SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

January 2022, Vol. 65 112105:1–112105:17
https://doi.org/10.1007/s11432-021-3348-6

Learning practically feasible policies for online 3D bin packing

Hang ZHAO[1†], Chenyang ZHU[1†], Xin XU[2], Hui HUANG[3] & Kai XU[1*]

[1]*School of Computer Science, National University of Defense Technology, Changsha* 410073*, China;*
[2]*College of Intelligence Science and Technology, National University of Defense Technology, Changsha* 410073*, China;*
[3]*College of Computer Science and Software Engineering, Shenzhen University, Shenzhen* 518060*, China*

Received 8 July 2021/Revised 20 August 2021/Accepted 22 September 2021/Published online 27 December 2021

Abstract We tackle the online 3D bin packing problem (3D-BPP), a challenging yet practically useful variant of the classical bin packing problem. In this problem, the items are delivered to the agent without informing the full sequence information. The agent must directly pack these items into the target bin stably without changing their arrival order, and no further adjustment is permitted. Online 3D-BPP can be naturally formulated as a Markov decision process (MDP). We adopt deep reinforcement learning, in particular, the on-policy actor-critic framework, to solve this MDP with constrained action space. To learn a practically feasible packing policy, we propose three critical designs. First, we propose an online analysis of packing stability based on a novel stacking tree. It attains a high analysis accuracy while reducing the computational complexity from $O(N^2)$ to $O(N \log N)$, making it especially suited for reinforcement learning training. Second, we propose a decoupled packing policy learning for different dimensions of placement which enables high-resolution spatial discretization and hence high packing precision. Third, we introduce a reward function that dictates the robot to place items in a far-to-near order and therefore simplifies the collision avoidance in movement planning of the robotic arm. Furthermore, we provide a comprehensive discussion on several key implemental issues. The extensive evaluation demonstrates that our learned policy outperforms the state-of-the-art methods significantly and is practically usable for real-world applications.

Keywords bin packing problem, online 3D-BPP, reinforcement learning

Citation Zhao H, Zhu C Y, Xu X, et al. Learning practically feasible policies for online 3D bin packing. Sci China Inf Sci, 2022, 65(1): 112105, https://doi.org/10.1007/s11432-021-3348-6

1 Introduction

The bin packing problem (BPP) is one of the most famous problems in combinatorial optimization. It aims to pack a collection of items with various weights into the minimum number of bins. The total weight of items in each bin is below the bin's capacity c [1]. BPP is a classic NP-hard problem. We are interested in its 3D variant, i.e., 3D-BPP [2] which introduces much more solving complexity. Given item i with 3D "weight" about length l_i, width w_i, and height h_i, 3D-BPP coordinates planning item's assignment in three dimensions simultaneously. Each 3D dimension has its capacity including $L \geqslant l_i$, $W \geqslant w_i$, and $H \geqslant h_i$. It is assumed that $l_i, w_i, h_i, L, W, H \in Z^+$. 3D-BPP also pursues to pack the set of items $\mathcal{I}$ with the fewest bins.

3D-BPP finds widely practical applications in modern packaging, logistics, and manufacturing. It is especially a core technique in developing palletizing robots for intelligent logistics (Figure 1). A palletizing robot is designed to pack boxes into rectangular bins of standard dimension. Maximizing the storage use of bins improves production efficiency like inventorying, wrapping, transportation, and warehousing. Due to its computational complexity, 3D-BPP is relatively less explored than 1D-BPP. Especially when the problem scale increases, the exact algorithms (either using integer linear programming or branch-and-bound) cannot give a solution to the problem within a limited time. Solving the medium-scale 3D-BPP still has to resort to heuristic algorithms [3, 4].

* Corresponding author (email: kevin.kai.xu@gmail.com)
† Zhao H and Zhu C Y have the same contribution to this work.

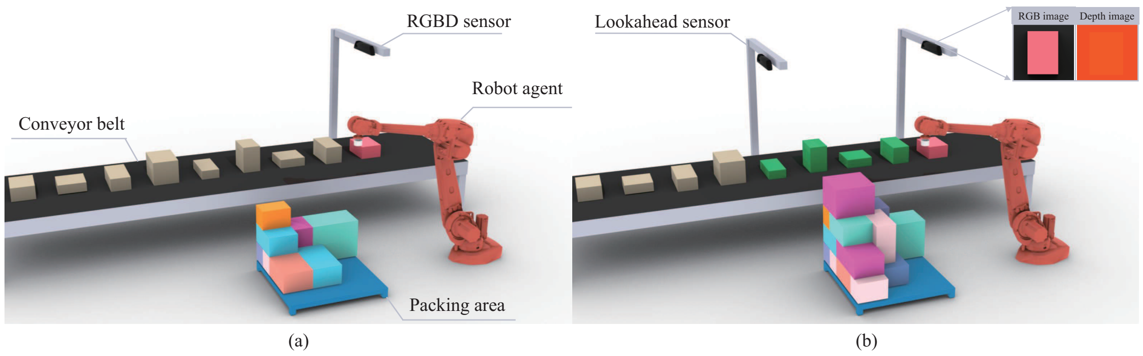

Figure 1 (Color online) Online 3D-BPP has widely practical applications in logistics, manufacture, warehousing, etc. (a) The agent can only observe the next item to be packed (shaded in red), BPP-1; (b) more items (shaded in green) can be observed with additional sensors, BPP-k.

In many application scenarios, an even more difficult problem setting, online 3D-BPP, is highly demanded. In online 3D-BPP, the information of the full item sequence is not provided to the agent/robot (similar to Tetris). As shown in Figure 1, a robot packs continuously coming parcels online. The RGB-D sensors placed around the robot can only provide a partial vision of the item sequence due to the limited camera view field. Since the conveyor is forwarding sequentially during the packing, only a small period can be given for the robot to unload and place the parcel. Such constraints make online 3D-BPP not a pure combinatorial optimization problem since the optimal solution cannot be obtained by brute-force enumeration.

We approach online 3D-BPP by formulating it as a sequential decision making problem and solving it with deep reinforcement learning (DRL), similar to [5]. Albeit being quite effective, several limitations of [5], such as heuristic analysis of packing stability, limited resolution of spatial discretization, and collision agnostic of packing scheme, hinder the practical applicability of learned policy. To this end, we propose the following substantial enhancements to learn practically more feasible policies for online 3D-BPP.

First, we impose a physically plausible yet highly efficient stability analysis to facilitate the learning of stable packing policies. In particular, we propose an online analysis of packing stability with a novel stacking tree. Stacking tree analysis attains a 99.9% accuracy of stability analysis with an $\mathcal{O}(N\log N)$ complexity (N is item count). It results in several magnitudes of acceleration of stability analysis over traditional static equilibrium analysis methods which are $O(N^2)$, making it especially suited for reinforcement learning (RL) training. Meanwhile, the accurate analysis also leads to a significant boost of space utilization (by about 10%) over [5] based on hand-designed, conservative feasibility evaluation.

Second, to deal with large action space caused by high-resolution bin discretization for accurate packing, we propose a novel decoupled packing policy learning scheme. It learns three packing policies for the length and the width dimensions, and the horizontal orientation of the item to be packed, respectively. The three policies are learned with conditionally probabilistic dependency between each other. This enables our method to work with up to 100×100 discretization resolution which is intractable for the method in [5].

Third, to learn a more practically usable policy, we introduce a reward function which encourages the robot to place items in a far-to-near order. This greatly eases the collision avoidance between the robot and the packed items and simplifies the movement planning of the robotic arm.

Our method is formulated as a constrained Markov decision process (CMDP) [6] and adopts the on-policy actor-critic framework [7,8]. Different from [5], we compute the feasibility mask for the placement actions based on the stacking tree analysis. The feasibility mask is then provided to the actor networks and then used to modulate the action probabilities output by the actor. We also discuss several practical issues in the real robot implementation of our algorithm. Finally, we conduct extensive evaluations to validate the efficacy of the new designs.

2 Related work

BPP is a long-term concern in combinatorial optimization, and the earliest literature can be traced back to the 1960s [9]. The most typical bin packing problem is 1D-BPP which seeks for an assignment of a collection of items with various scalar weights to multiple bins and minimizes the used bin number. Knowing to be strongly NP-hard, most existing studies focus on designing good heuristic and approximation algorithms and their worst-case performance analysis [10]. Bin packing problem exists many variants, 2D- and 3D-BPP are natural generalizations of them. For high-dimension BPP, an item has a high-dimension size of the width, height, and/or depth, which differentiates the verification of the packing feasibility. The complexity and the difficulty significantly increase for high-dimension BPP instances. According to the timing statistic reported in [2], exactly solving 3D-BPP of a size matching an actual parcel packing pipeline remains infeasible. There have been several strategies in designing fast approximate algorithms, e.g., guided local search [11], greedy search [12], and tabu search [13, 14]. In contrast, genetic algorithms lead to better solutions as a global, randomized search [15, 16].

Similar strategies have also been adapted to online BPP studies like [5, 17–19]. Different from the offline setting, the size information of coming items is unknown for the agent to optimize the packing and the packing order cannot be adjusted as well. It makes online BPP a much more challenging problem. Some studies [17, 18] have employed the hand-coded heuristics based on the human experience. Meanwhile, studies like [5] have adopted DRL to learn how to pack things effectively through trials and error optimization.

