Attend2Pack: Bin Packing through Deep Reinforcement Learning with Attention

Jingwei Zhang [*1] **Bin Zi** [*1] **Xiaoyu Ge** [1 2]

Abstract

This paper seeks to tackle the bin packing problem (BPP) through a learning perspective. Building on self-attention-based encoding and deep reinforcement learning algorithms, we propose a new end-to-end learning model for this task of interest. By decomposing the combinatorial action space, as well as utilizing a new training technique denoted as prioritized oversampling, which is a general scheme to speed up on-policy learning, we achieve state-of-the-art performance in a range of experimental settings. Moreover, although the proposed approach attend2pack targets offline-BPP, we strip our method down to the strict online-BPP setting where it is also able to achieve state-of-the-art performance. With a set of ablation studies as well as comparisons against a range of previous works, we hope to offer as a valid baseline approach to this field of study.

1. Introduction

1.1. Background

At loading docks, the critical problem faced by the workers everyday is how to minimize the number of containers needed to pack all cargos in order to reduce transportation expenses; at factories and warehouses, palletizing robots are programmed to stack parcels onto pallets for efficient shipping. The central problem in these real life scenes is the bin packing problem (BPP), which is having an ever-growing impact on logistics, trading and economy under globalization and the boosting of e-commerce.

Given a set of items and a number of bins, BPP seeks for the packing configuration under which the number of bins needed is minimized while utilization (Sec.2) is maximized. There are in general two types of bin packing problems, *online*-BPP and *offline*-BPP, depending on whether the upcoming loading items are known in advance. Specifically, in offline-BPP, the information of all items to be packed are known beforehand, while items arrive one by one in online-BPP and the agent is only aware of the current item at hand. The typical application of online-BPP is palletizing, e.g., instantaneously packing items transported by a conveyor belt onto a pallet with a fixed bottom size (Zhao et al., 2021), where the number of items to be packed onto one pallet is usually in the order of tens; while the practical use cases of offline-BPP include loading cargos into containers so as to be transported by trucks or freighters, where the number of cargos per container could be in the order of hundreds. In this paper, we mainly focus on offline-BPP, especially the single container loading problem (SCLP) (Gehring & Bortfeldt, 2002; Huang & He, 2009; Zhu et al., 2012) in which the goal is to pack all items into a bin in such a way that utilization is maximized, while our proposed method can be ablated to strict online-BPP settings (Sec.3.3).

1.2. Traditional Algorithms

As a family of NP-hard problems (Lewis, 1983), the optimal solution of BPP cannot be found in polynomial time (Sweeney & Paternoster, 1992; Martello et al., 2000; Crainic et al., 2008). Approximate methods and domain-specific heuristics thus have been proposed to find near-optimal solutions. Such methods include: branch and bound (Martello, 1990; Lysgaard et al., 2004; Lewis, 2008); integer linear programming (Dyckhoff, 1981; Cambazard & O'Sullivan, 2010; Salem & Kieffer, 2020); extreme-point-based constructive heuristics (Crainic et al., 2008) where the corner of new items must be placed in contact with one of the corner points inside the container; block-building-based heuristic (Fanslau & Bortfeldt, 2010) that brings substantial performance gains, in which items are first constructed into common building blocks then stacked into containers; as well as meta-heuristic algorithms (Abdel-Basset et al., 2018; Loh et al., 2008). However, the general applicability of these algorithms has been limited either by high computational costs or the requirement for domain experts to adjust the algorithm specifications for each newly encountered scenario.

[*]Equal contribution [1]Dorabot Inc, Shenzhen, Guangdong, China [2]Australian National University, Canberra, Australia. Correspondence to: <jingwei.zhang, bin.zi@dorabot.com>.

Reinforcement Learning for Real Life (RL4RealLife) Workshop in the 38^{th} International Conference on Machine Learning, 2021.

1.3. Learning-based Algorithms

Inspired by the success brought by powerful deep function approximators (Mnih et al., 2015; He et al., 2016a; Vaswani et al., 2017), BPP research has witnessed a surge of algorithms that approach this problem from a learning perspective. As a combinatorial optimization problem, although it requires exponential time to find the optimal solution in the combinatorial solution space, it is relatively cheap to evaluate the quality of a given solution for BPP. This makes the family of policy gradient algorithms (Williams, 1992; Sutton et al., 1999) a good fit to such problems, as given an evaluation signal that is not necessarily differentiable, policy gradient offers a principled way to guide search with gradients, in contrast to gradient-free methods. Another advantage of learning-based approaches is that they are usually able to scale linearly to the input size, and have satisfactory generalization capabilities.

Such attempts of utilizing deep reinforcement learning techniques for combinatorial optimization starts from (Bello et al., 2016) which tackles the traveling salesman problem (TSP) with pointer networks (Vinyals et al., 2015). (Kool et al., 2018) proposes an attention-based (Vaswani et al., 2017) model that brings substantial improvements in the vehicle routing problem (VRP) domain.

As for BPP, (Zhao et al., 2021; Young-Dae et al., 2020; Tanaka et al., 2020) focus on online-BPP and learn how to place each given box, (Hu et al., 2017; Duan et al., 2019) learn to generate sequence order for boxes while placing them using a fixed heuristic; therefore these works do not target the full problem of offline-BPP. Ranked reward (RR) (Laterre et al., 2019) learns on the full problem by conducting self-play under the rewards evaluated by ranking. Although RR outperforms MCTS (Browne et al., 2012) and the GUROBI solver (Gurobi Optimization, 2018) especially with large problem sizes, it directly learns on the full combinatorial action space which could pose major challenges for efficient learning. (Zhao et al., 2021; Jiang et al., 2021) also propose end-to-end approaches for offline-BPP, however, their state representation for encoding might not be optimized for BPP. In this work, we propose a new attention-based model specifically designed for BPP. To facilitate efficient learning, we decompose the combinatorial action space and propose prioritized oversampling to boost on-policy learning. We conduct a set of ablation studies and show that our method achieves state-of-the-art results in a range of comparisons.

2. Methods

We present our approach in the context of 3D bin packing with single bin, from which its application to the 2D situation should be straightforward.

We formulate the bin packing problem (BPP) as a Markov decision process. At the beginning of each episode, the agent is given a set of N boxes each with dimensions (l^n, w^n, h^n), as well as a bin of width W and height H with a length of $L = \sum_{n=1}^{N} \max(l^n, w^n, h^n)$ such is long enough to contain all these N boxes under all variations of packing configurations, as long as each box is in contact with at least one other box. These together make up the input set of a 3D bin packing problem $\mathcal{I} = \{(l^1, w^1, h^1), (l^2, w^2, h^2), \dots, (l^N, w^N, h^N), (L, W, H)\}$. The goal for the agent is to pack all N boxes into the bin in such a way that the utility $r_\mathrm{u} \in [0, 1]$ is maximized, under geometry constraints that no overlap between items is allowed. The utility is calculated as

$$r_\mathrm{u}(\mathcal{C}_\pi) = \frac{\sum_{n=1}^{N} l^n w^n h^n}{L_{\mathcal{C}_\pi} W H}, \tag{1}$$

where $\mathcal{C}_\pi$ is a packing configuration (discussed in more details in Sec.2.1) generated by following packing policy π, and $L_{\mathcal{C}_\pi}$ represents for this packing configuration $\mathcal{C}_\pi$ the length of the minimum bounding box containing all the N packed boxes inside of the bin. The utility is given to the agent as a terminal reward r_u only upon when the last box has been packed into the bin, meaning that the agent would receive no step or intermediate reward during an episode. We choose this terminal reward setup since it is non-trivial to design local step reward functions that would perfectly align with the global final evaluation signal. We also designed and experimented with other terminal reward formulations but found utility to be a simple yet effective solution, as well as the fact that it has a fixed range which makes it easier to search for hyperparameters that can work robustly across various datasets.

When placing a box into the bin, the box can be rotated $90°$ along each of its three axes, leading to a total number of $O = 6$ possible orientations. When it comes to the coordinate for placement inside a bin of $L \times W \times H$, there are also in total a maximum number of $L \times W \times H$ possible locations to place a box (we denote the rear-left-bottom corner of the bin as the origin $(0, 0, 0)$ and define the location of a box n inside the bin also by the coordinate (x^n, y^n, z^n) of its rear-left-bottom corner). This leads to a full action space of size $N \times O \times L \times W \times H$ for the full bin packing problem.

