# Learning to Solve Grouped 2D Bin Packing Problems in the Manufacturing Industry

Wenxuan Ao
Department of Electronic Engineering, BNRist, Tsinghua University & TsingRoc
Beijing, China
aowx21@mails.tsinghua.edu.cn

Guozhen Zhang
Department of Electronic Engineering, BNRist, Tsinghua University & TsingRoc
Beijing, China
zgz18@mails.tsinghua.edu.cn

Yong Li
Department of Electronic Engineering, Tsinghua University
Beijing, China
liyong07@tsinghua.edu.cn

Depeng Jin
Department of Electronic Engineering, Tsinghua University
Beijing, China
jindp@tsinghua.edu.cn

## ABSTRACT

The two-dimensional bin packing problem (2DBP) is a critical optimization problem in the furniture production and glass cutting industries, where the objective is to cut smaller-sized items from a minimum number of large standard-sized raw materials. In practice, factories manufacture hundreds of customer orders (sets of items) every day, and to relieve pressure in management, a common practice is to group the orders into batches for production, ensuring that items from one order are in the same batch instead of scattered across the production line. In this work, we formulate this problem as the grouped 2D bin packing problem, a bi-level problem where the upper level partitions orders into groups and the lower level solves 2DBP for items in each group. The main challenges are (1) the coupled optimization of upper and lower levels and (2) the high computational efficiency required for practical application. To tackle these challenges, we propose an iteration-based hierarchical reinforcement learning framework, which can learn to solve the optimization problem in a data-driven way and provide fast online performance after offline training. Extensive experiments demonstrate that our method not only achieves the best performance compared to all baselines but is also robust to changes in dataset distribution and problem constraints. Finally, we deployed our method in the ARROW Home factory in China, resulting in a 4.1% reduction in raw material costs. We have released the source code and datasets to facilitate future research.

## CCS CONCEPTS

• **Computing methodologies** → **Reinforcement learning**; • **Applied computing** → **Industry and manufacturing**.


*KDD '23, August 6–10, 2023, Long Beach, CA, USA*

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599860

## KEYWORDS

Grouped bin packing, reinforcement learning, bi-level optimization

**ACM Reference Format:**
Wenxuan Ao, Guozhen Zhang, Yong Li, and Depeng Jin. 2023. Learning to Solve Grouped 2D Bin Packing Problems in the Manufacturing Industry. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3580305.3599860

## 1 INTRODUCTION

In the manufacturing industry, an essential class of tasks is cutting panels of different sizes out of standard-sized raw materials like glass, wood, steel, plastic, etc. This is usually formulated as a two-dimensional bin packing (2DBP) problem, where the panels to be cut are referred to as items, and the standard-sized raw materials are referred to as bins. The objective is to find the best way to arrange and fit the items into the bins without overlap such that the total number of bins, or *bin usage* for short, is minimized. As bin usage directly affects the production cost of a factory, it is crucial to find optimized packing plans. Over the years, many researchers have been developing methods for solving 2DBP, including exact methods [13, 25, 27, 31], heuristic methods [3, 12, 16, 23, 38], and learning-based methods [7, 21].

However, *batched manufacturing*, one of the most common practices in real-life applications, is not considered in the formulation of 2DBP. Specifically, the orders from the customers, which are sets of items, are grouped into batches for manufacturing to limit the maximum number of items in each batch. This is because the items will be mixed together after the manufacturing step, and in the assembling step, the workers need to first sort the items by the orders they belong to and then assemble the items into products, which would be highly challenging if the batch contained too many items. For example, if the batch contains 100 orders and each order contains 100 items, there will be a total of 10,000 items produced in the manufacturing step. And to assemble an order, the workers will have to collect the corresponding 100 items from the 10,000 items, which is time-consuming and impractical. Therefore, it is essential to model the order grouping step and the size constraint in real-life manufacturing. Additionally, how the orders are grouped into batches also significantly impacts the final bin packing cost, as

it directly decides the input items of the following bin packing step and also requires optimization.

We formulate the co-optimization of order grouping and bin packing as grouped 2D bin packing problem (G2DBP), a bi-level problem where in the upper level, we partition the orders into groups with item count limit, while in the lower level, we solve 2DBP for the items of each group. There are two main challenges in solving G2DBP for real-life applications:

- **Coupled optimization of upper and lower level**: In G2DBP, the order grouping plan of the upper level determines the input of the lower level, while the bin packing result of the lower level determines the objective function that the upper level tries to minimize. For one thing, the number of possible order grouping plans is exponential, and it is already challenging for the upper level to find the optimal grouping plan. For another thing, due to the NP-hardness of the lower level bin packing problem [9], the objective function is a complex and highly nonlinear function of the grouping plan, which makes it even more challenging to optimize.
- **High computational efficiency requirement**: In real-life applications, computational efficiency is as important as the solution quality, and it is desired for the method to provide a relatively high-quality solution in a reasonable time. For the upper-level grouping problem, traditional methods like simulated annealing [18], tabu search [34], and grouping genetic algorithm [6] rely on trial and error and cannot efficiently find good solutions. For the lower level 2DBP, generative one-pass heuristics are efficient but provide low-quality solutions, while iterative heuristics provide higher-quality solutions but are inefficient.

To tackle the challenges, we propose an iteration-based hierarchical reinforcement learning framework (IHRL) with two reinforcement learning agents, GroupRL and PermRL, designed for the optimization tasks of the upper and lower levels. Specifically, for the first challenge, we propose a hierarchical encoding scheme and two modification operators in GroupRL, which enable it to extract rich information from input features and to iteratively improve the order grouping plan with the modification operators based on the feedback from the lower level and the input features. For the second challenge, we reformulate the bin packing problem as a permutation problem and design PermRL that can learn permutation strategies from data and provide fast online inference performance. We summarize our contributions as follows:

- To the best of our knowledge, we are the first to formulate and solve the grouped 2D bin packing problem in a data-driven way, which models batched manufacturing, one of the most common practices in the manufacturing industry.
- We propose an iteration-based hierarchical reinforcement learning framework for G2DBP, in which we design two models, GroupRL and PermRL, that are specialized for the tasks of the upper and lower levels and can learn optimization strategies from data.
- We conduct extensive experiments on both real and synthetic datasets. The results show that our method not only outperforms all baseline methods but is also generalizable and robust to distribution shift and change of problem constraints.
- We deploy our method in the factory of ARROW Home, China, which greatly improves material utilization and reduces the raw material cost by 4.1%.

## 2 PRELIMINARIES

### 2.1 Definitions

For clarity, we list the key definitions in the 2D bin packing problem and grouped 2D bin packing problem as follows:

*Bin:* we refer to the raw material sheets as bins, which are rectangular and all of the same size. We denote the width as $W$ and height as $H$.

*Item:* we refer to the parts or panels that need to be cut from bins as items, which are also rectangular but with different sizes. We denote item $i$ as $t_i = (w_i, h_i)$, where $w_i$ and $h_i$ are the width and height of the item.

