrete mathematics and optimiza tion* .

Martello, S.; Pisinger, D.; and Vigo, D. 2000. The three-dimensional bin packing problem. *Operations research* 48(2): 256–267.

Martello, S.; and Toth, P. 1990. Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics* 28(1): 59–70.

Martello, S.; and Vigo, D. 1998. Exact solution of the two-dimensional finite bin packing problem. *Management science* 44(3): 388–399.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540): 529.

Nareyek, A. 2003. Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer decision-making*, 523–544. Springer.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 8024–8035.

Scholl, A.; Klein, R.; and Jürgens, C. 1997. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research* 24(7): 627–645.

Schrijver, A. 1998. *Theory of linear and integer programming*. John Wiley & Sons.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* .

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676): 354–359.

Sweeney, P. E.; and Paternoster, E. R. 1992. Cutting and packing problems: a categorized, application-orientated research bibliography. *Journal of the Operational Research Society* 43(7): 691–706.

Takahara, S.; and Miyamoto, S. 2005. An evolutionary approach for the multiple container loading problem. In *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, 6–pp. IEEE.

Uchibe, E.; and Doya, K. 2007. Constrained reinforcement learning from intrinsic and extrinsic rewards. In *2007 IEEE 6th International Conference on Development and Learning*, 163–168. IEEE.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, 2692–2700.

Wang, R.; Nguyen, T. T.; Kavakeb, S.; Yang, Z.; and Li, C. 2016. Benchmarking dynamic three-dimensional bin packing problems using discrete-event simulation. In *European Conference on the Applications of Evolutionary Computation*, 266–279. Springer.

Wang, Z.; Schaul, T.; Hessel, M.; Van Hasselt, H.; Lanctot, M.; and De Freitas, N. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* .

Wu, Y.; Mansimov, E.; Grosse, R. B.; Liao, S.; and Ba, J. 2017. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, 5279–5288.

Zhang, W.; and Dietterich, T. G. 2000. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Journal of Artificial Intelligence Reseach* 1: 1–38.