Stability estimation is essential in the bin packing problem. However, most previous studies that focus on the 3D-BPP ignore it in their solution due to the high computational load. Estimating the stability of stack objects for physics engines is widely studied in past decades. Ref. [20] introduced novel complementarity formulation, solver, and error correction algorithms for the collision detection framework which can enable the stability estimation running in real-time for large scale objects. In contrast, Ref. [21] presented a real-time physics engine to improve structured stacking behavior with small-scale objects. Ref. [22] presented a constraint-based method to stabilize a stack of piles. Ref. [23] introduced a hypothesis for physical simulation simplification that freezing transformations of objects in a random pile does not affect the visual plausibility of a simulation. However, none of them can be efficient enough for stability estimation in a DRL training framework.

DRL has demonstrated tremendous success in learning complex behavior skills and solving challenging control tasks with high-dimensional raw sensory state-space [7, 24, 25]. The success can be attributed to the utilization of high-capacity deep neural networks for powerful feature representation learning and function approximation. Since the seminal work of deep Q-network (DQN) [25], a large body of literatures emerged. The existing research is largely divided into two lines: value function learning [25, 26] and policy search [27, 28]. Actor-critic methods, designed to combine the two approaches, have grown in popularity. Asynchronous advantage actor-critic (A3C) [7] is a representative actor-critic method. It combines advantage updates with the actor-critic formulation and adopts asynchronously updated policy and value function networks trained in parallel with multiple agents. In A2C [29], on the other hand, the networks of multiple agents are updated synchronously.

We base our actor-critic architecture on ACKTR [8] which applies trust-region optimization to both the actor and the critic. We propose a series of adaptions for solving online 3D-BPP. To realize constrained reinforcement learning, we propose a simple approach by projecting the trajectories sampled from the actor to the constrained state-action space. To enable our agent to fit the high-resolution demand for real packing, we decompose the actor into three actor-heads to predict the corresponding actions in sequence. We also feed the last predicted action to the next actor-head as a conditional probabilistic dependency for a more stable training process.

Reinforcement learning (RL) for combinatorial optimization has booming development in recent years. Bello et al. [30] combined RL pretraining and active search and demonstrated that RL-based optimization outperforms the supervised learning framework when tackling NP-hard combinatorial problems. Kool et al. [31] encoded TSP graphs with Transformer [32] and trained this model with RL method. Zhang et al. [33] enabled an end-to-end RL agent to master priority dispatching rules for solving job-shop scheduling problems. Wang et al. [34] applied RL to automatically generating a move plan for indoor scene arrangement. Refs. [35] and [36] are both devoted to solving offline 3D-BPP where the main goal is to find an optimal sequence of items.

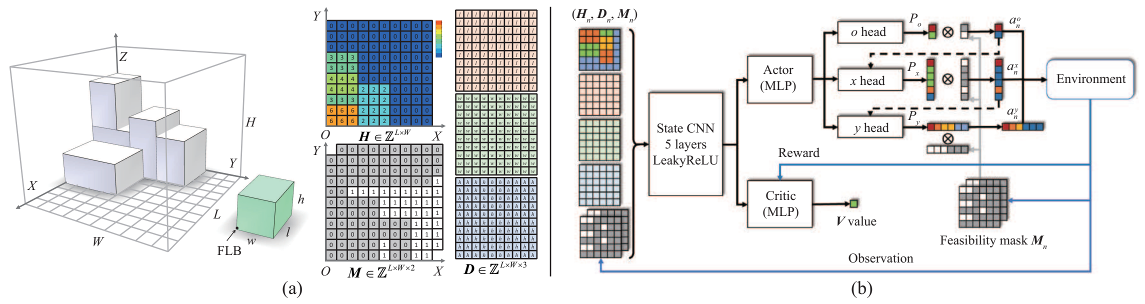

Figure 2 (Color online) The environment state of the agent. (a) The grey boxes indicate the items already packed, which also represents the bin configuration. The green box is the next item to be packed and it can only be placed at the grid cell where feasibility mask $\boldsymbol{M}$ is 1. (b) The network architecture with decomposed actor-heads. Note that the training of three actor-heads is coupled with conditional probabilistic dependencies. Actor-heads perform their prediction tasks in sequence.

3 Method

We adopt a similar problem configuration for online 3D-BPP as [5]. Each item $n \in \mathcal{I}$ is a cube whose size is $[l_i, w_i, h_i]$. As soon as an item arrives the packing area, the agent needs to place it in the bin immediately while only being aware of the information of the next few coming items $\mathcal{I}_o \subset \mathcal{I}$. Our method is implemented based on the architecture of constrained DRL and we will start our problem statement and formulation under the context of DRL.

3.1 Problem statement and formulation

The DRL formulation of our method can be expressed as $(\mathcal{S}, \mathcal{A}, P, R)$, which is an MDP constructed with a set of environment states $\mathcal{S}$, the action set $\mathcal{A}$, reward function $R: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and transition probability function $P: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$. $P(s'|s,a)$ gives the probability of transiting from s to s' for given action a. Our method is model-free since we do not learn $P(s'|s,a)$ explicitly. The policy $\pi: \mathcal{S} \to \mathcal{A}$ 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.

Environment state. A complete 3D-BPP state representation should include the following three parts: the current configuration of the bin, the coming items to be placed, and the feasibility mask. To parameterize the bin configuration, we discretize its bottom area as an $L \times W$ regular grid along the length (X) and the width (Y) directions, respectively. We record at each grid cell the current height of stacked items, leading to a 2D integer height map $\boldsymbol{H}_n \in \mathbb{Z}^{L\times W}$ (see Figure 2). The dimensionality of item n is given as $\boldsymbol{d}_n = [l_n, w_n, h_n]^{\mathrm{T}} \in \mathbb{Z}^3$. The feasibility mask $\boldsymbol{M}_{n,o}$ is a binary matrix of size $L \times W$ indicating the placement feasibility of n with orientation o at each grid cell. Putting together, the current environment state can be written as $s_n = \{\boldsymbol{H}_n, \boldsymbol{d}_n, \boldsymbol{d}_{n+1}, \ldots, \boldsymbol{d}_{n+k-1}, \boldsymbol{M}_{n,o}\}$. We first consider the case where $k = |\mathcal{I}_o| = 1$ (Figure 1(a)), 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 $k > 1$ (Figure 1(b)) afterwards.

Action and state update. We consider only horizontal, axis-align orientations of an item, which means that each item n has two possible orientations $o_n(d_n) = \{[l_n, w_n, h_n]^{\mathrm{T}}, [w_n, l_n, h_n]^{\mathrm{T}}\}$. During the packing, the agent places orientated n's front-left-bottom (FLB) corner (Figure 2(a)) at a certain grid cell 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) with the orientation adjustment o_n, this action is represented as $a_n = (x_n, y_n, o_n)$. The range of $\mathcal{A}$ would increase dramatically with a large resolution of x_n, y_n. In other words, it would be difficult to optimize action while high place accuracy is needed. To solve this problem, we propose an action space decomposition method in Subsection 3.3. After a_n is executed, $\boldsymbol{H}_n$ is updated by adding h_n to the maximum height over all the cells covered by n: $\boldsymbol{H}'_n(x,y,o) = h_{\max}(x,y,o) + h_n$ for $x \in [x_n, x_n + (1-o_n)l_n + o_n w_n], y \in [y_n, y_n + (1-o_n)w_n + o_n l_n]$, with $h_{\max}(x,y)$ being the maximum height among those cells.

Feasibility constraint. A practical online BPP solution should also premeditate the stability of a placement besides securing enough valid space for future items. Placing an item in a risky LP will

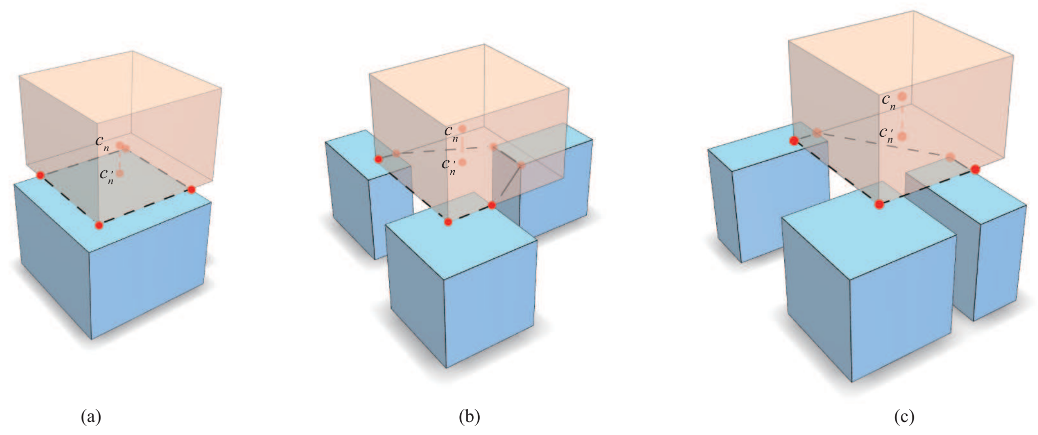

Figure 3 (Color online) (a) The centroid of n is supported by a packed item directly and therefore n is stable. (b) The vertical projection c_n' of centroid c_n is inside the convex hull which is constructed by contact points of $\mathcal{I}_{\text{support}}$. (c) c_n' falls outside of the supported convex hull and n is unstable in this situation.

end the packing episode early and even cause economic damage in practice. We propose a stacking tree based stability estimation method in Subsection 3.2 to form feasibility masks (see Figure 2) as optimization constraints to avoid insecure behaviors. Our problem becomes a CMDP [6] in this scenario. Typically, one augments the MDP with an auxiliary cost function $C : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ mapping state-action tuples to costs, and requires that the expectation of the accumulated cost should be bounded by c_m: $J_C(\pi) = E_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma_C^t C(s_t, a_t)] \leqslant c_m$. Several methods have been proposed to solve CMDP based on e.g., algorithmic heuristics [37], primal-dual methods [38], or constrained policy optimization [39]. 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 article, we propose to exploit the mask $\boldsymbol{M}$ to guide the DRL training to enforce the feasibility constraint without introducing excessive training complexity.