*Order:* an customer order contains a list of items, $o = \{t_1, t_2, ..., t_n\}$, where $n$ is the size of the order. The items in the same order must be manufactured together.

*Group:* a group is a list of orders, $G = \{o_1, o_2, ..., o_m\}$, where $m$ is the size of the group. The group is the basic unit of manufacturing. As the items need to be sorted out for the assembly of each order after manufacturing, there is a limit on the maximum number of items a group contains.

*Bin packing plan:* it is a way to arrange and fit the given list of items on a list of bins such that the edges of the items are parallel to the edges of the bins and the items do not overlap.

*Grouping plan:* it is a way to partition the given list of orders into several groups.

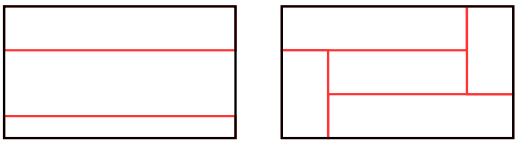


**Figure 1: Guillotine and non-guillotine cuts.**

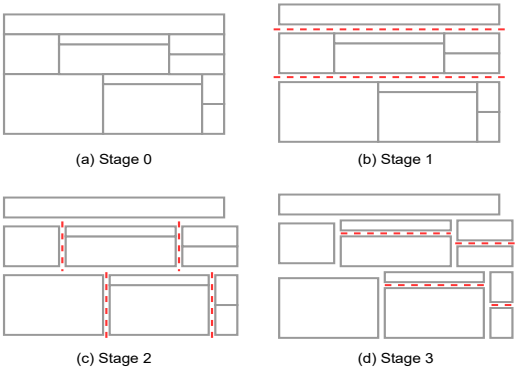


**Figure 2: Guillotine cuts with multiple stages.**

## 2.2 Problem Statement

*2.2.1 2D Bin Packing.* The problem of 2DBP is to find the optimal bin packing plan of the items that minimizes the number of bins. Formally, given an infinite number of bins and a list of items, the objective is to find solution $\{(b_1, x_1, y_1), (b_2, x_2, y_2), ..., (b_n, x_n, y_n)\}$ with $b_i, x_i, y_i \in \mathbb{N}$ to the following optimization problem:

$$\begin{aligned} &\text{minimize} \ \max_{1 \le i \le n} b_i, \\ s.t.\ &\forall i, j,\ 1 \le i, j \le n \ \wedge\ b_i = b_j, \\ &\quad [x_i, x_i + w_i) \cap [x_j, x_j + w_j) = \emptyset \vee \\ &\quad [y_i, y_i + h_i) \cap [y_j, y_j + h_j) = \emptyset, \\ &\forall i,\ 1 \le i \le n, \\ &\quad x_i + w_i \le W \ \wedge\ y_i + h_i \le H. \end{aligned} \tag{1}$$

*2.2.2 Grouped 2D Bin Packing.* The G2DBP is a bi-level optimization problem, where in the upper level, the objective is to find the optimal grouping plan of the orders, and in the lower level, the objective is to find the optimal 2DBP solution for each of the groups. The overall objective is to minimize the total number of bins. Formally, we denote the orders as $o_i = \{t_{i,1}, t_{i,2}, ..., t_{i,n_i}\}$, where $t_{i,j}$ is the $j$-th item in the $i$-th order. Given a list of orders $\mathcal{G} = \{o_1, o_2, ..., o_m\}$, the objective is to find groups $G_1, G_2, ..., G_K \subseteq \mathcal{G}$ that is solution to the following optimization problem:

$$\begin{aligned} &\text{minimize} \ \sum_{i=1}^{K} \text{BP}(\{t | o \in G_i, t \in o\}), \\ s.t.\ &G_1 \cup G_2 \cup ... \cup G_K = \mathcal{G}, \\ &\forall i, j,\ 1 \le i, j \le K,\ G_i \cap G_j = \emptyset. \end{aligned} \tag{2}$$

where $\text{BP}(\cdot)$ represents the minimum bin usage of input items obtained by solving the 2DBP problem in (1).

*2.2.3 Guillotine Constraint.* In manufacturing, a common constraint for 2DBP is that the packing plan must be able to be cut using only *guillotine* cuts [28]. As illustrated in Figure 1, the guillotine cuts go from one side all the way to the other side, orthogonal to either side of the bin. This constraint may be induced by the manufacturing equipment or the characteristic of the material itself. For example, wood materials are cut with table saws that move in one direction and cannot produce non-guillotine T-shape cutting patterns. And glass materials have to be cut from side to side to make a clean split. Nonetheless, it is allowed to use multiple stages of guillotine cuts to produce complex cutting patterns. For example, we can cut the bin horizontally and split, then cut vertically and split, etc. But to simplify the manufacturing procedure, the number of stages of cutting is usually limited to a maximum of three, with two horizontal cutting stages and one vertical cutting stage in between, as illustrated in Figure 2.

*2.2.4 Shelf Packing Algorithm.* In practice, to produce bin packing plans that satisfy the three-staged guillotine-cut constraint, the shelf packing algorithm is used to pack the items. The algorithm is based on the following observations. After the first stage of horizontal guillotine cuts, the bin is divided into multiple parts referred to as *shelves*. And after the second stage of vertical guillotine cuts, each shelf is divided into multiple *blocks*. Because there is at most one more stage of horizontal guillotine cuts, each block may contain no item (waste block), one item, or multiple items with the same width.

Given a sequence of items, the shelf algorithm packs them one by one. It starts with an empty list of bins, shelves, and blocks. When an item comes, it decides where to put the item by sequentially performing the following steps:

- Search for a block with the same width of the item and the height of the remaining space at the top is no less than the item's height. If found, put the item on top of the current items in the block.
- Search for a shelf with a height no less than the height of the item and the width of the remaining space on the right is no less than the item's width. If found, create a new block with the item and put it to the right of the current blocks in the shelf.
- Search for a bin where the height of the remaining space on the top is no less than the item's height. If found, create a new shelf with the item and put it on top of the existing shelves in the bin. Otherwise, add a new bin and put th shelf at its bottom.

# 3 METHODOLOGY

## 3.1 Overall Framework

The overall framework of our method is illustrated in Figure 3. We design an iteration-based hierarchical reinforcement learning framework to co-optimize the grouping and the bin packing plans, given all the customer orders containing the specifications for the items to be manufactured. First, we take a greedy approach to divide the orders into as few groups as possible under the group size constraint, and we use this as the initial grouping plan. Then, two reinforcement learning agents, named **GroupRL** and **PermRL**, iteratively improve the grouping plan and solve bin packing, respectively, with heuristics learned from the data. In the lower level, PermRL generates bin packing plans for each group by sequentially selecting the best next item to pack. In the upper level, GroupRL adjusts the current grouping plan by swapping the orders of two groups or taking an order from one group and relocating it to another group based on the feedback of the lower-level bin packing results and the features of the orders and groups. Finally, after multiple iterations of the improvement loop, we put the best solution found in the process into production. The items are then categorized and assembled into products according to the customer orders, ready for packaging and shipping.