2.1. Decomposing the Action Space

To deal with the potentially very large combinatorial action space, we decompose the bin packing policy π of the agent into two: a sequence policy π^s that selects at each time step t the next box to pack $s_t \in \{1, \dots, N\} \setminus s_{1:t-1}$ (with action space $\mathcal{S}$ of size $|\mathcal{S}| = N$) and a placement policy π^p that picks the orientation and the location for the currently selected box $p_t = (o^{s_t}, x^{s_t}, y^{s_t}, z^{s_t})$ (with action space $\mathcal{P}$ of size $|\mathcal{P}| = O \times L \times W \times H$). We note that once a location

(x^{s_t}, y^{s_t}) is selected for the current box, its coordinate along the height dimension z^{s_t} is also fixed because all items must have support along the height dimension; also since we expect the packing to be compact as possible that each box must be in contact with at least one other box, then once its coordinate y^{s_t} along the width dimension is selected, its coordinate x^{s_t} along the length dimension can be automatically located as the rearest feasible point where the current box would be in contact with at least one other box or the rear wall of the bin, and at the same time does not overlap with any other boxes. Therefore, the actual working action space for the placement policy π^{p} is of size $|\mathcal{P}| = O \times W$, that by selecting a placing coordinate y^{s_t} along the width dimension, the other two coordinates for s_t are located as the rearest-lowest feasible point.

This decomposition can be regarded as casting the full joint combinatorial action space among two agents, formulating BPP as a fully-cooperative multi-agent reinforcement learning task (Panait & Luke, 2005; Busoniu et al., 2008). We note that now the learning system could be more properly modeled as a sequential Markov game (Littman, 1994; Lanctot et al., 2017; Yang & Liu, 2020) with two agents: a sequence agent with action space $\mathcal{S}$ and a placement agent with action space $\mathcal{P}$ sharing the same reward function.

Following this decomposition, the joint policy π given input $\mathcal{I}$ is factorized as

$$\pi\big(s_1, p_1, s_2, p_2, \dots, s_N, p_N \,\big|\, \mathcal{I}\big) = \prod_{t=1}^{N} \pi^{\mathrm{s}}\Big(s_t \Big| f^{\mathrm{s}}_{\mathcal{I}}(s_{1:t-1}, p_{1:t-1})\Big) \pi^{\mathrm{p}}\Big(p_t \Big| f^{\mathrm{p}}_{\mathcal{I}}(s_{1:t}, p_{1:t-1})\Big), \tag{2}$$

in which the sequence policy π^{s} and the placement policy π^{p} together generate the full solution configuration $\mathcal{C}_{\pi^{\mathrm{s}},\pi^{\mathrm{p}}} = \{s_1, p_1, s_2, p_2, \dots, s_N, p_N\}$ in an interleaving manner; $f_{\mathcal{I}}$ is the encoding function that maps (partially completed) configurations into state representations for a particular input set $\mathcal{I}$, which will be discussed below.

We will denote the deep network parameters for encoding as $\boldsymbol{\theta}^{\mathrm{e}}$ (Sec.2.2), the parameters for decoding the sequence policy as $\boldsymbol{\theta}^{\mathrm{s}}$ (Sec.2.3), and those for the placement policy as $\boldsymbol{\theta}^{\mathrm{p}}$ (Sec.2.4); their aggregation is denoted as $\boldsymbol{\theta}$. Dependencies on these parameters are sometimes omitted in notation.

2.2. Encoding

2.2.1. Box Embeddings

At the beginning of each episode, the dimensions of each box in the input are first encoded through a linear layer

$$\bar{\mathbf{b}}^n = \mathrm{Linear}(l^n, w^n, h^n). \tag{3}$$

The set of N such embeddings $\bar{\mathbf{b}}$ ($|\bar{\mathbf{b}}| = d$) are then passed through several multi-head (with M heads) self-attention layers (Vaswani et al., 2017; Radford et al., 2019), with each layer containing the following operations

$$\tilde{\mathbf{b}}^n = \bar{\mathbf{b}}^n + \mathrm{MHA}\big(\mathrm{LN}\big(\bar{\mathbf{b}}^1, \bar{\mathbf{b}}^2, \dots, \bar{\mathbf{b}}^N\big)\big), \tag{4}$$

$$\mathbf{b}^n = \tilde{\mathbf{b}}^n + \mathrm{MLP}\big(\mathrm{LN}\big(\tilde{\mathbf{b}}^n\big)\big). \tag{5}$$

where MHA denotes the multi-head attention layer (Vaswani et al., 2017), LN denotes layer normalization (Ba et al., 2016) and MLP denotes a feed-forward fully-connected network with ReLU activations. We organize the skip connections following (Radford et al., 2019) to allow identity mappings (He et al., 2016b). This produces a set of box embeddings $\mathcal{B} = \{\mathbf{b}^1, \mathbf{b}^2, \cdots, \mathbf{b}^N\}$ (each of dimension $|\mathbf{b}| = d$), which are kept fixed during the entire episode. We note that this is in contrast to several previous works (Li et al., 2020; Jiang et al., 2021) that need to re-encode the box embeddings at every time step during an episode.

2.2.2. Frontier Embedding

Besides the box embeddings $\mathcal{B}$, in order to provide the agent with the description of the situation inside the bin, we additionally feed the agent with a frontier embedding $\mathbf{f}$ (also of size $|\mathbf{f}| = d$) at each time step. More specifically, we define the frontier $\mathbf{F}$ of a bin as the front-view of the loading structure. For a bin of width W and height H, its frontier is of size $W \times H$, with each entry representing for that particular location the maximum x coordinate of the already packed boxes. To compensate for possible occlusions, at each time step t, we stack the last two frontiers $\{\mathbf{F}_{t-1}, \mathbf{F}_t\}$ ($\mathbf{F}_t$ denotes the frontier before the box $(l^{s_t}, w^{s_t}, h^{s_t})$ has been packed into the bin) and pass them through a convolutional network to obtain the frontier embedding $\mathbf{f}_t$.

2.3. Decoding Sequence

We build on the pointer network (Vinyals et al., 2015; Bello et al., 2016; Kool et al., 2018) to model the sequence policy π^{s} in order to allow for variable length inputs.

At time step t, the task for the sequence policy π^{s} is to select the index s_t of the next box to be packed out of all unpacked boxes: $s_t \in \{1, \dots, N\} \setminus s_{1:t-1}$. The embeddings for these "leftover" boxes consistute the set $\mathcal{B} \setminus \mathbf{b}^{s_{1:t-1}}$. Besides the leftover embeddings, those already packed boxes along with the situation inside the bin can be described by the frontier and subsequently the frontier embedding $\mathbf{f}_t$. Therefore we instantiate the encoding function $f^{\mathrm{s}}_{\mathcal{I}}$ (Eq.2) for the sequence policy as

$$\bar{\mathbf{q}}^{\mathrm{s}}_t = f^{\mathrm{s}}_{\mathcal{I}}(s_{1:t-1}, p_{1:t-1}; \boldsymbol{\theta}^{\mathrm{e}}) = \big\langle \langle \mathcal{B} \setminus \mathbf{b}^{s_{1:t-1}} \rangle, \mathbf{f}_t \big\rangle, \tag{6}$$

with $\langle\rangle$ representing the operation that takes in a set of $d-$dimensional vectors and return their mean vector which is also $d-$dimensional. This encoding function generates the query vector $\bar{\mathbf{q}}^{\mathrm{s}}_t$ for sequence decoding.

Following (Bello et al., 2016; Kool et al., 2018), we add a glimpse operation via multi-head (M heads) attention (Vaswani et al., 2017) before calculating the policy probabilities, which has been shown to bring performance gains. At the beginning of each episode, from each of the box embeddings $\mathbf{b}^n$, M sets of fixed context vectors are obtained via learned linear projections. With $h = \frac{d}{M}$, the context vectors $\bar{\mathbf{k}}^{n,m}$ (glimpse key of size h) and $\bar{\mathbf{v}}^{n,m}$ (glimpse value of size h) for the m^{th} head, as well as $\mathbf{k}^n$ (logit key of size d) shared across all M heads are obtained through

$$\bar{\mathbf{k}}^{n,m} = \mathbf{W}^{\bar{\mathbf{k}},m}\mathbf{b}^n, \;\; \bar{\mathbf{v}}^{n,m} = \mathbf{W}^{\bar{\mathbf{v}},m}\mathbf{b}^n, \;\; \mathbf{k}^n = \mathbf{W}^{\mathbf{k}}\mathbf{b}^n, \quad (7)$$

with $\mathbf{W}^{\bar{\mathbf{k}},m}, \mathbf{W}^{\bar{\mathbf{v}},m} \in \mathbb{R}^{h\times d}$ and $\mathbf{W}^{\mathbf{k}} \in \mathbb{R}^{d\times d}$ (in general the key and value vectors are not required to be of the same dimensionality, here $\mathbf{W}^{\bar{\mathbf{k}},m}$ and $\mathbf{W}^{\bar{\mathbf{v}},m}$ are of the same size only for notational convenience). The sequence query vector $\bar{\mathbf{q}}^{\text{s}}_t$ is split into M glimpse query vectors each of dimension $|\bar{\mathbf{q}}^{\text{s},m}_t| = h$. The compatibility vector $\bar{\mathbf{c}}^m_t$ ($|\bar{\mathbf{c}}^m_t| = N$) is then calculated with its entries

$$\bar{\mathbf{c}}^m_{t,n} = \begin{cases} \frac{\bar{\mathbf{q}}^{\text{s},m\top}_t \bar{\mathbf{k}}^{n,m}}{\sqrt{h}} & \text{if } n \notin \{s_{1:t-1}\}, \\ -\infty & \text{otherwise.} \end{cases} \quad (8)$$