3.2 Stability estimation

Stack stability estimation is critical in the bin packing problem, especially in 3D. A good stability estimation for LPs of n would not only secure the safety of the placement but also decrease the searching range of the action space. To calculate $\boldsymbol{M}_{n,o}$, the most straightforward solution is to simulate the force analysis among packed items $\mathcal{I}_{\text{packed}} \subset \mathcal{I}$ while n is placed with orientation o. However, this simulation would become extremely complicated while the $|\mathcal{I}_{\text{packed}}|$ increasing since force analysis with densely structured stacking is an NP-hard problem [20]. To ensure the stack stability estimation for LPs is real-time, we propose a centroid-based approach which improves the efficiency by more than 100 times.

Supported centroid. In industrial bin packing applications, item n to be packed usually has uniform mass distribution. An item n is stable if its centroid is supported in this scenario. Specifically, an LP of n is considered stable if it satisfies any of the following conditions: (1) The centroid c_n of n is directly supported by a packed item with this LP; (2) n is supported by a group of items $\mathcal{I}_{\text{support}} \subset \mathcal{I}_{\text{packed}}$ and c_n is inside the convex hull which is constructed by contact points of $\mathcal{I}_{\text{support}}$, as illustrated in Figure 3. The centroid-based discrimination is easy to implement with the Boolean operation and can be highly parallelized.

Adaptive stacking tree. However, we not only need to estimate the stability of the current LP but also calculate the stability changes of $\mathcal{I}_{\text{packed}}$ which is introduced by the new contact. The mass of n changes the centroids of items which support it, and these changes may alter their stability. In other words, a top-down traverse of stability update is necessary. However, the amount of work done in this fashion is $\mathcal{O}(N^2)$ since a whole traverse is needed for each placed item. We propose an adaptive stacking tree structure to update the mass distribution flow of $\mathcal{I}_{\text{packed}}$ efficiently in $\mathcal{O}(N \log N)$. The key idea here is that the mass of n at LP would only distribute to the items $\mathcal{I}_{\text{active}} \subset \mathcal{I}_{\text{packed}}$ which support it, directly or indirectly. The stability of $\mathcal{I}_{\text{packed}} - \mathcal{I}_{\text{active}}$ would not change, and the stability update should only be done with $\mathcal{I}_{\text{active}}$ but not the whole packed bin.

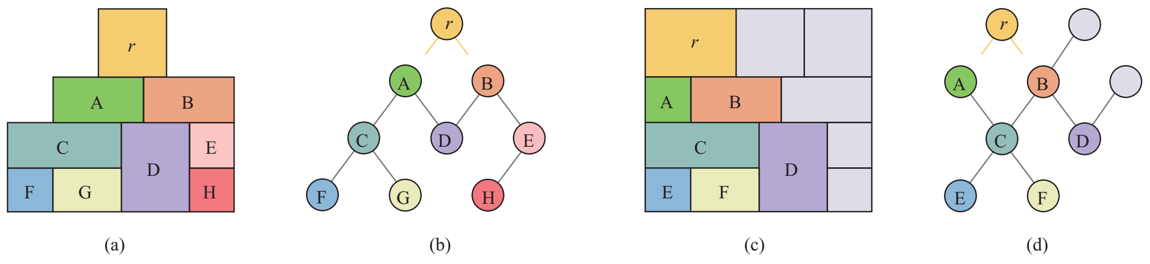

Figure 4 (Color online) (a) r is the current item to be packed which will trigger an adaptive stacking tree to be updated. (b) The adaptive stacking tree of (a). The mass of r flows along each graph edge. (c) Placement of r does not change the mass flow over items with gray color. (d) The adaptive stacking tree of (c). The mass distribution of colored nodes needs to be updated. Note that the mass of inactive items (gray ones) will be distributed to their adjacent active items during the update.

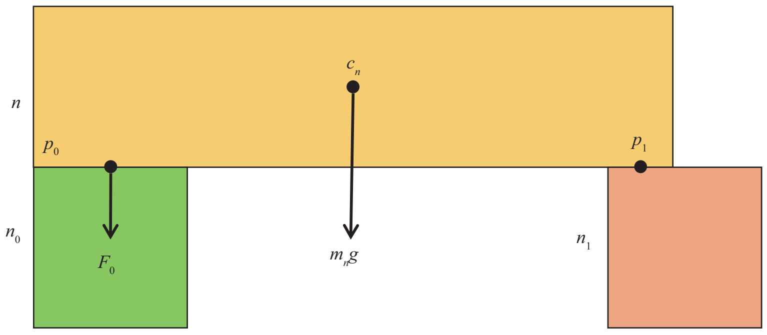

Figure 5 (Color online) The gravity of the item n is distributed to the supporting objects according to the principle of leverage.

In Figure 4, the mass distribution flow of $\mathcal{I}_{\text{packed}}$ can be formed as a graph $\mathcal{G}$, the nodes represent the items in the bin and the edges indicate the amount of mass distribution between two adjacent items. As we discussed above, only a subgraph $\mathcal{G}_n$ of $\mathcal{G}$ needs to be updated while n is placed at LP, where $\mathcal{G}_n$ only contains items in $\mathcal{I}_{\text{active}}$. Obviously, $\mathcal{G}_n$ is a tree which we define as the adaptive stacking tree. $\mathcal{G}_n$ can be constructed from n with a top-down fashion, and only edges of $\mathcal{G}_n$ need to be updated while n is placed. The mass distribution update is processed from the root n to the bottom in $\mathcal{G}_n$ based on the calculated mass distribution, while the edges in $\mathcal{G} - \mathcal{G}_n$ should stay unchanging. This process can be easily implemented during packing in an incremental fashion based on a linked list data structure, which is both memory and time efficient.

Stability update. Updating mass distribution in $\mathcal{G}_n$ is critical for stability estimation. However, it would be highly time-consuming if we operate a force simulation here. We formulate a simple but effective approach based on the principle of leverage, which can achieve 99.9% accuracy with 100 times efficiency. While n is placed at LP, three types of mass distribution would be discussed next. If n is only supported by a single item n_0, the whole mass m_n of n and items above it would transport to n_0. The new centroid of the group of $\{n, n_0\}$ would change to

$$c_{\{n,n_0\}} = \frac{c_n \cdot m_n + c_{n_0} \cdot m_{n_0}}{m_n + m_{n_0}}. \tag{1}$$

Note that we only calculate centroid in the XY plane since the value along the Z axis would not influence the stability. If n is supported by two items n_0 and n_1 safely, we assume the mass distribution of m_n would obey the principle of leverage. As shown in Figure 5, n_0 needs to undertake F_0 to make the whole system stable.

$$F_0 = g \cdot \|(c_n - p_1) \times m_n\| / \|p_0 - p_1\|, \tag{2}$$

$$M\{n, n_0\} = F_0/g + m_{n_0}, \tag{3}$$

where p_0 and p_1 are two contact points of n_0 and n_1. We can also calculate F_1 similarly. Then the centroid of group $\{n, n_0\}$ and $\{n, n_1\}$ can be updated with (1) as well as the mass flows $M\{n, n_0\}$ and $M\{n, n_1\}$ which are distributed to n_0 and n_1. If n is safely supported by more than two items $\{n_k\}$, the mass distribution of m_n would be calculated with (2) for each two items in $\{n_k\}$. Then we will have $C_k^2 + 1$ constraints for mass distribution, and a least-squares optimization would be adopted to find the

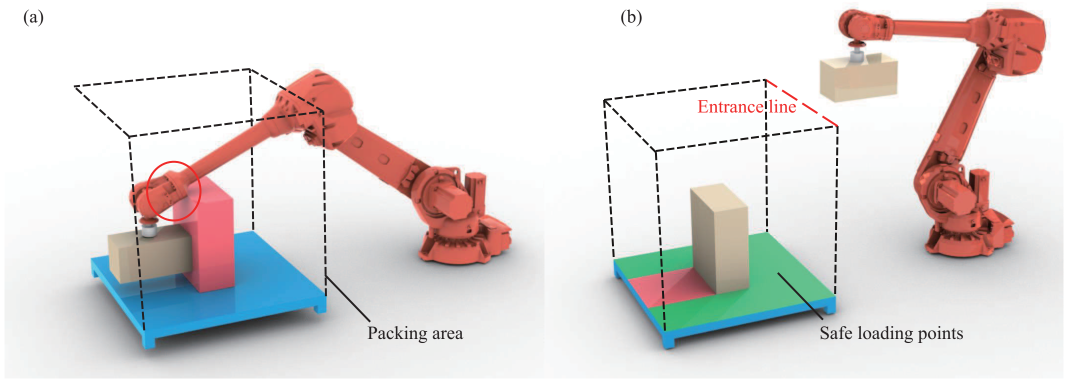

Figure 6 (Color online) (a) The red box packed in the middle of the bin first introduces potential collisions for the following packing. (b) If the robot always enters the packing area at the same entrance line, packing on the green area may introduce fewer potential collisions.

optimal F_i for each item in $\{n_k\}$. The centroid of group $\{n, n_i\}$ can be updated as well. Then each group $\{n, n_i\}$ would trigger the next iteration of update as well as n, and this process would stop if the update reaches the bin bottom or instability is detected.