## 3.2 Initial Order Grouping

As the saying goes, a good beginning is half the task. It is important to provide a good initial grouping plan as the starting point of the iteration. The objective of order grouping is to divide the given orders into smaller groups under the size limit and minimize the total number of used bins of each group. Intuitively, for larger groups, the bin utilization (the area of the items divided by the area of the bins) is higher as there are more choices in item selection in the bin packing step. Following this intuition, we design a greedy method named *Min-Group* to minimize the number of groups, or in other words, to fit as many items as possible into each group.

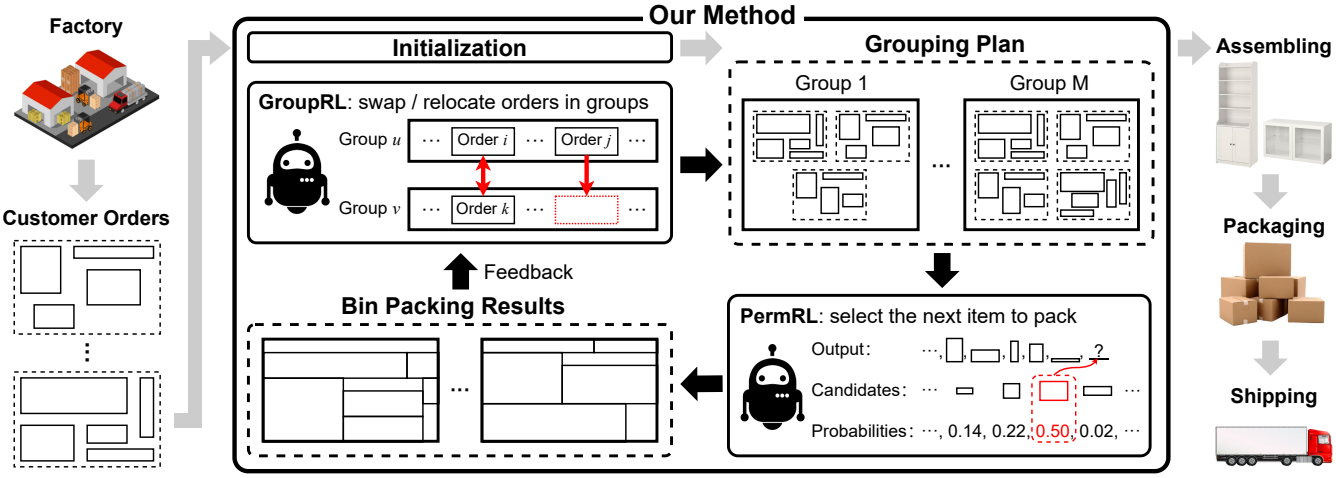


**Figure 3: The overall framework of our method in the production pipeline.**

Specifically, it first sorts the orders by the number of items in non-increasing order. And then, for each order, it tries to insert it into the first group the order fits or create a new group if the order cannot be inserted into any existing groups. The pseudo-code of Min-Group is described in Algorithm 1.

## 3.3 The Lower Level and PermRL

The task of the lower level is to find the optimal packing plan for the items of each group. As described in Section 2, due to the 3-stage guillotine cut constraint, we have to arrange the items using the *shelf* algorithm, which takes an ordered sequence of items as input and returns the 2D bin packing plan. Therefore, the task of the lower level becomes finding the best permutation of the items to minimize the number of used bins.

Traditional methods are either sorting-based ones, which sort the items according to specific properties, or searching-based ones, which iteratively search for an optimal solution. The sorting-based methods are faster than searching-based ones but also provide lower-quality solutions. To achieve a better tradeoff between speed and quality, we use reinforcement learning to learn from data the best heuristic to permute the items. In this way, we can produce higher-quality solutions than sorting-based methods with little extra time. We formulate the permutation of the items as a Markov decision process (MDP), where the items are selected one by one. And we design a reinforcement learning agent, PermRL, with an encoder-decoder structure [20, 37] to process sequential input and produce sequential output.

**Algorithm 1:** The Min-Group Algorithm.

**Input:** a list of orders $O = \{O_1, ..., O_n\}$, the maximum number of items of each group $M$
**Output:** grouped orders $\mathcal{G} = \{G_1, ..., G_m\}$ ($G_i \subseteq O$)

$O' = \{O'_1, ..., O'_n\} \leftarrow O$ sorted non-increasingly by size;
$\mathcal{G} = \{\}$;
**for** $i \leftarrow 1$ **to** $n$ **do**
  found $\leftarrow$ false;
  **for** $j \leftarrow 1$ **to** $|\mathcal{G}|$ **do**
    **if** $|O'_i| + \sum_{O \in G_j} |O| \le M$ **then**
      // Add $O'_i$ to $G_j$
      $G_j \leftarrow G_j \cup \{O'_i\}$;
      found $\leftarrow$ true;
      **break**;
    **end**
  **end**
  **if** *not* found **then**
    // Add new group $\{O'_i\}$ to $\mathcal{G}$
    $\mathcal{G} \leftarrow \mathcal{G} \cup \{\{O'_i\}\}$;
  **end**
**end**
**return** $\mathcal{G}$;

### 3.3.1 *MDP Formulation.*

- **State**: The state includes the size of the bins, the sizes of all the input items, and the indices of the items selected in the previous time step.
- **Action**: Select an item that is not selected in previous time steps.
- **Transition**: Add the selected item to the output and mark it as selected.
- **Reward**: Reward is defined as the negative bin packing cost (the number of bins used). And the reward is episodic (only given in the last step).

*3.3.2 Encoder.* The encoder encodes the input items into item embeddings. We denote the features of the input items as $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$, where each $\mathbf{x}_i \in \mathbb{R}^2$ is the height and width of the $i$-th item normalized by dividing the height and width of the bin. The initial embeddings of the items are computed as $\mathbf{h}_i^{(0)} = W^{(0)}\mathbf{x}_i + b^{(0)}$, where $W^{(0)} \in \mathbb{R}^{M \times 2}, b^{(0)} \in \mathbb{R}^M$ are trainable parameters and $M$ is the embedding dimension. Then we apply $L$ multi-head attention (MHA) layers [36] to the item embeddings to propagate information

among the items. For each layer $l = 1, 2, ..., L$, the embeddings are computed as follows,

$$\mathbf{h}_1^{(l)}, \mathbf{h}_2^{(l)}, ..., \mathbf{h}_n^{(l)} = \text{MHA}(\mathbf{h}_1^{(l-1)}, \mathbf{h}_2^{(l-1)}, ..., \mathbf{h}_n^{(l-1)}). \quad (3)$$

For simplicity of notion, the final embedding $\mathbf{h}_i^{(L)}$ of the last layer is written as $\mathbf{h}_i$ in the following.