Then we can obtain the updated sequence query for the m^{th} head as the weighted sum of the glimpse value vectors

$$\mathbf{q}^{\text{s},m}_t = \sum_{n=1}^{N} \text{softmax}(\bar{\mathbf{c}}^m_t)_n \cdot \bar{\mathbf{v}}^{n,m}. \quad (9)$$

Concatenating the updated sequence query vectors from all M heads we obtain $\mathbf{q}^{\text{s}}_t$ ($|\mathbf{q}^{\text{s}}_t| = d$). Passing it through another linear projection $\mathbf{W}^{\mathbf{q}} \in \mathbb{R}^{d\times d}$, the updated compatibility $\mathbf{c}_t$ ($|\mathbf{c}_t| = N$) is calculated by

$$\mathbf{c}_{t,n} = \begin{cases} C \cdot \tanh\left(\frac{(\mathbf{W}^{\mathbf{q}}\mathbf{q}^{\text{s}}_t)^\top \mathbf{k}^n}{\sqrt{d}}\right) & \text{if } n \notin \{s_{1:t-1}\}, \\ -\infty & \text{otherwise,} \end{cases} \quad (10)$$

where the logits are clamped between $[-C, C]$ following (Bello et al., 2016; Kool et al., 2018) as this could help exploration and bring performance gains (Bello et al., 2016).

Finally the sequence policy can be obtained via

$$\pi^{\text{s}}\big(\cdot \,\big|\, f^{\text{s}}_{\mathcal{I}}(s_{1:t-1}, p_{1:t-1}; \boldsymbol{\theta}^{\text{e}}); \boldsymbol{\theta}^{\text{s}}\big) = \text{softmax}(\mathbf{c}_t). \quad (11)$$

During training, the box index s_t is selected by sampling from the distribution defined by the sequence policy $s_t \sim \pi^{\text{s}}\big(\cdot\big|f^{\text{s}}_{\mathcal{I}}(s_{1:t-1}, p_{1:t-1})\big)$ while during evaluation the greedy action is selected $s_t = \arg\max_a \pi^{\text{s}}\big(a\big|f^{\text{s}}_{\mathcal{I}}(s_{1:t-1}, p_{1:t-1})\big)$.

2.4. Decoding Placement

Given the box index s_t selected by the sequence policy π^{s}, the placement policy π^{p} will select a placement $p_t = (o^{s_t}, y^{s_t})$ that decides for this box its orientation o^{s_t} and its coordinate y^{s_t} along the width dimension within the bin. As discussed in Sec.2.1, once y^{s_t} is selected, the other two coordinates of the box x^{s_t} and z^{s_t} can be directly located.

As for the encoding function $f^{\text{p}}_{\mathcal{I}}$ for the placement policy, it contains the processing of three kinds of embeddings: the current box embedding $\mathbf{b}^{s_t}$, the leftover embeddings $\mathcal{B} \setminus \mathbf{b}^{s_{1:t}}$ and the frontier embedding $\mathbf{f}_t$ (which is the same as for $f^{\text{s}}_{\mathcal{I}}$ since the situation inside the bin has not changed after s_t is selected). Therefore $f^{\text{p}}_{\mathcal{I}}$ takes the form of

$$\mathbf{q}^{\text{p}}_t = f^{\text{p}}_{\mathcal{I}}(s_{1:t}, p_{1:t-1}; \boldsymbol{\theta}^{\text{e}}) = \big\langle \mathbf{b}^{s_t}, \langle \mathcal{B} \setminus \mathbf{b}^{s_{1:t}} \rangle, \mathbf{f}_t \big\rangle. \quad (12)$$

This yields the query vector $\mathbf{q}^{\text{p}}_t$ for placement decoding.

Unlike the sequence policy π^{s} which is selecting over a fixed set of candidate boxes for each time step during an episode, the candidate pool for the placement policy π^{p} is changing from time step to time step, which makes it not quite suitable to model π^{p} using another pointer network (Vinyals et al., 2015) with nonparametric softmax.

Therefore, we deploy a placement network (parameterized by $\boldsymbol{\theta}^{\text{p}}$) that takes in the query vector $\mathbf{q}^{\text{p}}_t$ and output action probabilities with standard parametric softmax. More specifically, the placement network $\boldsymbol{\theta}^{\text{p}}$ outputs logits $\bar{\mathbf{l}}^{\text{p}}_t = \boldsymbol{\theta}^{\text{p}}(\mathbf{q}^{\text{p}}_t)$ of size $O \times W$. If a placement (o_i, y_j) is infeasible that after being placed the current box will cut through walls of the container, its corresponding entry in the logit vector $\bar{\mathbf{l}}^{\text{p}}_{t,i\times j}$ will be masked as $-\infty$, while all other feasible entries will be processed as $C \cdot \tanh(\bar{\mathbf{l}}^{\text{p}}_{t,\cdot})$. This processed logit vector $\mathbf{l}^{\text{p}}_t$ is then used to give out action probabilities for placement selection

$$\pi^{\text{p}}\big(\cdot \,\big|\, f^{\text{p}}_{\mathcal{I}}(s_{1:t}, p_{1:t-1}; \boldsymbol{\theta}^{\text{e}}); \boldsymbol{\theta}^{\text{p}}\big) = \text{softmax}(\mathbf{l}^{\text{p}}_t). \quad (13)$$

As in sequence selection, the placement p_t is sampled during training while chosen greedily during evaluation.

2.5. Policy Gradient

So far we have been discussing that given an input set $\mathcal{I}$, how the sequence agent π^{s} and the placement agent π^{p} cooperatively yield a solution configuration $\mathcal{C}_\pi = \{s_1, p_1, s_2, p_2, \ldots, s_N, p_N\}$ in an interleaving manner over N time steps (π denotes the aggregation of π^{s} and π^{p}). In order to train this system such that π^{s} and π^{p} could cooperatively maximize the final utility $r_{\text{u}} \in [0, 1]$, which is equivalent to minimizing the cost $c(\mathcal{C}_\pi) = 1 - r_{\text{u}}(\mathcal{C}_\pi)$, we define the overall loss function as the expected cost of the configurations generated by following π:

$$\mathcal{J}(\boldsymbol{\theta}|\mathcal{I}) = \mathbb{E}_{\mathcal{C}_\pi \sim \pi_{\boldsymbol{\theta}}}\left[c(\mathcal{C}_\pi|\mathcal{I})\right], \quad (14)$$

which is optimized by following the REINFORCE gradient estimator (Williams, 1992)

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}|\mathcal{I}) = \mathbb{E}_{\mathcal{C}_\pi \sim \pi_{\boldsymbol{\theta}}} \Big[\Big(c(\mathcal{C}_\pi|\mathcal{I}) - b(\mathcal{I}) \Big) \cdot \sum_{t=1}^{N} \Big(\nabla_{\boldsymbol{\theta}^{\mathrm{e,s}}} \log \pi^{\mathrm{s}}(s_t; \boldsymbol{\theta}^{\mathrm{e,s}}) + \nabla_{\boldsymbol{\theta}^{\mathrm{e,p}}} \log \pi^{\mathrm{p}}(p_t; \boldsymbol{\theta}^{\mathrm{e,p}}) \Big) \Big]. \tag{15}$$

We note that to avoid cluttering, we consume the dependency of π^{s} on $f^{\mathrm{s}}_{\mathcal{I}}(s_{1:t-1}, p_{1:t-1}; \boldsymbol{\theta}^{\mathrm{e}})$ into $\boldsymbol{\theta}^{\mathrm{e}}$ and use $\boldsymbol{\theta}^{\mathrm{e,s}}$ to denote the aggregation of $\boldsymbol{\theta}^{\mathrm{e}}$ and $\boldsymbol{\theta}^{\mathrm{s}}$; the same as for π^{p}.

For the baseline function $b(\mathcal{I})$ in the above equation, we use the greedy rollout baseline proposed by (Kool et al., 2018) as this has been shown to give superior performance. More specifically, the parameters of the best model $\boldsymbol{\theta}^{\mathrm{BL}}$ encountered during training is used to roll out greedily on the training input $\mathcal{I}$ to give out $b(\mathcal{I})$; at the end of every epoch i, the model $\boldsymbol{\theta}^{\mathrm{BL}}$ will be compared against the current model $\boldsymbol{\theta}_i$ and will be replaced if the improvement of $\boldsymbol{\theta}_i$ over $\boldsymbol{\theta}^{\mathrm{BL}}$ is significant according to a paired t-test ($\alpha = 5\%$). More details can be found in (Kool et al., 2018).

2.6. Prioritized Oversampling

Based on the observations from preliminary experiments, we suspect that the placement policy could have a more local effect than the sequence policy on the overall quality of the resulting configuration: we found that in hard testing scenarios, a sub-optimal early sequence selection made by π^{s} would make it challenging for the overall system to yield a satisfactory solution configuration; while given a box to place, the effect of a sub-optimal decision of π^{p} usually do not propagate far. Another obsevation that leads to this suspision is that those testing samples on which the trained system perform poorly are usually those that have a more strict requirement on the sequential order of the input boxes.