3.3 Network architecture and training method

We adopt ACKTR [8], which is a state-of-the-art on-policy framework. It iteratively updates an actor and a critic using Kronecker-factored approximate curvature (K-FAC) [40] with trust region. The actor is trained to learn a policy network that outputs the probability of choosing each action (i.e., placing n at each LP). The critic trains a state-value network predicting the state value $V(s_n)$ to indicate how much reward will be earned from s_n. It is adopted to train our actor network. Ref. [5] has demonstrated that ACKTR has a surprising superiority over other DRL algorithms like SAC [41].

The BPP state consists of three components: the height map $\boldsymbol{H}_n$, the coming items to be placed $\boldsymbol{D}_n$, and the feasibility mask $\boldsymbol{M}_{n,o}$. We use a convolutional neural network (CNN) to encode the raw BPP state. For calculation convenience, we "stretch" $\boldsymbol{d}_n$ into a three-channel tensor $\boldsymbol{D}_n \in \mathbb{Z}^{L\times W\times 3}$ so that each channel of $\boldsymbol{d}_n$ spans an $L \times W$ matrix with all of its elements being l_n, w_n, or h_n, respectively (also see Figure 2(a)). Consequently, state $s_n = (\boldsymbol{H}_n, \boldsymbol{D}_n, \boldsymbol{M}_{n,o})$ becomes an $L \times W \times 6$ array (Figure 2(b)).

Reward shaping. We want the agent to learn practical skills which meet the demands of both safety and efficiency without being influenced by various robot types. The efficiency need is satisfied by the reward term which is about the volumetric occupancy introduced by the current item: $10 \times l_n \cdot w_n \cdot h_n/(L \cdot W \cdot H)$ for item n. Ref. [5] reported that the step-wise reward is superior to a termination one (e.g., the final space utilization).

We found that if the agent is only trained with r_n, the actor network would intend to place items uniformly on the XY plane of the bin. However, the robot arm is usually equipped aside rather than above the bin in practice, packing items near the robot arm first would potentially introduce collisions as shown in Figure 6. To avoid it, Ref. [42] proposed that the packing should start from the farther corner as possible. Nonetheless, the performance would drop significantly if we force the actor to follow this principle strictly. To balance the two factors, we introduce a side reward to give our agent a soft constraint.

We assume that the robot arm would always enter the packing area at the same entrance line (EL) (Figure 6). If there is no obstacle on the straight line from EL to a feasible LP, the packing item at this LP can hardly introduce collisions and this type of LP is a safe LP. In other words, the more LPs satisfying this principle there are, the more likely collision-free our next packing is. Let V_{safe} (Figure 6) denote the sum of upon volume of all safe LPs. Our side reward is formulated as $r'_n = V_{\text{safe}}/(L \cdot W \cdot H)$ to ensure the maximum amount of safe LPs. The final reward of the system is then revised as $r_n = \alpha \times l_n \cdot w_n \cdot h_n/(L \cdot W \cdot H) + \beta V_{\text{safe}}/(L \cdot W \cdot H)$, where $\alpha = 10$ and $\beta = 0.1$ since the packing performance is still the primary. The packing episode ends when the current item is not placeable and the r_n is zero.

Action space decomposition. The resolution choice of $\boldsymbol{D}_n$ would significantly influence the range of action space $\mathcal{A}$. However, we need a high resolution of $\boldsymbol{D}_n$ to ensure the packing accuracy in practical applications which would dramatically enlarge $\mathcal{A}$. To solve this problem, we propose a multi-task training method. The action prediction task is decomposed into three subtasks, predicting x_n, y_n and o_n, respectively, (Figure 2). The decomposition would reduce the action space from $O(n^3)$ to $O(3n)$. Different from previous work [43] which learns state-action value functions separately for each task, we train different actors for each task and use unified critic function $V(s_n)$ for the three individual actors to predict P_x, P_y, P_o. Noting that the training of three actors is coupled as well, the advantage is that x_n, y_n, and o_n can be predicted in a correlated fashion which is reasonable in our problem configuration. The policy gradient for actor network parameters training to predict P_x, P_y, P_o is formulated as

$$\nabla\theta_{\text{actor}} = (r_n + \gamma V(s_{n+1}) - V(s_n))\nabla \log P_{\text{actor}}(a_n|s_n), \tag{4}$$

where $\theta_{\text{actor}}, \text{actor} \in \{x, y, o\}$ is the tunable parameters for different actors, $\gamma \in [0, 1]$ is the discount factor, and we set γ as 1 so that $V(s_n)$ can directly present how much reward the agent can obtain from s_n.

Loss function. The whole network is trained via a composite loss. The actor loss L_{actor} is inferred from (4) and the critic loss is constructed based on our reward function r_n, which are the loss functions used for training the actor and the critic, respectively. Next, we use the feasibility mask $\boldsymbol{M}_n$ given by stability estimation to modulate outputs of three actors in order, i.e., the probability distribution of the actions. In theory, if the LP at (x, y) is infeasible for n with orientation adjustment o, the corresponding probability $P(a_n^o = o|s_n), P(a_n^x = x|s_n), P(a_n^y = y|s_n)$ should be set to 0, respectively. However, we find that setting P to a small positive quantity like $\epsilon = 10^{-20}$ works better in practice — it provides a strong penalty to an invalid action but a smoother transformation benefiting to the network training. Our loss function is defined as

$$L = \alpha \cdot L_{\text{actor}} + \beta \cdot L_{\text{critic}} + \omega \cdot E_{\text{inf}} - \psi \cdot E_{\text{entropy}}, \tag{5}$$

$$\begin{cases} L_{\text{actor}} = (r_n + \gamma V(s_{n+1}) - V(s_n)) \log P_{\text{actor}}(a_n|s_n), \\ E_{\text{inf}} = \displaystyle\sum_{\boldsymbol{M}_{n,o}(x,y)=0} P_{\text{actor}}(a_n|s_n), \\ E_{\text{entropy}} = \displaystyle\sum_{\boldsymbol{M}_{n,o}(x,y)=1} -P_{\text{actor}}(a_n|s_n) \cdot \log\left(P_{\text{actor}}(a_n|s_n)\right), \\ L_{\text{critic}} = (r_n + \gamma V(s_{n+1}) - V(s_n))^2, \end{cases} \tag{6}$$

where actor $\in \{x, y, o\}$. To further discourage infeasible actions, we explicitly minimize the summed probability at all infeasible LPs with E_{inf}. Meanwhile, the action entropy loss E_{entropy} is adopted to push the agent to explore more LPs. In this way, we stipulate the agent to explore only feasible actions. We recommend the following parameters which lead to consistently good performance throughout our tests: $\alpha = 1$, $\beta = 0.5$, and $\omega = \psi = 0.01$.

3.4 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 $n + k - 1$). With additional information embedded into the environment state, the agent is expected to learn the policy $\pi(a_n|\boldsymbol{H}_n, \boldsymbol{d}_n, \ldots, \boldsymbol{d}_{n+k-1})$ and have better performance. Ref. [5] claimed that simply encoding the lookahead information into the network input cannot help. We adopt the search-based solution to enable the agent to leverage lookahead information explicitly.

Virtual placement order. Ref. [5] claimed that the current item n's placement should be conditioned on the next $k-1$ one. They enumerate different permutations of the sequence $(\boldsymbol{d}_n, \ldots, \boldsymbol{d}_{n+k-1})$ and drive the actor network to give related plans. To evaluate each sequence, they sum up the accumulated reward and the critic value of the end state after the k-th item is placed. The most promising a_n can be found in the sequence with the highest evaluation score. Note that only n's placement is determined in one permutation search and the actual placement of the k items still follows the order of arrival. To make the search scalable when k is large, Ref. [5] adapted the Monte Carlo tree search (MCTS) [44] to this problem and scaled down the computational complexity from $O(k!)$ to $O(km)$ where m is the number of permutations sampled.

Adaptions for MCTS. We have made some adjustments to [5]'s MCTS method so that it can be employed in practical scenarios. Firstly, we multiply each virtual item's mass with an extremely

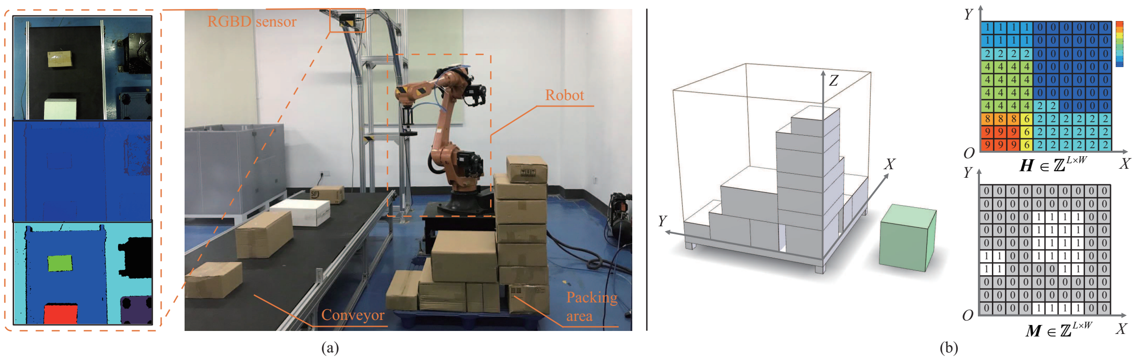
RGBD sensor
Robot
Conveyor
Packing area
(a)
Z
X
Y
Y
O
$H \in \mathbb{Z}^{L\times W}$
X
Y
O
$M \in \mathbb{Z}^{L\times W}$
X
(b)

Figure 7 (Color online) We implement our system in a practical industry environment. (a) The online autonomous bin packing system with an RGB-D sensor and a robot arm; (b) the digital twin which constructed by our method based on the captured data.

small coefficient (10^{-6}) during the search so that the stability estimation for n will not be influenced by nonexistent items. The mass coefficient cannot be zero otherwise the virtual items will lose physical constraint with other virtual ones. We also notice that calculating the feasibility mask based on height map update and stability estimation in MCTS is inevitably time-consuming for real-time since this process would be invoked thousands of times in a whole MCTS. To improve the efficiency, we adopt a root-based parallelization for the MCTS. Different from leaf-based parallelization and tree-based parallelization [45], there is no communication between clones during the parallelization. All the root children of the separate Monte Carlo trees are merged with their corresponding clones after every parallelization iteration. After the search, we choose the action a_n corresponding to the permutation with the highest path value. MCTS allows a scalable lookahead for BPP-k with a complexity of $O(km)$ where m is the number of permutations sampled.