*3.3.3 Decoder.* The decoder selects one unselected item at a time, forming a permutation of the $n$ input items after $n$ steps. At each time step $t$, the decoder computes the context vector $\mathbf{q}_t$, updates it with one step of MHA to obtain $\mathbf{q}'_t$, and computes the probability of selecting which item by using yet another MHA between $\mathbf{q}'_t$ and item embeddings $\mathbf{h}_i$.

We denote the index of the item chosen at time step $t$ as $\boldsymbol{\pi}_t$. We denote the indices of the items chosen up to time step $t$ as $\boldsymbol{\pi}_{1:t}$ and the rest as $\overline{\boldsymbol{\pi}_{1:t}}$. The context vector is computed as follows to capture the global information and history:

$$\mathbf{q}_t = [\bar{\mathbf{h}} \| \text{GRU}(\mathbf{h}_{\boldsymbol{\pi}_{1:(t-1)}})], \quad (4)$$

where $[\cdot\|\cdot]$ means concatenation, $\bar{\mathbf{h}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i$, and $\text{GRU}(\cdot)$ represents the final hidden state of the gated recurrent unit (GRU) after processing the input sequence. The context vector $\mathbf{q}_t \in \mathbb{R}^{2M}$ is split into $\mathbf{q}_t^1, ..., \mathbf{q}_t^H \in \mathbb{R}^{2m}$ for $H$ heads, where $H \cdot m = M$. The item embeddings $\mathbf{h}_i$ are split into $\mathbf{h}_i^1, ..., \mathbf{h}_i^H \in \mathbb{R}^m$. The updated context vector $\mathbf{q}'_t$ is computed as follows:

$$\tilde{a}_{t,i}^j = \begin{cases} \frac{(W_Q^j \mathbf{q}_t^j)^T (W_K^j \mathbf{h}_i^j)}{\sqrt{m}} & \text{if } i \notin \boldsymbol{\pi}_{1:(t-1)}, \\ -\infty & \text{otherwise.} \end{cases} \quad (5)$$

$$a_{t,i}^j = \frac{\exp \tilde{a}_{t,i}^j}{\sum_{k=1}^{n} \exp \tilde{a}_{t,k}^j}, \quad (6)$$

$$\mathbf{q}'_t = \sum_{j=1}^{H} W_O^j \sum_{i=1}^{n} a_{t,i}^j \mathbf{h}_i^j, \quad (7)$$

where $W_Q^j \in \mathbb{R}^{2m \times m}$, $W_K^j \in \mathbb{R}^{m \times m}$, and $W_O^j \in \mathbb{R}^{m \times 2m}$ are trainable parameters. Finally, the probability for selecting item $i$ at time step $t$ is computed as follows,

$$\tilde{p}_t(i) = \begin{cases} C \cdot \tanh \frac{(W_{Q2}^j \mathbf{q}'_t)^T (W_{K2}^j \mathbf{h}_i)}{\sqrt{m}} & \text{if } i \notin \boldsymbol{\pi}_{1:(t-1)}, \\ -\infty & \text{otherwise.} \end{cases} \quad (8)$$

$$p_t(i) = \frac{\exp \tilde{p}_t(i)}{\sum_{k=1}^{n} \exp \tilde{p}_t(i)}, \quad (9)$$

where $W_{Q2}^j \in \mathbb{R}^{2M \times M}$ and $W_{K2}^j \in \mathbb{R}^{M \times M}$ are trainable parameters, and $C$ is a clipping coefficient. The probability of outputting permutation $\boldsymbol{\pi}$ is

$$p_\theta(\boldsymbol{\pi}) = \prod_{t=1}^{n} p_t(\boldsymbol{\pi}_t), \quad (10)$$

where $\theta$ indicates all the above trainable parameters of the agent.

Following the convention of reinforcement learning, we also refer to the permutation generated by sampling from (9) at each step as a *rollout* of the agent's policy.

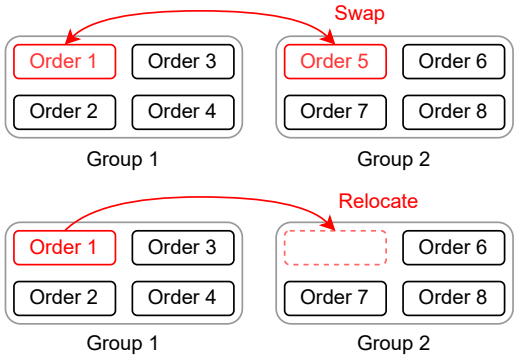


**Figure 4: The *swap* and *relocate* operators.**

*3.3.4 Training.* PermRL is trained with the POMO algorithm [22], a variant of the REINFORCE algorithm that considers multiple rollouts with different start points. In each iteration of the training loop, we randomly generate a problem instance by sampling a sequence of $N$ items from the dataset, and we encode them using the encoder. At the first step of decoding, we sample $K \leq N$ items according to (9). Then we use each of the $K$ items as the first item and run the remaining $N - 1$ decoding steps to obtain $K$ different rollouts (permutations of the input), $\boldsymbol{\pi}^1, ..., \boldsymbol{\pi}^K$. For a permutation $\boldsymbol{\pi}$, We define the reward $R(\boldsymbol{\pi})$ as the negative of the number of bins used under permutation $\boldsymbol{\pi}$. The policy gradient can be approximated as follows,

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (R(\boldsymbol{\pi}^i) - \bar{R}) \nabla_\theta \log p_\theta(\boldsymbol{\pi}^i), \quad (11)$$

where the average reward $\bar{R} = \frac{1}{N}\sum_{i=1}^{N} R(\boldsymbol{\pi}^i)$ is used as the baseline to reduce the variance. The parameter $\theta$ is updated using the Adam optimizer [19] with a learning rate of $10^{-5}$.

## 3.4 The Upper Level and GroupRL

To find the best grouping plan for the orders, we iteratively improve the current order grouping plan by making adjustments based on the feedback from the lower level. We design a reinforcement learning agent, GroupRL, to extract helpful information from the hierarchical input features and learn the best adjustment operations through training. For the adjustment, we design two *operators* as illustrated in Figure 4. The $swap(u, v, i, j)$ operator swaps the order $i$ of group $u$ with the order $j$ of the group $v$. And the $relocate(u, v, i)$ operator takes the order $i$ out of group $u$ and inserts it into group $v$. Theoretically, starting from the initial grouping plan, we can reach the optimal grouping plan in a finite number of steps with the two operators.

The MDP can be formulated as follows:

- **State**: The state includes the current grouping plan, the bin packing results (utilization of each used bin) of the groups and orders, the items of the orders, item sizes, the limit on the maximum number of items contained in a group, and historical information of previous states and actions.
- **Action**: Choose $swap(u, v, i, j)$ or $relocate(u, v, i)$ operator and determine its parameters.