However, in our problem setup, the agent would only receive an evaluation signal upon the completion of a full episode, which involves subsequently rolling out π^{s} and π^{p} over multiple time steps, with sequence selection and placement selection interleavingly conditioned and dependent on each other. This makes it not quite suitable to use schemes such as utilizing a counterfactual baseline (Foerster et al., 2018) to address multi-agent credit assignment, especially that in our case marginalizing out the effect of a sequence action s_t is not straightforward and could be expensive.

So instead, inspired by prioritized experience replay (Schaul et al., 2016) for off-policy learning, we propose prioritized oversampling (PO) as a general technique to speed up on-policy training. The general idea of prioritized oversampling is to give the agent a second chance of learning those hard examples on which it had performed poorly during regular training. Following the above discussions, we focus on finding better sequence selections for those hard samples during the second round of learning. To achieve this we first follow a beam-search-based scheme; but we later found that a much simpler re-learning procedure could bring performance gains on par with the rather complicated beam-search-based training, therefore we choose the simple alternative as our final design choice.

Prioritized oversampling is carried out in two stages: collect and re-learn. First, during training, when learning on a batch $\mathcal{B}$ of batch size B, the advantages of this batch are calculated as $c(\mathcal{C}_\pi|\mathcal{I}) - b(\mathcal{I})$ (Eq.15). We note that the lower the advantage the better is the configuration $\mathcal{C}_\pi$, since here we use the notion of costs instead of rewards. From those training samples, a PO batch $\mathcal{B}^{\mathrm{PO}}$ of size B^{PO} (usually $B^{\mathrm{PO}} \ll B$) is then collected that contains those with the highest advantage values hence the worst solution configurations. After a total number of B PO samples have been collected (e.g. after ${}^{B}/_{B^{\mathrm{PO}}}$ steps of normal training), the second step of PO re-learning can be carried out. We note that it might not be necessary to conduct normal training and PO re-learning using the same batch size, while we found out empirically that this lead to more stable training dynamics especially when learning with optimizers with adaptive learning rates (Kingma & Ba, 2015). We also deploy a separate optimizer for PO re-learning and found this to be crucial to stabilize learning, as the statistics of the PO batches could differ dramatically from the normal training batches.

As for the PO re-learning, we experiment with two types of learning strategies: beam-search-based training and normal training. In beam-search-based training with a beam size of K, for each PO sample, K instead of 1 solution configurations will be generated by the end of an episode. More specifically, at $t = 1$, π^{s} will sample K^2 box indices $\{s_{1,1}, s_{1,2}, \cdots, s_{1,K^2}\}$. These will all be given to π^{p} which will select a placement for each $s_{1,k}$, out of which K candidates with the highest $\pi^{\mathrm{s}}(s_{1,k})\pi^{\mathrm{p}}(p_{1,k})$ will be selected to be passed as the starting configuration to the next time step. In all time steps $t > 1$, π^{s} will receive K candidate starting configurations, and it will sample for each of these K configurations K next box indices. These K^2 box indices will be passed to π^{p} to select one placement for each. Pruning will again be conducted according to $\prod_{\tau=1}^{t} \pi^{\mathrm{s}}(s_\tau)\pi^{\mathrm{p}}(p_\tau)$ from which K candidates are left. From preliminary experiments with this setup, we found that those K final trajectories picked out by beam-search do not necessarily yield better performance, we suspect that the reason is that the pruning criteria used here does not necessarily translate to good utility. We then adjust the algorithm that after the K configurations have been generated at the end of episodes, we only choose the one with the best advantage to train the model. We found that this give better performance. However, we found that a much simpler training scheme, that instead

of conducting training-time beam-search, simply conduct regular training on the PO samples during the re-learning phase could already bring performance gains, albeit being much simpler in concept and in implementation. Therefore we choose the latter strategy as the final design choice.

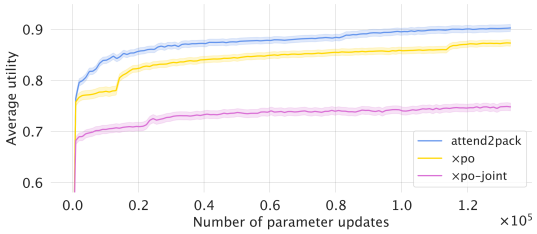
We note that in order to compensate for oversampling, during the normal training of batch $\mathcal{B}$, we scale down the losses of each of those samples that are collected into $\mathcal{B}^{PO}$ by η; while during PO re-learning, the coefficient for those PO samples is set to $1 - \eta$. η is calculated by tracking a set of running statistics. We denote the samples in $\mathcal{B}$ with positive advantages hence relatively inferior performance as $\mathcal{B}^+$, and denote its size as B^+. Then we keep track of the running statistics (with momentum β) of the sample size B^+ and of the average advantages of $\mathcal{B}^+$, denoted as B^+_β and a^+_β respectively, which are updated by: $a^+_\beta = {}^{(\beta \cdot B^+_\beta \cdot a^+_\beta + (1-\beta) \cdot B^+ \cdot a^+)}/_{(\beta \cdot B^+_\beta + (1-\beta) \cdot B^+)}$, $B^+_\beta = \beta \cdot B^+_\beta + (1-\beta) \cdot B^+$. We conduct the same running statistics tracking for the PO batches $\mathcal{B}^{\text{PO}}$ from which we get a^{PO}_β. Then we get the coefficient $\eta = \max(\min(1 - {}^{a^+_\beta}/_{a^{\text{PO}}_\beta}, 1.0), 0.5)$.

3. Experiments

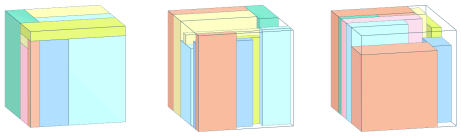
3.1. Experimental Setup

When conducing experiments to validate our proposed method, we do not find it straightforward to compare against previous works, as the datasets used in previous evaluations vary from one to another. The main varying factors are the size of the bin, the number of boxes to pack, the range of the size of the boxes, as well as how the box dimensions are sampled. In order to offer as a valid benchmark, in this paper we conduct ablations as well as careful studies against a variety of previous state-of-the-art methods on BPP, comparing to each according to its specific setups. Unless otherwise mentioned, we run 2 trainings under 2 non-tuned random seeds for each training configuration.

Across all experiments, we use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of $1\text{e}{-4}$ and a batch size of $B = 1024$, and each epoch contains 1024 such minibatches. After every epoch, evaluation is conducted on a fixed set of validation data of size 1e4. The plots we present in this section are all of the performance of these evaluations during training, while the final testing results presented in tables are on testing data of size 1e4 that are drawn independently from the training/validation data, and the testing results of the best agent out of the 2 runs are presented in the tables. A fixed training budget of 128 epochs is set for our full model. For those experiments with prioritized oversampling (PO), a PO batch of size $B^{\text{PO}} = 16$ is collected from each training batch, which means that a full batch ready for PO re-learning would be gathered every ${}^{B}/_{B^{\text{PO}}} = 64$ training steps. This means that after an epoch of 1024 training steps, ${}^{1024}/_{64} = 16$ more PO training steps would have been conducted than setups without PO. Over a total number of 128 epochs this sums up to $128 \cdot 16 = 2 \cdot 1024$, equals to 2 full epochs of more training steps. Thus to ensure a fair comparison, we set the total number of epochs for those setups without PO to 130.



(a) Average utility obtained in evaluation during training, each plot shows mean $\pm$ 0.1 $\cdot$ standard deviation.



(b) attend2pack (c) ×po (d) ×po-joint

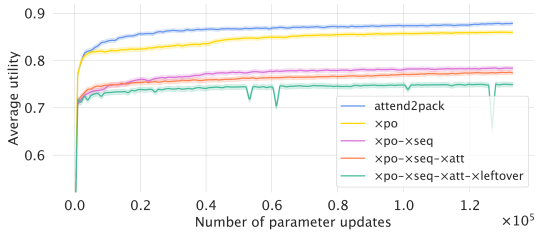
Figure 1. Plots (Fig.1a) and visualizations (Fig.1b-1d) for Sec.3.2, on dataset generated by cutting $10 \times 10 \times 10$ bin into 10 boxes.

The raw box dimensions are normalized by the maximum size of the box edges before fed into the deep network. As for the network, the initial linear layer FF contains 128 hidden units, which is followed by 3 multi-head (with $M = 8$ heads) self-attention layers with an embedding size of $d = 128$. The costs c are scaled up by 3.0 before used to calculate losses. Other important hyperparameters are $C = 10$, $\beta = 0.95$.