However, parallelizing MCTS will sacrifice the sampling frequency per thread and harm the performance. We modify the MCTS node expansion strategy to ensure that MCTS focuses more on meaningful sequences. The items coming soon in actual order will be more likely to be sampled during the permutation search since a partial sequence given by a virtual order is sometimes meaningless. The sample possibility of item(v_i) is $P_{\text{sample}}(x) \sim \mathcal{N}(0,1)$ in our implementation where x means item(v_i)'s sorted arrival index among unselected items. This attention-based fashion would give MCTS a soft constraint to sample more items with the actual order.

3.5 Implementation

We implement our system in a practical industry environment to validate the actual packing performance. Figure 7 illustrates the online autonomous bin packing system with an RGB-D sensor and a robot arm.

Environment configuration. The model of the RGB-D sensor is Percipio®FM811-GIX-E1 whose depth capture resolution is 1280 × 960. The robot arm is STEP®SR20E which can pick and place box-like items (up to 10 kg) with an air pump driven suction cup. Our packing control system is implemented on a desktop computer (`ubuntu 16.04`), which equips with an `Intel Xeon Gold 5115` CPU @ 2.40 GHz, 64 G memory, and an `Nvidia Titan V` GPU with 12 G memory. The size of box items we adopted in our test meets the transportation standard of S.F.Express® and Taobao®, which ranges from 20 to 50 cm on each dimension. The items would come continuously and randomly with the conveyor belt at a fixed speed and will be packed by the robot arm to the bin placed on the floor which is driven by our system.

Terminals. Our system contains three terminals. The robot terminal drives the robot arm with Codesys, which is a development environment for programming controller applications. The planning terminal is Python-based, which runs the DRL network we proposed to find the optimal packing strategy. The detection terminal is working with the RGB-D sensor, which segments the input depth image and recognizes the size and location of the coming items. These three terminals are communicated through a TCP protocol which centered on the detection terminal.

Executive procedure. Before running the whole system, an alignment between our three terminals is required. A hand-eye calibration [46] is performed between the detection terminal and robot terminal, and thus the coordinate system of the object recognized by the RGB-D sensor can be aligned with the robot coordinate system. Meanwhile, we measure the relative position between the pallet and the robot and align the pallet coordinate system with the robot coordinate system as well.

The detection terminal would capture the depth image of items on the conveyor belt, and the PEAC method [47] is performed to judge whether there is an item in the field and measure its size. If an item is detected, the detection terminal would stop the conveyor belt and send the item size $\boldsymbol{d}_n = [l_n, w_n, h_n]^{\mathrm{T}} \in \mathbb{Z}^3$ to the planning terminal for packing strategy search.

While the planning terminal is calculating the optimal LP for the input item n, the robot terminal would drive the robot arm to pick this item up. It would take few seconds for the robot to complete the picking and the planning terminal can finish the calculating during this period. Once the robot arm leaves the camera's field of view, the detection terminal will restart the conveyor belt and the robot terminal would pack the picked item to the LP given by our planning terminal. At normal speed, it takes 8–9 s for our system to pack a box item from the conveyor belt to the pallet while human palletizers need to spend over 11 s without counting the rest time.

4 Experiments

Our DRL agent is trained in PyTorch [48]. Training on a spatial resolution of 100×100 takes about 12 h and a single decision time is less than 10 ms. We compare our method with some state-of-the-art online bin packing methods to demonstrate the superiority. OnlineBPH [17] and OnlineDRL [5] are two representatives, where OnlineBPH is non-learning based and OnlineDRL learns how to pack from the data. Ref. [5] also proposed a heuristic baseline called 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.

4.1 Training and test set

We set $L = W = H = 100$ in our experiments with 125 pre-defined item dimensions ($|\mathcal{I}| = 125$). The 100×100 resolution is large enough for most of the packing applications in the real world. To avoid over-simplified scenarios, we limit $l_i \leqslant L/2$, $w_i \leqslant W/2$, and $h_i \leqslant H/2$. The training and test sequences are synthesized by generating items out of $\mathcal{I}$, and the total volume of items should be equal to or bigger than the bin's volume. We employ three types of data proposed by [5] to train and evaluate our method. One of them generates the sequences by randomly sampling items (RS) out of $\mathcal{I}$. Since the optimality of an RS sequence is unknown, the other two types of sequences are generated via cutting stock [49]. Specifically, items in a sequence are created by sequentially "cutting" the bin into items of the pre-defined 125 types so that we understand the sequence may be perfectly packed and restored to the bin. CUT-1 sorts the cut items into the sequence based on Z coordinates of their FLBs, from bottom to top. CUT-2 sorts the cut items on their stacking dependency. We generate 2000 sequences on RS, CUT-1, and CUT-2 respectively for testing purposes. The performance of the packing algorithm is quantitated with space utilization (space uti.) and the total number of items packed in the bin (# items).

4.2 Ablation study and evaluation

Table 1 reports an ablation study about different adoptions. The feasibility mask $\boldsymbol{M}_n$ saves the efforts of exploring invalid actions during the training and guarantees the basic performance of the algorithm. The performance is impaired if the infeasibility loss $E_{\inf}$ is not contributed to the final loss since some exploration would be wasted for infeasible actions. E_{entropy} encourages the agent to find better solutions and benefits the final performance. The decision condition input (Figure 2(b)) for each actor-head also facilitates more stable training and better performance. Figure 8 visualizes the effect of different algorithm settings.

4.3 Stacking tree based stability estimation

Next, we demonstrate that the proposed stacking tree based stability estimation is necessary for both efficiency and performance. Table 2 reports a quantitated comparison between the proposed method and four alternatives on the RS benchmark. The simulator reports the stability estimation given by Bullet [50] according to a numerical force analysis. Bullet is able to present precise stability estimation while it is time-consuming. Secondly, Ref. [5] introduced a heuristic-based approach to discriminate stability of n depends on whether it satisfies any of the following conditions: (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

Table 1 Effect of different adoptions on the RS dataset[a)]

$\boldsymbol{M}_n$	E_{inf}	E_{entropy}	Cond.	Space uti. (%)	# items
✗	✗	✗	✗	39.5	15.2
✗	✓	✓	✓	60.0	23.2
✓	✗	✓	✓	67.9	26.1
✓	✓	✗	✓	68.3	26.4
✓	✓	✓	✗	68.4	26.5
✓	✓	✓	✓	**71.3**	**27.6**

a) Cond. means the decision condition input for each actor-head; space uti. means space utilization; # items means the total number of items packed in the bin.

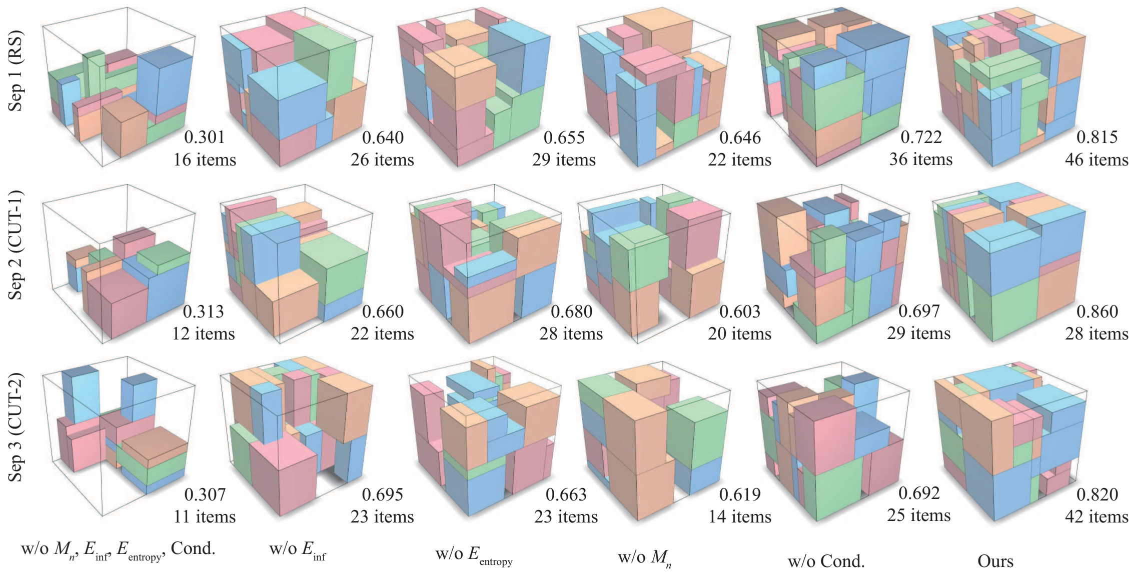

Figure 8 (Color online) Visualization of the ablation study. The numbers beside each bin are space uti. and # items.

Table 2 Quantitated comparison between the proposed method and four alternatives on RS benchmark. Our method can achieve the best packing performance with the high accuracy of stability estimation. Note that the counting stops when unstable placement occurs.