- **Transition**: Apply the chosen operator to the current grouping plan.
- **Reward**: Reward is defined as the difference in total bin packing cost after the transition. The reward is positive for a decrease in cost, and negative otherwise.

In Section 3.3.4, for PermRL, we use the POMO algorithm that trains on the rollout level, i.e., sampling complete rollouts for training, because there is no reward for intermediate actions but only one reward at the end. For GroupRL, however, there are nonzero rewards for intermediate actions. Algorithms that train on transition level, i.e., sampling transitions of (*state, action, reward, next_state*) for training, are more suitable and can better utilize this fine-grained information. Therefore, we adopt PPO [35] to train GroupRL, which is one of the best-performing and most widely used transition-level algorithms. The model of GroupRL takes an actor-critic structure, where the actor outputs the probability distribution over the valid actions at each step, and the critic outputs the value of the state, which is used as the baseline in the PPO algorithm.

*3.4.1 Hierarchical State Encoding.* The information in the state is hierarchical and of variable length: there are a variable number of groups, each group contains a variable number of orders, and each order contains a variable number of items. To propagate information in the hierarchy from bottom to top, we need to find a function that maps from this variable length input into fixed-sized output. The function is also desired to be permutation-invariant as the permutation of the input is meaningless or undefined in our problem. Intuitively, the function family that meets the above requirements is composed of aggregate functions like *min*, *max*, *mean*, and their combinations. This intuition is also proved theoretically in [39]. Therefore, we use the structure of aggregate functions followed by a multi-layer perceptron (MLP) to distill information across layers in the hierarchy. Here the MLP is used to learn the combinations of aggregate functions.

Specifically, we refer to the $i$-th group as $g_i$, the $j$-th order in $g_i$ as $o_{i,j}$, and the $k$-th item in $o_{i,j}$ as $t_{i,j,k}$. Similar to that in the PermRL, we compute the embedding $\mathbf{t}_{i,j,k} \in \mathbb{R}^M$ of item $t_{i,j,k}$ as a linear projection from the normalized height and width of the item. We denote the bin packing feature of an order or a group $\alpha$ as $\mathrm{BP}(\alpha) = (a, b, c) \in \mathbb{R}^3$, where $a, b, c$ are the *min*, *max*, *mean* of bin utilization (total area of items in the bin divided by the area of the bin) of the bins in the packing result of $\alpha$. The embedding $\mathbf{o}_{i,j} \in \mathbb{R}^{3M+3}$ of order $o_{i,j}$ is computed as follows,

$$\mathbf{o}_{i,j} = [\mathrm{BP}(o_{i,j}) \| \mathrm{MIN}_k(\mathbf{t}_{i,j,k}) \| \mathrm{MAX}_k(\mathbf{t}_{i,j,k}) \| \mathrm{MEAN}_k(\mathbf{t}_{i,j,k})], \quad (12)$$

where MIN, MAX, and MEAN are element-wise operations to aggregate item information. Then, the embedding $\mathbf{g}_i \in \mathbb{R}^{3M+L+3}$ of group $g_i$ is computed as follows,

$$\tilde{\mathbf{o}}_{i,j} = W_2^O \tanh(W_1^O \mathbf{o}_{i,j} + \mathbf{b}_1^O) + \mathbf{b}_2^O, \quad (13)$$

$$\mathbf{g}_i = [\mathrm{BP}(g_i) \| \mathrm{MIN}_j(\tilde{\mathbf{o}}_{i,j}) \| \mathrm{MAX}_j(\tilde{\mathbf{o}}_{i,j}) \| \mathrm{MEAN}_j(\tilde{\mathbf{o}}_{i,j}) \| \mathbf{hist}_i], \quad (14)$$

where $W_1^O \in \mathbb{R}^{(3M+3)\times 2M}, W_2^O \in \mathbb{R}^{2M\times M}, \mathbf{b}_1^O \in \mathbb{R}^{2M}, \mathbf{b}_2^O \in \mathbb{R}^M$ are trainable parameters, $\mathbf{hist}_i \in \mathbb{R}^L$ is a 0-1-encoded history vector indicating whether group $g_i$ is selected in the previous $L$ steps, 1 for selected and 0 otherwise.

*3.4.2 Actor.* We denote the action probability as $p(u, v, i, j)$. If $j \neq -1$, it represents the probability of action $swap(u, v, i, j)$. If $j = -1$, it represents the probability of action $relocate(u, v, i)$. At each step, the actor computes the probability distribution over the actions, and chooses the action for the step by sampling or taking the argmax of the distribution. This could be done by first enumerating all valid actions that do not violate the group size constraint and then computing the probability for each action. However, this is very inefficient because of the large action space. Using the fact that $p(u, v, i, j) = p(u, v)p(i, j|u, v)$, we can break this into two steps: first sample $(u', v')$ from $p(u, v)$ and then sample $(i', j')$ from $p(i, j|u', v')$. If there are $P$ choices for $(u', v')$ and $Q$ choices for $(i', j')$, this two-step sampling procedure reduces the amount of computations from $P \times Q$ to $P + Q$.

We design two neural networks to model $p(u, v)$ and $p(i, j|u, v)$ separately. For $p(u, v)$, We compute it with the embedding of the groups:

$$\tilde{\mathbf{g}}_i = W_2^g \tanh(W_1^g \mathbf{g}_i + \mathbf{b}_1^g) + \mathbf{b}_2^g, \quad (15)$$

$$\tilde{p}(u, v) = C \tanh(W_2^p \tanh(W_1^p [\tilde{\mathbf{g}}_u \| \tilde{\mathbf{g}}_v] + \mathbf{b}_1^p) + \mathbf{b}_2^p), \quad (16)$$

$$p(u, v) = \frac{\exp \tilde{p}(u, v)}{\sum_{u', v'} \exp \tilde{p}(u', v')}, \quad (17)$$

where $W_1^g \in \mathbb{R}^{(3M+L+3)\times 2M}, W_2^g \in \mathbb{R}^{M\times 2M}, \mathbf{b}_1^g \in \mathbb{R}^{2M}, \mathbf{b}_2^g \in \mathbb{R}^M$, $W_1^p \in \mathbb{R}^{2M\times 2M}, W_2^p \in \mathbb{R}^{M\times 2M}, \mathbf{b}_1^p \in \mathbb{R}^{2M}, \mathbf{b}_2^p \in \mathbb{R}^M$ are trainable parameters, and $C$ is a clipping coefficient.