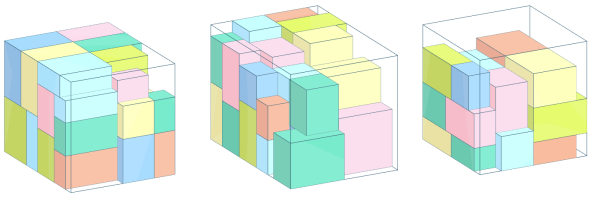
3.2. Ablation Study on Action Space

We first conduct an ablation study to evaluate the effectiveness of the action space decomposition, as well as the effect of utilizing PO. For this experiment, we use the data setup as in (Laterre et al., 2019), where boxes are generated by cutting a bin of size $10 \times 10 \times 10$, and the resulting box edges could take any integer in the range $[1 \,..\, 10]$. (Laterre et al., 2019) conducted experiments by cutting the aforementioned bin into 10, 20, 30, 50 boxes. Here we conduct ablations on the 10 boxes dataset, and a full comparison to this state-of-the-art algorithm (Laterre et al., 2019) will be presented in Sec.3.4.

Our full method is denoted attend2pack, which we compare against a ×po agent trained without utilizing PO, and a ×po-joint agent directly trained on the joint action space, with one softmax policy network directly outputting action probabilities over the full action space. The plots of eval-



(a) Average utility obtained in evaluation during training, each plot shows mean ± 0.1 · standard deviation.



(b) attend2pack (c) ×po-×seq-×att -×leftover (d) ×po-×seq-×att -×leftover cut

Figure 2. Results for Sec.3.3, on dataset generated by randomly sampling the edges of 24 boxes from $[2\,..\,5]$, with a bin of $W \times H = 10 \times 10$. Fig.2d shows the packing configuration of Fig.2c after being processed to compare with palletizing in onlin-BPP, where the items at least partially outside of the $10 \times 10 \times 10$ bin is removed, after which the packing is rotated such that the rear wall of the bin is now at the bottom of the pallet.

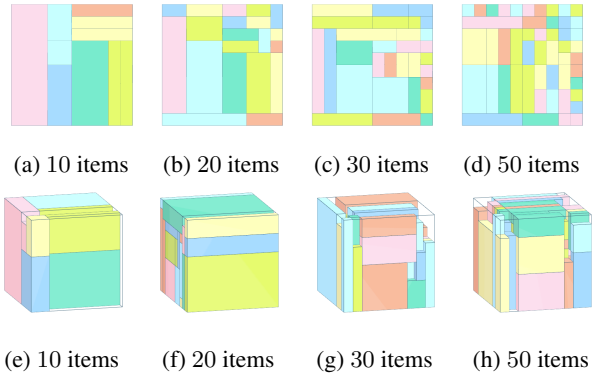
uation during training are shown in Fig.1a, which shows that decomposing the joint combinatorial action space does boost learning, and PO helps to further improve training efficiency. Packing results are also visualized in Fig.1b-1d.

3.3. Ablation Study in the Context of Online BPP

We conduct another set of ablation studies in the context of online-BPP against the state-of-the-art learning-based solution in this domain (Zhao et al., 2021). As discussed in the introduction, in online-BPP, the agent is only aware of the dimensions of the box at hand. Since our algorithm targets the offline-BPP problem, we strip our method down to the strict online setting and also along the way evaluate the effects of several components of our proposed method.

(Zhao et al., 2021) conducted experiments with boxes generated by cut and by random sampling. As comparing to their

[2]We note here the performance of RR are without MCTS simulations, as these results are presented numerically in (Laterre et al., 2019), while the results with MCTS simulations are only presented in box plots. In the box plots, the RR method with MCTS simulation is compared against plain MCTS (Browne et al., 2012), the Lego heuristic (Hu et al., 2017), and the GUROBI solver (Gurobi Optimization, 2018), where RR shows superior performance against all of these methods. By roughly inferring the numerical results from the box plots we can conclude that our method still outperforms RR with MCTS simulations except for the scenario in 3D with 10 boxes.



(a) 10 items (b) 20 items (c) 30 items (d) 50 items

(e) 10 items (f) 20 items (g) 30 items (h) 50 items

Figure 3. Visualizations for Sec.3.4, on dataset generated by cutting a square of 10×10 (2D) or a bin of $10 \times 10 \times 10$ (3D) into 10, 20, 30, 50 items, with the box edges ranging within $[1\,..\,10]$.

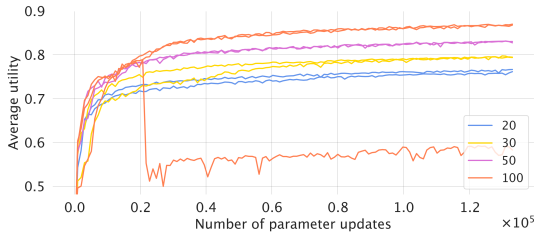
Table 1. Performance comparison on the cut dataset with ranked reward (RR) (Laterre et al., 2019). The first two columns show results in r_{RR}, while the last column gives r_{u} of our method to ease comparisons of future works [2].

	RR (r_{RR})	ATTEND2PACK (r_{RR})	ATTEND2PACK (r_{u})
2D-10	0.953 ± 0.027	$\mathbf{0.995} \pm 0.015$	$\mathbf{0.991} \pm 0.028$
2D-20	0.948 ± 0.020	$\mathbf{0.997} \pm 0.012$	$\mathbf{0.994} \pm 0.022$
2D-30	0.954 ± 0.015	$\mathbf{0.999} \pm 0.006$	$\mathbf{0.999} \pm 0.011$
2D-50	0.960 ± 0.016	$\mathbf{1.000} \pm 0.000$	$\mathbf{1.000} \pm 0.000$
3D-10	0.903 ± 0.060	$\mathbf{0.953} \pm 0.042$	$\mathbf{0.932} \pm 0.060$
3D-20	0.850 ± 0.032	$\mathbf{0.911} \pm 0.034$	$\mathbf{0.872} \pm 0.046$
3D-30	0.844 ± 0.030	$\mathbf{0.904} \pm 0.033$	$\mathbf{0.863} \pm 0.044$
3D-50	0.838 ± 0.034	$\mathbf{0.926} \pm 0.024$	$\mathbf{0.893} \pm 0.032$

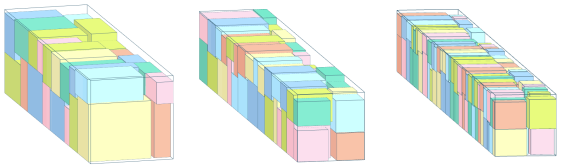
cut experiments is not easily feasible, we conduct experiments using their random data-generation scheme: each box edge is sampled from the integers in the range of $[2\,..\,5]$, while the bin is of size $10 \times 10 \times 10$. Their evaluation criteria is to count the average number of boxes that can be placed inside the bin, which is a bit different from our training scheme. Thus we compare the final results against theirs in the following way: based on the total volume of the bin and the expected volume of each box, we can calculate that the maximum expected number of boxes that can be packed is 24. We thus set $N = 24$ when generating datasets. After training, we remove all the boxes that are at least partially out of the $10 \times 10 \times 10$ bin and count the number of boxes still left inside of the bin to compare to (Zhao et al., 2021).

The training configurations under this study are:

- attend2pack: Our full method;
- ×po: Our full method but without PO;
- ×po-×seq: No PO training and no sequence policy learning. While this is very similar to the online-BPP, since self-attention are used to calculate the box embeddings $\mathcal{B}$, each individual box embedding $\mathbf{b}^n$ still



(a) Average utility obtained in evaluation during training, here plots each individual run for each configuration.



(b) 30 items (c) 50 items (d) 100 items

Figure 4. Plots (Fig.4a) and visualizations (Fig.4b-4d) for Sec.3.5, on dataset generated by randomly sampling the edges of 20, 30, 50, 100 boxes from $[20\,..\,80]$, with a bin of $W \times H = 100 \times 100$.

Table 2. Comparisons in Sec.3.5 on r_{u} against GA(Wu et al., 2010), EP(Crainic et al., 2008), MTSL(Duan et al., 2019), CQL(Li et al., 2020) and MM(Jiang et al., 2021).

	GA	EP	MTSL	CQL	MM	ATTEND2PACK
20	0.683	0.627	0.624	0.670	0.718	**0.767** $\pm$ 0.036
30	0.662	0.638	0.601	0.693	0.755	**0.797** $\pm$ 0.032
50	0.659	0.663	0.553	0.736	0.813	**0.831** $\pm$ 0.023
100	0.624	0.675	–	–	0.844	**0.870** $\pm$ 0.016

carries information about other boxes;

- ×po-×seq-×att: On top of the former setup, we replace the attention module in $\boldsymbol{\theta}^{\mathrm{e}}$ with a residual network with approximately the same number of parameters. But still this setup cannot be regarded as a strict online setting, since the encoding function for placement $f_{\mathcal{I}}^{\mathrm{p}}$ (Eq.12) contains the leftover embeddings which still gives information about the upcoming boxes.

- ×po-×seq-×att-×leftover: A strict online-BPP setup where the leftover embeddings are removed from $f_{\mathcal{I}}^{\mathrm{p}}$ from the former setup.