	Space uti. (%)	# items	Stability (%)	Time (s)
Bullet simulator [50]	–	–	–	5.8×10^{-2}
Heuristic rule [5]	58.3	22.8	100.0	1.8×10^{-4}
OnlineBPH [17]	29.1	11.0	11.7	–
Only current packing	66.3	25.68	93.7	3.8×10^{-4}
Ours	**71.3**	**27.6**	99.9	5.0×10^{-4}

of four bottom corners are supported; or (3) over 95% of n's bottom area is supported. This approach can also promise 100% accuracy for stability estimation. However, the heuristic constraints are too strict for our network, the packing performance would be limited in this scenario.

To demonstrate that our stacking tree structure is necessary for stability estimation, we also evaluate the accuracy and performance if only the current packing item n's stability is checked based on our supported centroid approach. It will fail in some cases, which can only achieve 93.7% accuracy. The whole implementation of our stability estimation gives the best packing performance with a 99.9% estimation accuracy while OnlineBPH [17] can only achieve 29.1% space utilization with 11.7% estimation accuracy. Note that our high-speed stability estimation method is for the convenience of DRL training. In practice, we can perform a slower but more accurate stability estimation thread in parallel to avoid the misestimation hazards which account for only 0.1%.

Table 3 Performance comparison with different resolutions between method with decomposed action space (ours) and method with uniform action space [5]. Note that Ref. [5] fails with 100×100 and 200×200 while our method can still maintain a good performance.

		10×10	30×30	50×50	100×100	200×200
Ours	Space uti. (%)	70.1	71.7	72.6	71.3	70.2
	# items	27.1	27.7	28.1	27.6	27.1
[5]	Space uti. (%)	72.3	72.4	51.7	–	–
	# items	27.9	27.9	20.6	–	–

Table 4 Comparison between different methods. The action decomposition method has reached a good level both in performance and training overhead. Unified action space would fail when the resolution increases to 100×100.

		Uniform	Continuous	Ours
10×10	Space uti. (%)	72.3	54.6	70.2
	# items	27.9	21.5	27.1
	GPU memory (MB)	948	942	944
	Network update time (s)	2.0×10^{-2}	1.4×10^{-2}	2.7×10^{-2}
100×100	Space uti. (%)	–	51.2	71.3
	# items	–	20.3	27.6
	GPU memory (MB)	8302	1352	1356
	Network update time (s)	84.7	1.6×10^{-2}	5.1×10^{-2}

4.4 Action space decomposition

Benefit from the design of action space decomposition, our model is able to handle the input states with higher resolution. The dimensions of the pre-defined item set $\mathcal{I}$ are scaled to fit the state representation at different resolutions. For instance, a cube item whose length is 20 cm in all dimensions is expressed as $l_i = w_i = h_i = 20$ at a resolution of 100×100, while $l_i = w_i = h_i = 2$ at a resolution of 10×10. The dimensions of all items are integer multiples of the different resolution unit lengths. Table 3 reports the packing performance change as the resolution grows. Note that the small fluctuations of performance are normal since different sizes of convolution kernels are adopted to encode features with different resolutions. The packing performance of our method remains stable with different resolutions while the performance of [5] drops significantly. The results demonstrate the superiority of the action space decomposition when dealing with high resolution input states.

We compared our design of action space decomposition with other forms of action space: unified action space and continuous action space. Unified action space refers to the uniform encoding of all actions like $a_n = x_n + L \cdot y_n + L \cdot W \cdot o_n$ and only one actor is trained. Another alternative is to map a unified action space to a continuous domain in the range of $[0, 1]$. As the agent predicts in this continuous domain, we project the result back to the discrete domain. We evaluate these three methods with two different resolutions, 10×10 and 100×100. The result is reported in Table 4.

The unified action space can achieve good performance with high computational efficiency when the state resolution is low. However, the performance drops significantly and the computing overhead has increased dramatically when the action space resolution is high. The continuous form of action space is computational friendly, but cannot achieve a good packing performance. Meanwhile, our action space decomposition method can not only maintain similar performance in different spatial dimensions but also maintain a reasonable computational overhead.

High-resolution action space provides more flexibility to practical packing. If an item's dimension is not an integer with the resolution of the action space, an intuitive approach is filling in this non-unit dimension upwards, e.g., from 15 to 20 cm, which also leads to space waste shown in Figure 9(a). If the resolution cell is small, the agent can place items in a more compact way, shown in Figure 9(b). When the item set $\mathcal{I}$ contains non-unit dimension items, our method's performance at 100×100 is 70.9%, while 67.0% at 10×10.

Another problem that may arise in the real packing process is that the drift exists between the excepted position and actual placement, due to robot manipulation error or item misdetection. Here we demonstrate that the high-resolution action space can tolerate item drift more via a toy example in Figures 9(c) and (d). We also provide quantitative experimental results here. Assuming that the size of the pallet is 100 cm $\times$ 100 cm, we will impose a random non-zero drift from -5 to 5 cm to the placed item with

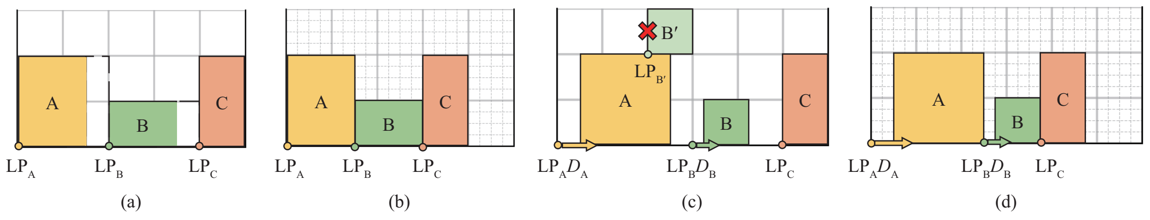

Figure 9 (Color online) A, B, and C arrive in order. The grey grids represent the resolution of the action space, and items can only be placed on the corners of the grids. (a) Filling up the non-unit dimension items A and B in low-resolution action space takes up five units while (b) packing at a high resolution takes less. (c) Packing in low-resolution action space, B cannot be placed at $\mathrm{LP}_{\mathrm{B}'}$ due to drift D_{A}, and placing B at LP_{B} introduces additional space occupancy. The drift D_{B} will affect the placement of C as well. (d) High-resolution packing tolerates item drift and desires less space.

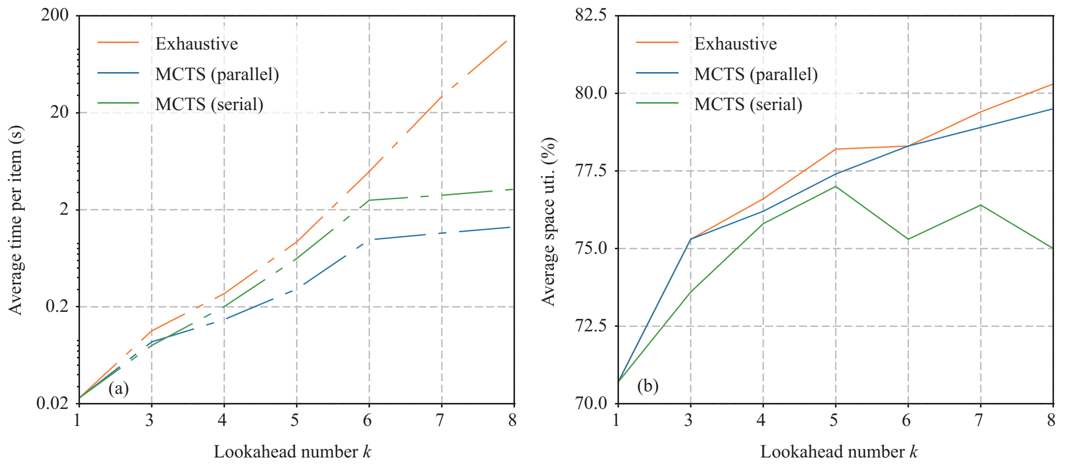

Figure 10 (Color online) (a) Our MCTS implementation avoids the factorial computational cost of exhaustive permutation. The parallelization of MCTS also reduces the execution time by nearly half. (b) Our parallel MCTS achieves similar performance (average space uti.) as the brute-force search over permutation tree. The performance of the serial MCTS drops when the lookahead size is larger than 5 due to the ever-increasing search space.

a probability of 20%. If we pack items at a high resolution of 100 × 100, 1 cm per cell, the packing utilization is 68.2% while the utilization is 71.3% if no drift is imposed. If we execute the same task at a low resolution of 10 × 10, which means 10 cm per cell, the packing utilization drops to 63.1% from 70.1%.

4.5 Scalability of BPP-*k*

Once the agent has mastered the capability of lookahead, it should better exploit the remaining space and deliver a more compact packing. The environment space increases exponentially as k value increases due to the factorial complexity. The agent should make its decision in a reasonable period. In Figure 10(a), we compare the time overhead and algorithm performance of the parallel MCTS with the serial approach. In Figure 10(b), we show that the performance of MCTS improves with the number of lookahead. Note that the root parallelization of MCTS not only improves the efficiency but also enables a larger search space, while the serial MCTS would be easily stuck in a local optimum with a large lookahead size.

In the process of MCTS simulation, only legally sampled sequences would be adopted in the training. Sampling quality would directly impact search efficiency. We evaluate 6 different node sampling strategies with $k = 10$ and the result is reported in Table 5. Here we can see, if we first sample the nodes that actually arrive late with strategy 5^x, the performance drops significantly since too many illegal sampled sequences are generated in this scenario. If we first sample the nodes that actually arrive early, MCTS begins to be able to utilize lookahead information and achieve better performance. $\mathcal{N}(0,1)$ is the best sampling strategy among them. Fixed sampling order performs worse since it cannot explore the search space well.