For $p(i, j|u, v)$, in order to provide a unified account for the different actions of swap and relocate, we define three scoring functions $\mathrm{S}_1(g, o)$, $\mathrm{S}_2(g, o)$, and $\mathrm{S}_3(o_1, o_2)$, which are MLPs that take group and order embeddings as input and output a real value. $\mathrm{S}_1(g, o)$ is the score of removing order $o$ from group $g$. $\mathrm{S}_2(g, o)$ is the score of inserting order $o$ into group $g$. $\mathrm{S}_3(o_1, o_2)$ is the affinity score of order $o_1$ and $o_2$. In this way, we can express the score of action $swap(u, v, i, j)$ as follows,

$$\begin{aligned} \mathrm{S}_{\mathrm{swap}}(u, v, i, j) = {} & \mathrm{S}_1(g_u, o_i) + \mathrm{S}_1(g_v, o_j) + \mathrm{S}_2(g_u, o_j) \\ & + \mathrm{S}_2(g_v, o_i) + \mathrm{S}_3(o_i, o_j). \end{aligned} \quad (18)$$

And we express the score of action $relocate(u, v, i)$ as follows,

$$\mathrm{S}_{\mathrm{relocate}}(u, v, i) = \mathrm{S}_1(g_u, o_i) + \mathrm{S}_2(g_v, o_j). \quad (19)$$

The action probability is computed as follows,

$$\tilde{p}(i, j|u, v) = \begin{cases} \mathrm{S}_{\mathrm{swap}}(u, v, i, j) & \text{if } j \neq -1, \\ \mathrm{S}_{\mathrm{relocate}}(u, v, i) & \text{if } j = -1. \end{cases} \quad (20)$$

$$p(i, j|u, v) = \frac{\exp \tilde{p}(i, j|u, v)}{\sum_{i', j'} \exp \tilde{p}(i', j'|u, v)}. \quad (21)$$

*3.4.3 Critic.* The critic estimates the state value function $\mathrm{V}(s)$. We compute it with the embeddings of the groups in state $s$:

$$\mathrm{V}(s) = \mathbf{W}_2^{cT} \tanh(W_1^c \, \mathrm{MEAN}_i(\tilde{\mathbf{g}}_i) + \mathbf{b}_1) + b_2^c, \quad (22)$$

where $W_1^c \in \mathbb{R}^{M\times M}, \mathbf{W}_2^c \in \mathbb{R}^M, \mathbf{b}_1 \in \mathbb{R}^M, b_2^c \in \mathbb{R}$ are trainable parameters.

**Table 1: Performance comparison on the Real, G200, and G100 datasets.**

| Method | Real | | | G200 | | | G100 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Bin Usage | Gap To LB | Time/s | Bin Usage | Gap To LB | Time/s | Bin Usage | Gap To LB | Time/s |
| LB | 2900 | 0.00% | - | 2780 | 0.00% | - | 741 | 0.00% | - |
| MG-W | 3844 | 32.55% | 0.01 | 4491 | 61.55% | 0.01 | 1308 | 76.52% | <0.01 |
| MG-H | 3320 | 14.48% | 0.01 | 3186 | 14.60% | 0.01 | 915 | 23.48% | <0.01 |
| MG-A | 3401 | 17.28% | 0.01 | 3437 | 23.63% | 0.01 | 1001 | 35.09% | <0.01 |
| SA-W | 3631 | 25.21% | 2.49 | 4152 | 49.35% | 2.70 | 1157 | 56.14% | 1.28 |
| SA-H | 3198 | 10.28% | 2.38 | 3088 | 11.08% | 2.69 | 854 | 15.25% | 1.24 |
| SA-A | 3275 | 12.93% | 2.78 | 3305 | 18.88% | 2.85 | 919 | 24.02% | 1.35 |
| GGA-W | 3536 | 21.93% | 3494.3 | 3979 | 43.13% | 4805.4 | 1067 | 43.99% | 2226.8 |
| GGA-H | 3172 | 9.38% | 3823.2 | 3056 | 9.93% | 4741.5 | 836 | 12.82% | 2109.7 |
| GGA-A | 3244 | 11.86% | 4141.7 | 3244 | 16.69% | 5355.9 | 890 | 20.11% | 2408.1 |
| **Ours** | **3111** | **7.28%** | **18.94** | **2999** | **7.88%** | **22.26** | **820** | **10.66%** | **11.40** |
| Improvement | 1.92% | 2.10% | - | 1.87% | 2.05% | - | 1.91% | 2.16% | - |

## 4 EXPERIMENTS

To demonstrate the effectiveness of our proposed method, we conduct extensive experiments on both the real-world dataset and synthetic datasets generated from real-world data, answering the following research questions:

- **RQ1**: How is the performance of our method compared with other baselines?
- **RQ2**: How much do the GroupRL and PermRL contribute to the overall performance?
- **RQ3**: How robust is our method against the change of input distribution and constraints?
- **RQ4**: How is deployment performance in practical applications?

### 4.1 Experiment Settings

*4.1.1 Datasets and metrics.* The real-world dataset **Real** contains two weeks of production data with a total number of 2,418 orders and 81,083 items. The limit on group size is 200. For the training of our method, we generate a synthetic dataset **G200** that mimics the distribution of the real-world data. We also generate dataset **G100** that cuts the number of items and orders from G200 in half to compare the performance of the methods when the limit on group size is reduced to 100. See Appendix A for more details.

We focus on two metrics: bin usage and gap. Bin usage is the total number of bins used for the given input. Gap is the bin usage compared to the estimated lower bound of bin usage. The lower bound (**LB**) is obtained by removing the group size constraint for the upper level as well as the guillotine cut constraint for the lower level. Specifically, we put all orders into one group and use the MaxRect algorithm for bin packing, the bin usage of which is shown to be 1.005 times the optimal value [16].

*4.1.2 Baselines.* As there are no available methods that can directly solve the bi-level G2DBP problem, we select the best methods for the upper level and lower level and combine them together. For the lower level shelf bin packing task, we include the *Width-First* (**W**), *Height-First* (**H**) and *Area-First* (**A**) heuristics [16] that select the next item with the largest width, height or area. For the upper-level order grouping task, we include three methods. The first one is *Min-Group* (**MG**), which is described in Section 3.2. The second one is simulated annealing (**SA**), which is widely used to solve various grouping problems [8, 10, 17]. The third one is grouping genetic algorithm (**GGA**)[6] with memetic search [24], which is one of the best-performing methods for the grouping problem [33]. See Appendix B and C for detailed parameter settings and system setup.

### 4.2 Overall Performance (RQ1)

The overall performance of our method and the baseline methods on the real dataset and synthetic datasets G200 and G100 is shown in Table 1. In the method column, we use notations like **MG-W** to refer to the baseline that uses *Min-Group* (**MG**) as the upper-level method and *Width-First* (**W**) as the lower method. The lower bound (**LB**, see Section 4.1.1) of bin usage is also included in the table. The improvement row is computed by comparing our method with the strongest baseline. For comparison, SA, GGA, and our method all run 1,000 searching steps on the test set. The average running time of the test cases is reported. From the experiment results, we can observe that:

- Our method achieves the best performance on across all three datasets. Specifically, for bin usage, our method beats the best baseline method GGA-H by 1.92%, 1.87%, and 1.91% on Real, G200, and G100 datasets, respectively. And Our method beats the second-best baseline SA-H by 2.72%, 2.88%, and 3.98% on Real, G200, and G100.
- Our method is time-efficient, solving the G2DBP problem of a day's production in less than half a minute. While the bin usage of the GGA-H baseline is close to our method, GGA-H takes much longer to solve the problem, making it impractical in factory manufacturing.
- Our method gives solutions closest to the lower bound. Compared with the best baseline, the gap to the lower bound is reduced by 2.10%, 2.05%, and 2.16% on Real, G200, and G100, respectively.