The results of this ablation are ploted in Fig.2a. We can see that both the leftover embeddings and the attention module brings performance gains. It can also be concluded from the big performance gap between ×po and ×po-×seq that the sequence policy is able to produce reasonable sequence orders. We observe that PO is again able to improve performance.

After processing the packing results of the agent of ×po-×seq-×att-×leftover with the aforementioned procedures, we compare against the performance of (Zhao et al., 2021), which packs on average 12.2 items with a utility of 54.7% (with a lookahead number of 5, the utility achieved by their method still does not surpass 60%); while the strictly online-BPP version of our method on average is able to pack 15.6 items with a utility of 67.0%, achieving state-of-the-art performance in the online-BPP domain. Visualizations of the packing results are shown in Fig.2b-2d.

3.4. 2D/3D BPP on Boxes Generated by Cut

As introduced in Sec.3.2, (Laterre et al., 2019) (RR) presented the state-of-the-art performance on datasets generated by cut in both the 2D and the 3D domain. More specifically, the boxes are generated by cutting a $10 \times 10 \times 10$ bin (or a 10×10 square for 2D) into 10, 20, 30 or 50 boxes (pseudo-code in Appendix I). They use a different reward than untility r_{u} defined as $r_{\mathrm{RR}} = {2 \cdot (\sum_{n=1}^{N} l^n w^n)^{1/2}}/{(L_\pi + W_\pi)}$ for 2D and $r_{\mathrm{RR}} = {3 \cdot (\sum_{n=1}^{N} l^n w^n h^n)^{1/3}}/{(L_\pi W_\pi + W_\pi H_\pi + H_\pi L_\pi)}$ for 3D. For the following experiments, we still train our method with the cost of $1 - r_{\mathrm{u}}$ but we compare the results in r_{RR}. We present the visualizations of the packing results are shown in Fig.3, and the final testing results in Table 1. We can observe that our method achieves state-of-the-art in this data domain.

3.5. 3D BPP on Randomly Generated Boxes

We continue to conduct comparisons against the state-of-the-art algorithm (Jiang et al., 2021) on randomly generated boxes, in which $W \times H = 100 \times 100$ for the bin, while the edges of boxes are sampled from integers in the range of $[20\,..\,80]$. (Jiang et al., 2021) present experimental results with box numbers of 20, 30, 50 and 100, on which the learning curves of our full method attend2pack are shown in Fig.4a. We can see that the performance improves stably during training, except for one of the trainings for 100 boxes which encounters catastrophic forgetting. We suspect that for more number of boxes each episode would involve more rollout steps thus an undesirable interaction between π^{s} and π^{p} is more likely to propagate to unexpected gradient updates. Since we observe larger variations for the training with 100 boxes, here we add one more run to give a more accurate presentation of the performance of our algorithm on this task. We present the visualizations of the packing results in Fig.4b-4d, and the final testing results on 1e4 testing samples in Table 2, where we can see that our method achieves state-of-the-art performance in this data domain.

3.6. Conclusions and Future Work

To conclude, this paper proposes a new model attend2pack that achieves state-of-the-art performance on BPP. Possible future directions include extending this framework to multi-bin settings, evaluating it under more packing constraints, and more sophisticated solutions for credit assignment.

References

Abdel-Basset, M., Manogaran, G., Abdel-Fatah, L., and Mirjalili, S. An improved nature inspired meta-heuristic algorithm for 1-d bin packing problems. *Personal and Ubiquitous Computing*, 22(5):1117–1132, 2018.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *Advances in NIPS 2016 Deep Learning Symposium*, pp. arXiv preprint arXiv:1607.06450, 2016.

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

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Busoniu, L., Babuska, R., and De Schutter, B. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

Cambazard, H. and O'Sullivan, B. Propagating the bin packing constraint using linear programming. In *International Conference on Principles and Practice of Constraint Programming*, pp. 129–136. Springer, 2010.

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

Duan, L., Hu, H., Qian, Y., Gong, Y., Zhang, X., Wei, J., and Xu, Y. A multi-task selected learning approach for solving 3d flexible bin packing problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1386–1394, 2019.

Dyckhoff, H. A new linear programming approach to the cutting stock problem. *Operations Research*, 29(6):1092–1104, 1981.

Fanslau, T. and Bortfeldt, A. A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, 22(2):222–235, 2010.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Gehring, H. and Bortfeldt, A. A parallel genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 9(4): 497–511, 2002.

Gurobi Optimization, I. Gurobi optimizer reference manual, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Hu, H., Zhang, X., Yan, X., Wang, L., and Xu, Y. Solving a new 3d bin packing problem with deep reinforcement learning method. *Workshop on AI application in E-commerce, Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. arXiv preprint arXiv:1708.05930, 2017.

Huang, W. and He, K. A caving degree approach for the single container loading problem. *European Journal of Operational Research*, 196(1):93–101, 2009.

Jiang, Y., Cao, Z., and Zhang, J. Solving 3d bin packing problem via multimodal deep reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1548–1550, 2021.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic gradient descent. In *ICLR: International Conference on Learning Representations*, pp. 1–15, 2015.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., Silver, D., and Graepel, T. A unified game-theoretic approach to multiagent reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 4193–4206, 2017.

Laterre, A., Fu, Y., Jabri, M. K., Cohen, A.-S., Kas, D., Hajjar, K., Dahl, T. S., Kerkeni, A., and Beguir, K. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. *Workshop on Reinforcement Learning in Games, Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI-19*, pp. arXiv preprint arXiv:1807.01672, 2019.

Lewis, C. Linear programming: Theory and applications. *Whitman College Mathematics Department*, 2008.

Lewis, H. R. Computers and intractability: A guide to the theory of np-completeness, 1983.

Li, D., Ren, C., Gu, Z., Wang, Y., and Lau, F. Solving packing problems by conditional query learning, 2020. URL https://openreview.net/forum?id=BkgTwRNtPB.

Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pp. 157–163. Elsevier, 1994.

Loh, K.-H., Golden, B., and Wasil, E. Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research*, 35(7):2283–2291, 2008.

Lysgaard, J., Letchford, A. N., and Eglese, R. W. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

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

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

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. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Panait, L. and Luke, S. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Salem, K. H. and Kieffer, Y. New symmetry-less ilp formulation for the classical one dimensional bin-packing problem. In *International Computing and Combinatorics Conference*, pp. 423–434. Springer, 2020.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pp. 1057–1063. Citeseer, 1999.

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

Tanaka, T., Kaneko, T., Sekine, M., Tangkaratt, V., and Sugiyama, M. Simultaneous planning for item picking and placing by deep reinforcement learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9705–9711. IEEE, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6000–6010, 2017.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pp. 2692–2700, 2015.

Wang, X., Thomas, J. D., Piechocki, R. J., Kapoor, S., Santos-Rodriguez, R., and Parekh, A. Self-play learning strategies for resource assignment in open-ran networks. *arXiv preprint arXiv:2103.02649*, 2021.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wu, Y., Li, W., Goh, M., and de Souza, R. Three-dimensional bin packing problem with variable bin height. *European journal of operational research*, 202(2):347–355, 2010.

Yang, N. and Liu, J. J. Sequential markov games with ordered agents: A bellman-like approach. *IEEE Control Systems Letters*, 4(4):898–903, 2020.

Young-Dae, H., Young-Joo, K., and Ki-Baek, L. Smart pack: Online autonomous object-packing system using rgb-d sensor data. *Sensors*, 20(16):4448, 2020.

Zhao, H., She, Q., Zhu, C., Yang, Y., and Xu, K. Online 3d bin packing with constrained deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):741–749, May 2021.

Zhu, W., Oon, W.-C., Lim, A., and Weng, Y. The six elements to block-building approaches for the single container loading problem. *Applied Intelligence*, 37(3):431–445, 2012.

Zhu, W., Zhuang, Y., and Zhang, L. A three-dimensional virtual resource scheduling method for energy saving in cloud computing. *Future Generation Computer Systems*, 69:66–74, 2017.

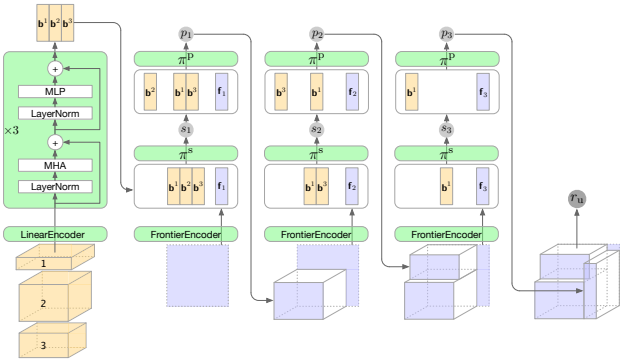


Figure 5. Schematic plot of the proposed attend2pack pipeline.

A. Appendix: Schematic Plot of attend2pack

A schematic plot of our proposed attend2pack algorithm is shown in Fig.5.

B. Appendix: Schematic Plot of Glimpse

A schematic plot of the glimpse operation (Eq.7-11) during sequence decoding is shown in Fig.6.

C. Appendix: Network Architecture

In this section we present the details of the network architecture used in our proposed attend2pack model.