Table 5 The node sample strategy will influence the performance of MCTS. $\mathcal{N}(0,1)$ is the best sampling strategy among them.

	5^x	Random	$1/x$	$\mathcal{N}(0,1)$	$(1/5)^x$	Fixed sampling order
Space uti. (%)	72.9	79.1	80.6	82.5	81.2	77.9
# items	27.7	31.1	31.6	32.3	31.6	30.0

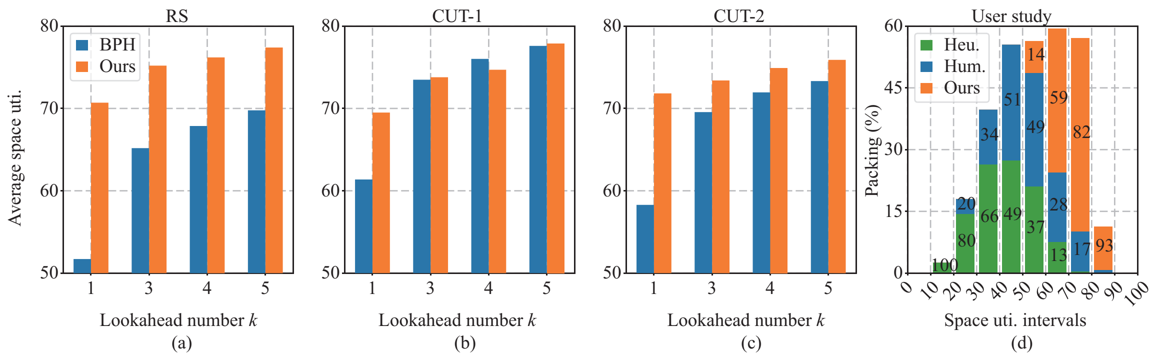

Figure 11 (Color online) (a)–(c) Comparison with the online BPH method [17] on BPP-k with 100×100 state resolution. Note that BPH allows selecting any one of the lookahead k items while ours must place them in the order of arrival. (d) The distribution of space utilization using boundary rule (Heu.), human intelligence (Hum.), and our BPP-1 method (Ours). The more to the right of the distribution, the better the effect of the algorithm.

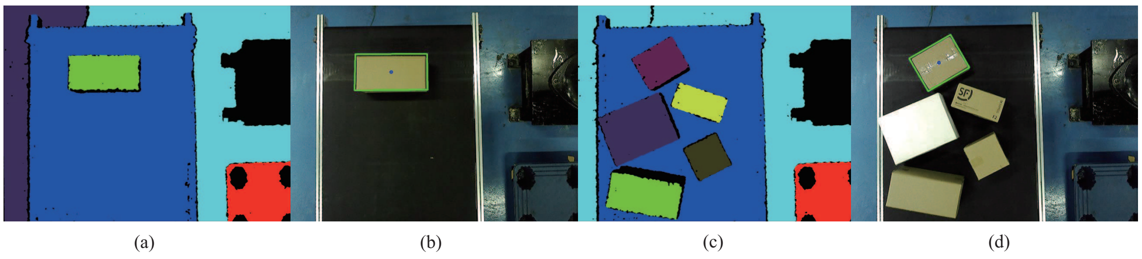

Figure 12 (Color online) (a) Segmentation result under BPP-1 view. (b) The item n is labeled with an anchor, and its dimension and location are recognized. (c) Segmentation result under BPP-k (k is a variable) view. (d) If multiple items are observed by the camera, we sort their positions and select the first one in the queue.

4.6 Comparison with existing methods

We compare to the state-of-the-art non-learning online bin packing method OnlineBPH [17] to demonstrate that our method can achieve competitive performance on this problem. Note that Ref. [17] allows the agent to select arbitrary one from k lookahead items (i.e., BPP-k with re-ordering) and therefore relaxes the order constraints. We also help this method to make a pre-judgment of stability to make it perform better. In Figures 11(a)–(c), we report the comparison under the setting of BPP-k with 100×100 state resolution on the three benchmarks. Our method can surpass OnlineBPH [17] in most cases even re-ordering is allowed in this method while our method does not. Note that OnlineDRL [5] which adopts the unified action space would fail in the training with such a high state resolution.

The most concerning issue should be the comparison between our algorithm and human intuition. To get the conclusion, we asked 50 human users to pack items manually with a Sokoban-like App proposed by [5]. The same sequence is collected and used to test AI (our method). The one with higher space utilization wins. 15 of the users are palletizing workers and the rest are CS-majored undergraduate/graduate students. No time limit is given. We conduct 2104 comparisons and the statistics are plotted in Figure 11(d). Our method outperforms human players in general (1772 AI wins vs. 289 human wins and 43 evens): it achieves 70.4% average space utilization while human players only have 56.3% (palletizing workers achieve 57.6% while CS-majored students achieve 55.2%).

4.7 Robot implementation in industry environment

We test our BPP-1 and BPP-k methods in a practical industry environment. The implementation details can be found in Subsection 3.5. For BPP-1, the RGB-D camera recognizes the coming item by segmenting

Table 6 Performance evaluation of BPP-1 and BPP-k in a practical industry environment. Our method can achieve good performance while keeping the stack stable and avoiding collision.

	Space uti. (%)	#items	Stability (%)	Collision rate (%)	
				Ours	w/o collision-free reward
BPP-1	71.2	49.0	100	2	35
BPP-k	77.9	53.5	100	0	30

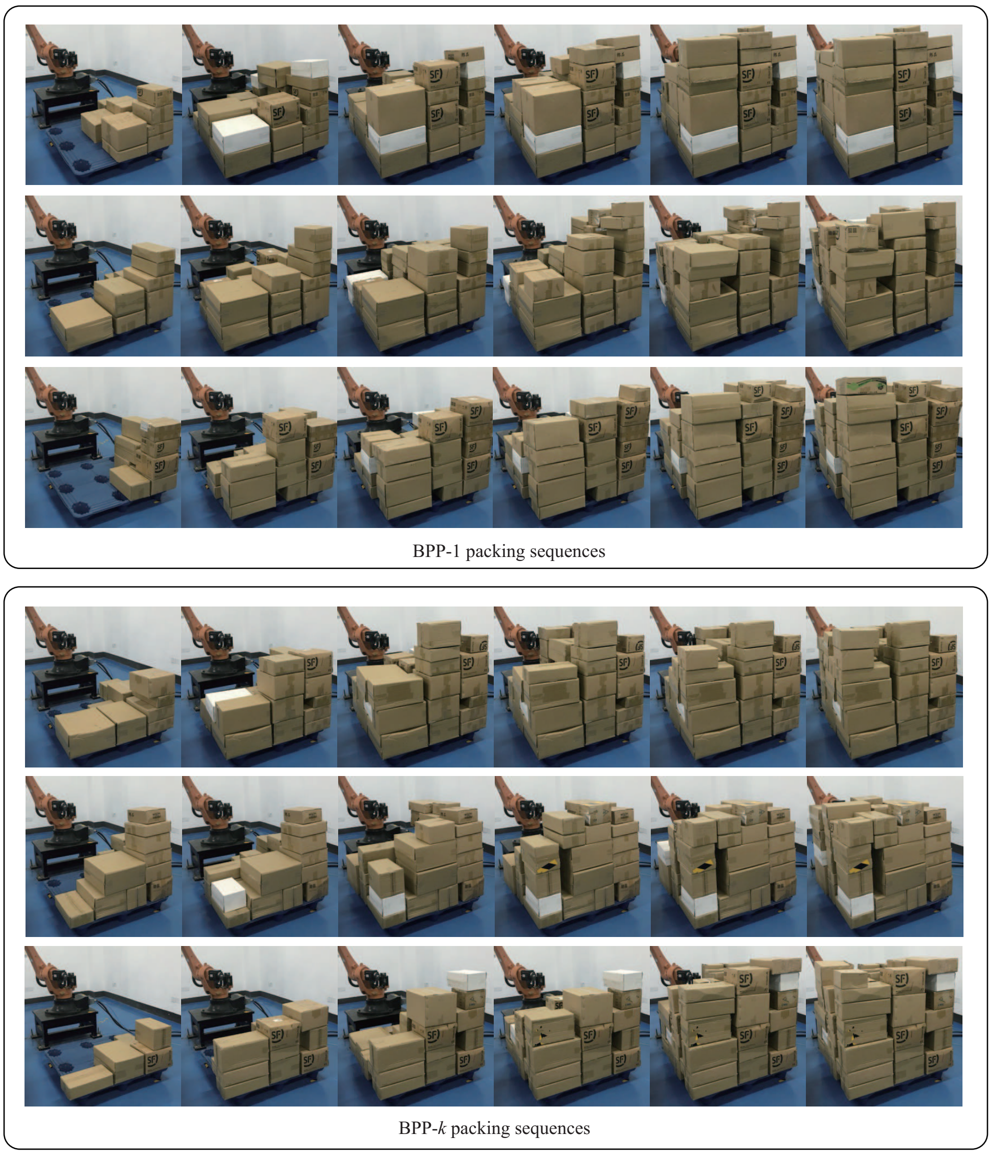

Figure 13 (Color online) Visual examples are given by our robot implementation. The robot places items in a far-to-near order and reduces collisions with packed items.

the captured depth map (Figure 12). If multiple items are observed by the camera at the same time, we sort their positions and select the item at the front of the queue to pack.