### 4.3 Ablation Study (RQ2)

To demonstrate the importance and contribution of GroupRL (GRL) and PermRL (PRL) to the final results, we test the performance of replacing GroupRL with MG, SA, GGA and replacing PermRL with the best heuristic Height-First (H). The result is shown in Figure 5 and we summarize the observation as follows,

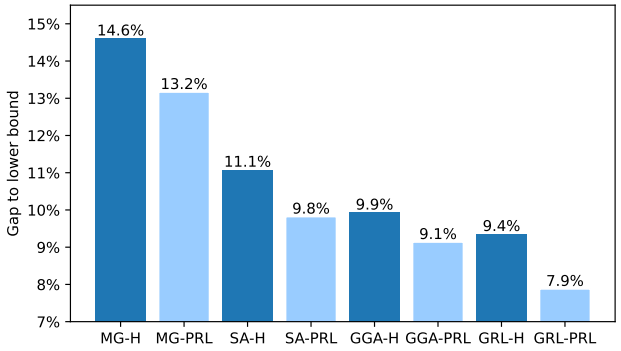


**Figure 5: Performance comparison of different combinations of upper and lower methods on G200.**

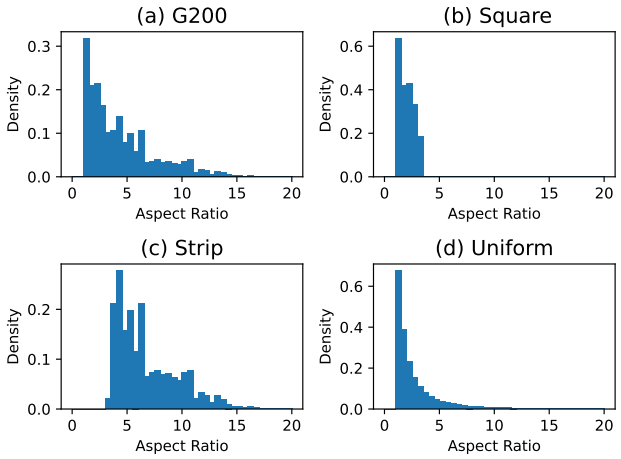


**Figure 6: The distribution of the aspect ratio of items.**

- PermRL learns better strategy than the heuristic. Comparing the results of $x$-H and $x$-PRL, we find that PermRL consistently achieves better performance no matter the upper method, which also demonstrates that the learned strategy is robust against the distribution shift incurred by the change of upper-level method.
- GroupRL can effectively utilize lower-level feedback. Comparing the results of SA-$x$, GGA-$x$, and GRL-$x$, GroupRL achieves the best performance and can find better solutions in the same number of search steps with the knowledge extracted from input features and feedback.

## 4.4 Generalization Performance (RQ3)

A common concern about learning methods is whether the heuristics or strategies learned on the training dataset can generalize to other datasets of different distributions and characteristics. This is also of practical importance as the distribution of orders and constraints may change over time in factory production. If the method has poor generalizability, it needs to be retrained frequently on new data, which may cause high maintenance costs and delay factory production.

To test the generalization performance of our method, we consider two kinds of scenarios. The first scenario is the change in the distribution of the shape of items. In the dataset G200, the items are sampled from the empirical distribution of the real-world data. Figure 6(a) shows the distribution of the aspect ratio (width/height) of the items in the dataset. We can see that it is a long-tail distribution with a mix of square-like items and strip-like items. The median of the distribution is 3.44. We divide the distribution in half at the median to obtain two distributions for more square-like items and more strip-like items, respectively. By sampling from these two distributions, we generate dataset **Square** and dataset **Strip** that have the same number of orders and items as G200 and only differ in the distribution of item shapes. Besides, we also generate dataset **Uniform**, of which the width and height of the items are sampled from the uniform distribution. The item shape distributions of the above datasets are shown in Figure 6.

The second scenario is the change of group size limit, which is also an important parameter in the G2DBP problem. We test our method trained with a group size limit of 200 on test cases with a group size limit of 100 and 300. For limit 100, we use dataset G100. And for limit 300, we use dataset **G300**, of which the data is the same as G200 but the group size limit is changed to 300.

We compare our method (trained on G200) with the baselines on the above 5 test cases. The results are shown in Figure 8. We can see that even though our method is not specifically trained for any of those datasets, it still consistently achieves better performance than any of the baseline methods, which demonstrates that our method is robust against the unpredictable change in input distribution and problem constraints that may occur in real-life production.

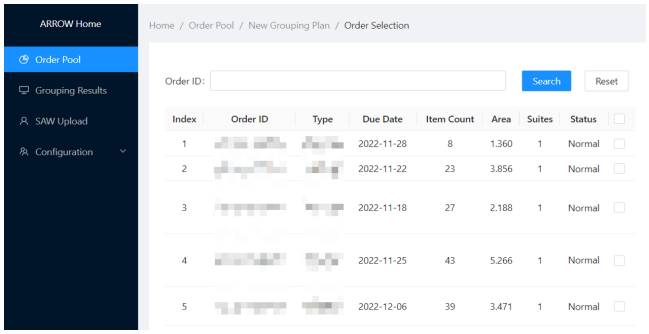


**Figure 7: The interface of our deployed system.**

## 4.5 Deployment Performance (RQ4)

We deployed our method in the factory of ARROW Home, one of the leading brands in customized furniture manufacturing. In the past, the orders of the day were grouped manually by the workers according to their experience. For example, they may try to put as many orders as possible in a group without exceeding the group size limit and put orders with similar shapes of items into the same group. This process is both tedious and inefficient. And on days with tight schedules, it gets even worse. Now with our method integrated in the production system, as shown in Figure 7, the workers

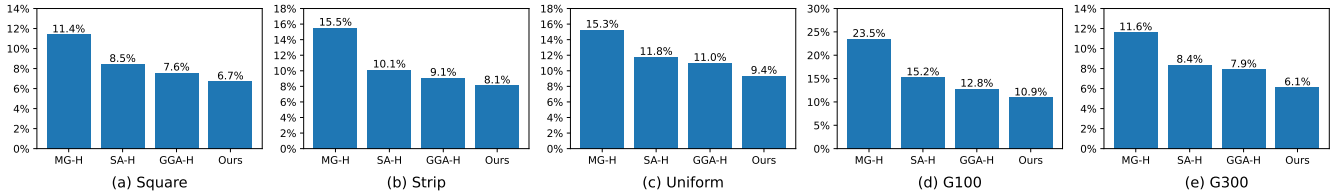


**Figure 8: Performance comparison on five transfer test cases in terms of the gap to lower bound.**

can create new grouping plan by simply selecting the orders to be manufactured from the order pool. Then our method will process the order information and output the optimized production plan with details in seconds. After confirmation, the orders are manufactured according to the plan. In two weeks of test production, the average utilization of each bin is improved from 70% to 73%, reducing the total cost of raw materials by 4.1%, which will save 1.2 million dollars in a year for the total cost of 30 million dollars.