The embedding network $\boldsymbol{\theta}^{\text{e}}$ firstly encodes a $N \times 3$ input into a $N \times 128$ embedding for each data point using an initial linear encoder with 128 hidden units (in preliminary experiments, we found that applying random permutation to the input list of boxes gave at least marginal performance gains over sorting the boxes, so we adhere to random permutation in all our experiments; also we ensure that all boxes are rotated such that $l^n \leq w^n \leq h^n$ before they are fed into the network). This is then followed by a multi-head self-attention module that contains 3 self-attention layers with 8 heads, with each layer containing a self-attention sub-layer (Eq.4) and an MLP sub-layer (Eq.5). The self-attention sub-layer contains a LayerNorm followed by a standard MHA module (Vaswani et al., 2017) with an embedding dimension of 128 and 8 heads, whose output is added to the input of the self-attention sub-layer with a skip connection. The MLP module contains a fully-connected layer with 512 hidden nodes followed by ReLU activation, which is then followed by another fully-connected layer with 128 hidden nodes. The output of the last fully-connected layer is added to the input of the MLP sub-layer with a skip connection.

As for the frontier encoding network, for a bin with $W \times H = 10 \times 10$ in the 3D situation (Sec.3.2, 3.3, 3.4), the stacked two lastest frontiers (Sec.2.2.2) of size $2 \times 10 \times 10$ is frstly fed into a 2D convolutional layer with 2 input channels and 8 output channels, with kernel size, stride and padding of 3, 1 and 1 respectively. This is followed by LayerNorm and ReLU activation. Then there is another convolutional layer with number of input channels of 8 and number of output channels of 8, and with kernel size, stride and padding of 5, 1 and 1 respectively. This is again followed by LayerNorm and ReLU activation. The output is then flattened into a vector of size 512, which is then passed into a fully-connected layer with 128 hidden units, which is again followed by LayerNorm and ReLU activation.

For the 2D situation with a 2D bin with $W = 10$ (Sec.3.4), the stacked last two frontier is of size 2×10, and is firstly fed into a 1D convolutional layer with number of input channels 2 and number of output channels 8, with kernel size, stride and padding of 3, 1 and 1 respectively. This is followed by LayerNorm and ReLU activation. The output is then flattened into a vector of size 80, then passed into a fully-connected layer with 128 hidden units, which is again followed by LayerNorm and ReLU activation.

For a bin with $W \times H = 100 \times 100$ in the 3D situation (Sec.3.5), the stacked last two frontier of size $2 \times 100 \times 100$ is firstly fed into a 2D convolutional layer with number of input channels 2 and number of output channels 4, with kernel size, stride and padding of 3, 2 and 1 respectively. Then there is another two convolutional layers both with number of input channels of 4 and number of output channels of 4, and with kernel size, stride and padding of 3, 2 and 1 respectively. This is again followed by LayerNorm and ReLU activation. The output is then flattened into a vector of size 676, which is then passed into a fully-connected layer with 128 hidden units, which is again followed by LayerNorm and ReLU activation. We note that that in the experiments in Sec.3.5, for the experiments with 100 number of boxes with bin of $W \times H = 100 \times 100$, instead of using the stacked last 2 frontiers to be the input to the frontier encoding network as in all our other experiments, we only use the most recent frontier due to limited memory.

The sequence decoding parameters $\boldsymbol{\theta}^{\text{s}}$ are presented in de-

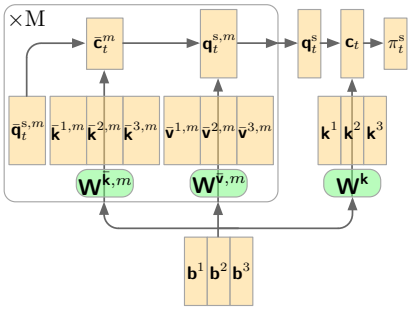
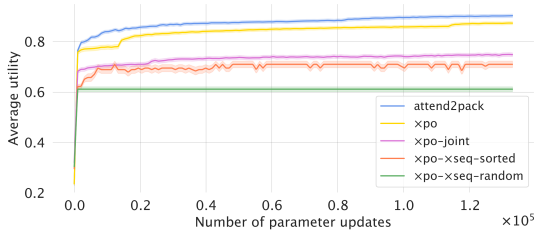
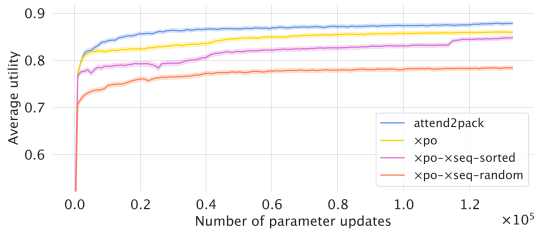


Figure 6. Schematic plot of the glimpse operation during sequence decoding (Sec.2.3: Eq.7-11).



(a) Additional experiments on the dataset used in Sec.3.2.



(b) Additional experiments on the dataset used in Sec.3.3 (we note that here ×po-×seq-random corresponds to ×po-×seq in Fig.2).

Figure 7. Results for App.D, where we plot the average utility obtained in evaluation during training, each plot shows mean $\pm$ 0.1 $\cdot$ standard deviation.

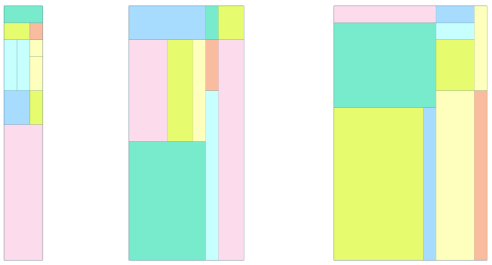
tails in Sec.2.3. We note an implementation detail is omitted in Sec.2.3 and 2.4, that for both the encoding functions $f_{\mathcal{I}^{\mathrm{s}}}$ (Eq.6) and $f_{\mathcal{I}^{\mathrm{p}}}$ (Eq.12), each of their input component will go through a linear projection before passing through the final $\langle\rangle$ operation

$$f^{\mathrm{s}}_{\mathcal{I}}(s_{1:t-1}, p_{1:t-1}; \boldsymbol{\theta}^{\mathrm{e}}) = \left\langle \mathbf{W}^{\text{s-leftover}} \langle \mathcal{B} \setminus \mathbf{b}^{s_{1:t-1}} \rangle, \mathbf{W}^{\text{frontier}} \mathbf{f}_t \right\rangle, \tag{16}$$

$$f^{\mathrm{p}}_{\mathcal{I}}(s_{1:t}, p_{1:t-1}; \boldsymbol{\theta}^{\mathrm{e}}) = \left\langle \mathbf{W}^{\text{selected}} \mathbf{b}^{s_t}, \mathbf{W}^{\text{p-leftover}} \langle \mathcal{B} \setminus \mathbf{b}^{s_{1:t}} \rangle, \mathbf{W}^{\text{frontier}} \mathbf{f}_t \right\rangle, \tag{17}$$

where all the weight matrices $\mathbf{W}$ are of size 128×128.

As for the placement decoding network $\boldsymbol{\theta}^{\mathrm{p}}$, it fistly contains a fully-connected layer with 128 hidden units followed by LayerNorm and ReLU activation. This is followed by another fully-connected layer with 128 hidden units with LayerNorm and ReLU activation. Then the final output fully-connected layer maps 128-dimensional embeddings to placement logits. The logit vector is of size 6×10 for 3D packing with bin of $W \times H = 10 \times 10$ (6 possible orientations) (Sec.3.2, 3.3, 3.4), 2×10 for 2D packing with $W = 10$ (2 possible orientations) (Sec.3.4), and 6×100 for 3D packing with bin of $W \times H = 100 \times 100$ (6 possible orientations) (Sec.3.5).



(a) $L = 3$ (b) $L = 9$ (c) $L = 12$

Figure 8. Visualizations for App.E, on dataset generated by cutting a rectangle into 10 boxes, with $W = 15$ and L uniformly sampled from $[2\,..\,15]$, where the box edges range within $[1\,..\,15]$.

D. Appendix: Additional Ablation Study on the Sequence Policy

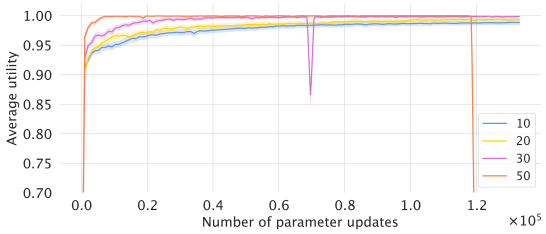
We conduct an additional set of comparisons on top of the ablation studies in Sec.3.2,3.3. Specifically, we compare additionally with training setups where the sequential order of the boxes are not learned, but is fixed by random permutation or sorted by a simple heuristic: sorted from large to small by volumn $l^n \cdot w^n \cdot h^n$, where ties are broken by w^n/h^n then by l^n/w^n (we note that $l^n \leq w^n \leq h^n$ (App.C)). The setups without learning the sequence order are denoted as ×po-×seq-random (fixed random order) and ×po-×seq-sorted (fixed sorted order) respectively. The experimental results are shown in Fig.7, from which we can observe that for the random dataset (Fig.7b), the simple heuristic of following a sorted sequential order could lead to satisfactory performance together with a learned placement policy; while this sorted heuristic could not yield strong performance for the cut dataset (Fig.7a). Therefore learning the sequence policy should be considered when dealing with an arbitrary dataset configuration.