Next, we evaluate our BPP-1 and BPP-k in this real demo as well in Table 6. Figure 13 shows some visual captures of the packing process given by our robot implementation. Note that, as shown in Figure 12(d), the number of observed items is uncertain and k is changing while the packing. This is even more challenging when k is fixed. To fully test our BPP-k method, we choose Express®box standard as our test data since its box size is relatively small which ranges from 20 to 50 cm on each dimension. The input would include more numbers of observed items in this configuration. We also test the packing stability and robot arm collision in this experiment. The packing given by our method is 100% stable in all the 50 test sequences with BPP-1 and BPP-k. With the help of our design of collision-free reward, the robot places items in a far-to-near order and only 1 sequence encounters a collision between the robot arm and the packed item with BPP-1. If the collision-free reward is not included, the agent would place items uniformly on the XY plane of the bin and the number of collision sequences would significantly increase.

5 Conclusion

We have provided a practical solution to online 3D-BPP with only partial sequence observation and deployed it on a real robot. To learn a feasible packing policy, we propose three new designs. First, we propose an online analysis of packing stability with a novel stacking tree. Second, we propose a decoupled packing policy learning for different dimensions of placement which enables high-resolution discretization and thus high packing precision. Third, we introduce a reward function that dictates the robot to place items in a far-to-near order and therefore simplifies the collision detection and movement planning of the robotic arm. It is also an interesting problem that investigating the possibility of teaching the agent to learn this far-to-near order without explicitly revising the reward function. Including the robot packing path planning into the RL training may provide a better and more customized strategy to avoid collision. However, the complexity of the optimization space and the efficiency of simulation would make it a challenging problem. As future work, we would like to investigate more on the problem of domain transfer of learned packing policies, e.g., monitoring the possible item drift with high precision. In addition, we would like to investigate more variants of this problem such as stability estimation of items with irregular shapes and study combined solutions of palletizing and depalletizing.

Acknowledgements This work was supported in part by National Key Research and Development Program of China (Grant No. 2018AAA0102200), National Natural Science Foundation of China (Grant Nos. 62132021, 61825305, 62002375, 62002376, 62102435), NUDT Research Grants (Grant No. ZK19-30), DEGP Key Project (Grant No. 2018KZDXM058), GD Science and Technology Program (Grant No. 2020A0505100064), and Shenzhen Science and Technology Program (Grant No. JCYJ20210324120213036). We thank Qijin SHE, Yin YANG, Kun HUANG, Yixing LAN, Kaiwen LI, Junkai REN, and Yao DUAN for active discussion. We also thank Hanchi HUANG for maintaining a good community to communicate reinforcement learning related technologies.

References

1 Korte B, Vygen J. Bin-packing. In: Kombinatorische Optimierung. Berlin: Springer, 2012. 499–516
2 Martello S, Pisinger D, Vigo D. The three-dimensional bin packing problem. Oper Res, 2000, 48: 256–267
3 Crainic T G, Perboli G, Tadei R. Extreme point-based heuristics for three-dimensional bin packing. Informs J Comput, 2008, 20: 368–384
4 Karabulut K, İnceoğlu M M. A hybrid genetic algorithm for packing in 3D with deepest bottom left with fill method. In: Proceedings of International Conference on Advances in Information Systems, 2004. 441–450
5 Zhao H, She Q, Zhu C, et al. Online 3D bin packing with constrained deep reinforcement learning. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence, the 33rd Conference on Innovative Applications of Artificial Intelligence, the 11th Symposium on Educational Advances in Artificial Intelligence, 2021. 741–749
6 Altman E. Constrained Markov Decision Processes. Boca Raton: CRC Press, 1999
7 Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning. In: Proceedings of International Conference on Machine Learning, 2016. 1928–1937
8 Wu Y, Mansimov E, Grosse R B, et al. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In: Proceedings of Advances in Neural Information Processing Systems, 2017. 5279–5288
9 Kantorovich L V. Mathematical methods of organizing and planning production. Manage Sci, 1960, 6: 366–422
10 Coffman E G, Garey M R, Johnson D S. Approximation algorithms for bin-packing—an updated survey. In: Proceedings of Algorithm Design for Computer System Design, 1984. 49–106
11 Faroe O, Pisinger D, Zachariasen M. Guided local search for the three-dimensional bin-packing problem. Informs J Comput, 2003, 15: 267–283
12 de Castro S J L, Soma N Y, Maculan N. A greedy search for the three-dimensional bin packing problem: the packing static stability case. Int Trans Oper Res, 2003, 10: 141–153
13 Lodi A, Martello S, Vigo D. Approximation algorithms for the oriented two-dimensional bin packing problem. Eur J Oper Res, 1999, 112: 158–166
14 Crainic T G, Perboli G, Tadei R. TS2PACK: a two-level tabu search for the three-dimensional bin packing problem. Eur J Oper Res, 2009, 195: 744–760
15 Li X, Zhao Z, Zhang K. A genetic algorithm for the three-dimensional bin packing problem with heterogeneous bins. In: Proceedings of Industrial and Systems Engineering Research Conference, 2014. 2039
16 Takahara S, Miyamoto S. An evolutionary approach for the multiple container loading problem. In: Proceedings of the 5th International Conference on Hybrid Intelligent Systems, 2005. 227–232
17 Ha C T, Nguyen T T, Bui L T, et al. An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the physical internet. In: Proceedings of European Conference on the Applications of Evolutionary Computation, 2017. 140–155
18 Wang R, Nguyen T T, Kavakeb S, et al. Benchmarking dynamic three-dimensional bin packing problems using discrete-event simulation. In: Proceedings of European Conference on the Applications of Evolutionary Computation, 2016. 266–279

19 Hong Y D, Kim Y J, Lee K B. Smart pack: online autonomous object-packing system using RGB-D sensor data. Sensors, 2020, 20: 4448

20 Erleben K. Velocity-based shock propagation for multibody dynamics animation. ACM Trans Graph, 2007, 26: 12

21 Thomsen K K, Kraus M. Simulating small-scale object stacking using stack stability. In: Proceedings of the 23rd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision WSCG 2015. Plzen: Vaclav Skala-UNION Agency, 2015. 5–8

22 Hsu S W, Keyser J. Automated constraint placement to maintain pile shape. ACM Trans Graph, 2012, 31: 1–6

23 Han D, Hsu S W, McNamara A, et al. Believability in simplifications of large scale physically based simulation. In: Proceedings of the ACM Symposium on Applied Perception, 2013. 99–106

24 Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning. 2015. ArXiv:1509.02971

25 Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature, 2015, 518: 529–533

26 Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning. 2015. ArXiv:1511.06581

27 Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on International Conference on Machine Learning, 2014. 387–395

28 Barth-Maron G, Hoffman M W, Budden D, et al. Distributed distributional deterministic policy gradients. 2018. ArXiv:1804.08617

29 Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. 2017. ArXiv:1707.06347

30 Bello I, Pham H, Le Q V, et al. Neural combinatorial optimization with reinforcement learning. 2016. ArXiv:1611.09940

31 Kool W, van Hoof H, Welling M. Attention, learn to solve routing problems! In: Proceedings of the 7th International Conference on Learning Representations, 2019

32 Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Proceedings of Annual Conference on Neural Information Processing Systems, Long Beach, 2017. 5998–6008

33 Zhang C, Song W, Cao Z, et al. Learning to dispatch for job shop scheduling via deep reinforcement learning. In: Proceedings of Annual Conference on Neural Information Processing Systems, 2020

34 Wang H, Liang W, Yu L F. Scene mover: automatic move planning for scene arrangement by deep reinforcement learning. ACM Trans Graph, 2020, 39: 1–15

35 Hu H, Zhang X, Yan X, et al. Solving a new 3D bin packing problem with deep reinforcement learning method. 2017. ArXiv:1708.05930

36 Laterre A, Fu Y, Jabri M K, et al. Ranked reward: enabling self-play reinforcement learning for combinatorial optimization. 2018. ArXiv:1807.01672

37 Uchibe E, Doya K. Constrained reinforcement learning from intrinsic and extrinsic rewards. In: Proceedings of the 6th International Conference on Development and Learning, 2007. 163–168

38 Chow Y, Ghavamzadeh M, Janson L, et al. Risk-constrained reinforcement learning with percentile risk criteria. J Mach Learn Res, 2017, 18: 6070–6120

39 Achiam J, Held D, Tamar A, et al. Constrained policy optimization. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, 2017. 22–31

40 Martens J, Grosse R. Optimizing neural networks with Kronecker-factored approximate curvature. In: Proceedings of International Conference on Machine Learning, 2015. 2408–2417

41 Haarnoja T, Zhou A, Abbeel P, et al. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018. ArXiv:1801.01290

42 Martello S, Pisinger D, Vigo D, et al. Algorithm 864: general and robot-packable variants of the three-dimensional bin packing problem. ACM Trans Math Softw, 2007, 33: 7

43 Tavakoli A, Pardo F, Kormushev P. Action branching architectures for deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2018

44 Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge. Nature, 2017, 550: 354–359

45 Chaslot G M B, Winands M H, van den Herik H J. Parallel Monte-Carlo tree search. In: Proceedings of International Conference on Computers and Games, 2008. 60–71

46 Dekel A, Harenstam-Nielsen L, Caccamo S. Optimal least-squares solution to the hand-eye calibration problem. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020. 13598–13606

47 Feng C, Taguchi Y, Kamat V R. Fast plane extraction in organized point clouds using agglomerative hierarchical clustering. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2014. 6218–6225

48 Paszke A, Gross S, Massa F, et al. PyTorch: an imperative style, high-performance deep learning library. In: Proceedings of Advances in Neural Information Processing Systems, 2019. 8024–8035

49 Gilmore P C, Gomory R E. A linear programming approach to the cutting-stock problem. Oper Res, 1961, 9: 849–859

50 Coumans E. Bullet physics simulation. In: Proceedings of ACM SIGGRAPH 2015 Courses, 2015