E. Appendix: Additional Experiments on 2D BPP on Boxes Generated by Cut

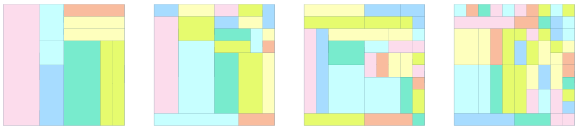
We conduct an additional set of experiments comparing against (Wang et al., 2021), which adopts the RR algorithm

Table 3. Comparisons in Sec.E on r_L against HVRAA(Zhu et al., 2017), Lego heuristic(Hu et al., 2017), MCTS(Browne et al., 2012) and RR(Laterre et al., 2019). The r_{u} obtained by our method is presented in the last row.

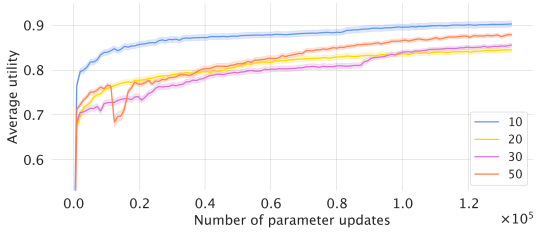
HVRAA	0.896 ± 0.239
LEGO HEURISTIC	0.737 ± 0.349
MCTS	0.936 ± 0.097
RR	0.964 ± 0.160
ATTEND2PACK	$\mathbf{0.971} \pm 0.051$
ATTEND2PACK (r_{u})	$\mathbf{0.971} \pm 0.051$



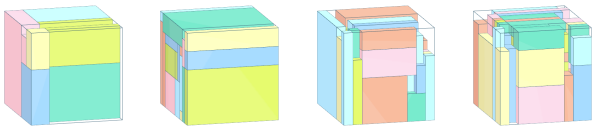
(a) Average utility obtained in evaluation during training, each plot shows mean ± 0.1 · standard deviation.



(b) 10 items (c) 20 items (d) 30 items (e) 50 items



(f) Average utility obtained in evaluation during training, each plot shows mean ± 0.1 · standard deviation.



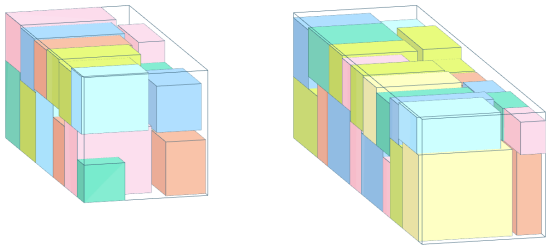
(g) 10 items (h) 20 items (i) 30 items (j) 50 items

Figure 9. Visualizations for Sec.3.4, on dataset generated by cutting a square of 10×10 (2D) or a bin of $10 \times 10 \times 10$ (3D) into 10, 20, 30, 50 items, with the box edges ranging within $[1\,..\,10]$.

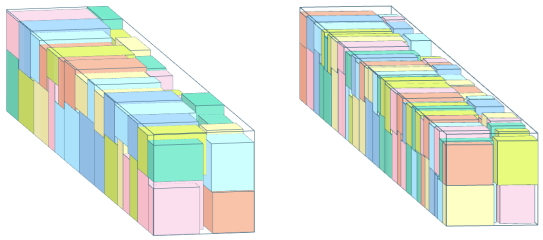
on a 2D BPP dataset generated by cutting 2D rectangles into 10 boxes, with $W = 15$ and L uniformly sampled from $[2\,..\,15]$, where the box edges are guaranteed to be within $[1\,..\,15]$. The reward measure used in (Wang et al., 2021) is $r_L = L_\pi / \frac{(\sum_{n=1}^{N} l^n w^n)}{W}$, where L_π denotes the length of the minimum bounding box that encompass all packed boxes at the end of episodes; as in Sec.3.4, we train our agent attend2pack with the utility reward r_u but compare to the results of (Wang et al., 2021) using their measure r_L. The packing visualizations and the final testing results are shown in Fig.8 and Table 3.

F. Appendix: More Visualizations for Sec.3.4, 3.5

Additional results for Sec.3.4, 3.5 are shown in Fig.9, 10.



(a) 20 items (b) 30 items



(c) 50 items (d) 100 items

Figure 10. Visualizations for Sec.3.5, on dataset generated by randomly sampling the edges of 20, 30, 50, 100 boxes from $[20\,..\,80]$, with a bin of $W \times H = 100 \times 100$.

G. Appendix: Dataset Specifications

We list the specifications of the datasets used in all our presented experiments in Table 4.

H. Appendix: Inference Time

We list the inference time of our presented experiments in Table 5, for which the CPU configuration of our computing resource is: Intel(R) Core(TM) i9-7940X CPU 3.10GHz 128GB, and the GPU configuration is: Nvidia TITAN X (Pascal) GPU with 12G memory.

I. Appendix: Cut Dataset Generation

The algorithm from (Laterre et al., 2019) that cuts a fixed-sized bin into a designated number of cubic items is presented in Alg.1. This algorithm is used to generate the dataset used in the experiments in Sec.3.2, 3.4 and App.E.

Table 4. Specifications of datasets in our presented experiments.

	DATASET TYPE	$L \times W (\times H)$	N	l^i,w^i,(h^i)
SEC.3.2, APP.D	CUT-3D	$10 \times 10 \times 10$	10	$[1\,..\,10]$
SEC.3.3, APP.D	RANDOM-3D	$\cdot \times 10 \times 10$	24	$[2\,..\,5]$
SEC.3.4	CUT-2D	10×10	10, 20, 30, 50	$[1\,..\,10]$
SEC.3.4	CUT-3D	$10 \times 10 \times 10$	10, 20, 30, 50	$[1\,..\,10]$
SEC.3.5	RANDOM-3D	$\cdot \times 100 \times 100$	20, 30, 50, 100	$[20\,..\,80]$
APP.E	CUT-2D	$[2\,..\,15] \times 15$	10	$[1\,..\,15]$

Table 5. Inference time for a batch of size 1024 on GPU (Intel(R) Core(TM) i9-7940X CPU 3.10GHz 128GB with a Nvidia TITAN X (Pascal) GPU with 12G memory) of our presented experiments (we note that the some operations for 3D are written in cuda, while the rest of the code is in python/pytorch).

		$L \times W (\times H)$	N	TIME(S)
2D	SEC.3.4	10×10	10	0.108s
2D	SEC.3.4	10×10	20	0.250s
2D	SEC.3.4	10×10	30	0.448s
2D	SEC.3.4	10×10	50	1.015s
2D	APP.E	$[2\,..\,15] \times 15$	10	0.145s
3D	SEC.3.3, APP.D	$\cdot \times 10 \times 10$	24	0.224s
3D	SEC.3.2,3.4, APP.D	$10 \times 10 \times 10$	10	0.097s
3D	SEC.3.4	$10 \times 10 \times 10$	20	0.185s
3D	SEC.3.4	$10 \times 10 \times 10$	30	0.269s
3D	SEC.3.4	$10 \times 10 \times 10$	50	0.504s
3D	SEC.3.5	$\cdot \times 100 \times 100$	20	0.535s
3D	SEC.3.5	$\cdot \times 100 \times 100$	30	0.805s
3D	SEC.3.5	$\cdot \times 100 \times 100$	50	1.379s
3D	SEC.3.5	$\cdot \times 100 \times 100$	100	2.821s

Algorithm 1 Cutting Bin from (Laterre et al., 2019)

Input: Bin dimensions: length L, width W, height H; minimum length for box edges M; number of boxes N to cut the bin into.
Output: A set of N boxes $\mathcal{S}$.
Initialize box list $\mathcal{S} = \{(L, W, H)\}$.
while $|\mathcal{S}| < N$ **do**
  Sample a box i (l^i, w^i, h^i) from $\mathcal{S}$ by probabilities proportional to the volumes of the boxes; pop box i from $\mathcal{S}$.
  Sample a dimension (e.g. w^i) from (l^i, w^i, h^i) by probabilities proportional to the lengths of the three dimensions.
  Sample a cutting point j from the selected dimension w^i by probabilities proportional to the distances from all points on w^i to the center of w^i. Check that $j \geq M$ and $w^i - j \geq M$, otherwise push the selected box (l^i, w^i, h^i) back into $\mathcal{S}$ and continue to the next iteration.
  Cut box (l^i, w^i, h^i) along w^i from the chosen point j to get two boxes (l^i, j, h^i) and $(l^i, w^i - j, h^i)$ and push those two boxes into $\mathcal{S}$.
end while
return $\mathcal{S}$