## 5 RELATED WORK

We discuss the related works on two aspects: bin packing problem and grouping problem.

**Bin packing problem** (BP) is one of the most widely studied problems in the area of operation research, as many mathematical or real-world problems can be formulated as BPPs. Generally, there are two kinds of objects in a BP: bins and items. The objective of the BP is to pack all the items into the bins while conforming to certain constraints. The simplest form of BP is the one-dimensional BP (1DBP), where the items can be thought of as balls with different weights, the bins are containers with a fixed maximum capacity, and the constraint is that the total weight of the balls in the bin cannot exceed the capacity of the bin. Naturally, BP can be extended to 2D or 3D where the items and bins are 2D or 3D objects, and the constraint is that the items must fit inside the bin without overlap.

For 2DBP, several methods have been proposed over the years. The non-learning-based methods can be categorized into exact and heuristic methods. The exact methods usually either solve BP though enumeration of packing patterns [13, 27] or turn BP into a mixed integer programming (MIP) problem and solve the MIP [25, 31]. As even 1DBP, the simplest form of BP, is NP-hard [9], current exact methods can only be applied to small-scale problems. In cases where the problem scale is large or speed is preferred over finding the exact optimum, heuristic methods are more often used. The bottom-left heuristic [3] pack the items one by one into the lowest and leftmost position. The MaxRect algorithm [16] maintains a list of maximal rectangles in the remaining space and sequentially packs items into the best one. These heuristics can be further combined with tabu search [38], simulated annealing [23], genetic algorithm [12], and other meta-heuristics to improve performance. However, in our problem setting, the simple sorting heuristics cannot provide high-quality solutions. At the same time, more sophisticated methods do not meet the high computational efficiency requirement, as the lower level is queried frequently by the upper level in bi-level optimization. To achieve a better tradeoff between quality and speed, we design a reinforcement learning model for learning from data how to sort items based on their features. As for learning-based methods, the existing works mainly focus on solving online 3DBP [14, 15, 30, 40] or 2DBP without the guillotine constraint [7, 21], which cannot be applied in our problem.

**Grouping problem** refers to a class of combinatorial optimization problems where a finite set $\mathcal{G}$ is partitioned into mutually disjoint subsets $G_1, G_2, ..., G_K$ (called *groups*). For different applications, there are different objective functions and constraints on the groups. For example, the 1DBP can be thought of as a grouping problem where $\mathcal{G}$ is the set of items and the groups $G_i$ are the bins. The objective is to minimize the number of groups, and the constraint is that the sum of the items in a group may not exceed a limit. Similar to the bin packing problem, for most settings of objective functions and constraints, the corresponding grouping problem is NP-hard, and heuristic methods are studied more often. The heuristic methods include variable neighborhood search (VNS) [1, 5, 29], simulated annealing [11, 18], tabu search [32, 34], grouping genetic algorithm (GGA) [4, 6] and particle swarm optimization (PSO) [2, 26]. According to the experiments of the survey [33], GGA is the best-performing method among them. Different from previous works, the grouping problem in our setting is a bi-level problem where the objective function of the grouping is implicitly defined by the lower level optimization problem and is highly non-linear, challenging for existing methods to optimize. In our work, we design a reinforcement learning model to better utilize the feedback from the lower level and extract knowledge from input features to effectively search for the optimal solution.

## 6 CONCLUSIONS

In this paper, we formulated the grouped 2D bin packing problem that arises from the batch manufacturing constraint in the manufacturing industry, which is a bi-level problem and the upper-level optimization of order grouping is coupled with the lower-level 2D bin packing of the items in the orders. We solve the problem in a data-driven way and propose an iteration-based hierarchical reinforcement learning framework with two reinforcement learning models, GroupRL and PermRL, that learn the best strategies for the optimization problems from the data. We conduct extensive experiments on both real and synthetic datasets to demonstrate the effectiveness of our method in terms of optimization results, computational efficiency, and the ability for trained models to generalize to unseen datasets. We also deploy our method in the production system of ARROW Home, and the online results show that our method improves the average bin utilization from 70% to 73% and reduces the cost of raw materials by 4.1%, which means a saving of 1.2 million dollars a year.

# REFERENCES


[1] Yassine Adouani, Bassem Jarboui, and Malek Masmoudi. 2018. A Variable Neighborhood Search with Integer Programming for the Zero-One Multiple-Choice Knapsack Problem with Setup. In *International Conference on Variable Neighborhood Search*.

[2] Emel Kizilkaya Aydogan, Yılmaz Delice, Ugur Özcan, Cevriye Gencer, and Özkan Bali. 2019. Balancing stochastic U-lines using particle swarm optimization. *Journal of Intelligent Manufacturing* 30 (2019), 97–111.

[3] Brenda S. Baker, Edward G. Coffman, and Ronald L. Rivest. 1980. Orthogonal Packings in Two Dimensions. *SIAM J. Comput.* 9 (1980), 846–855.

[4] Jose Alejandro Cano. 2019. Parameters for a Genetic Algorithm: An Application for the Order Batching Problem. *IBIMA Business Review* (2019).

[5] Ivan A. Davydov and Yury A. Kochetov. 2015. VNS-based heuristic with an exponential neighborhood for the server load balancing problem. *Electron. Notes Discret. Math.* 47 (2015), 53–60.

[6] Emanuel Falkenauer. 1994. A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems. *Evolutionary Computation* 2 (1994), 123–144.

[7] Jie Fang, Yunqing Rao, Xusheng Zhao, and Bing Du. 2023. A Hybrid Reinforcement Learning Algorithm for 2D Irregular Packing Problems. *Mathematics* (2023).

[8] Kamyla Maria Ferreira and Thiago Alves de Queiroz. 2018. Two effective simulated annealing algorithms for the Location-Routing Problem. *Appl. Soft Comput.* 70 (2018), 389–422.

[9] M. R. Garey and David S. Johnson. 1978. Computers and Intractability: A Guide to the Theory of NP-Completeness.

[10] Fernando Garza-Santisteban, Roberto Sánchez-Pámanes, Luis Antonio Puente Rodríguez, Iván Amaya, José carlos Ortíz-Bayliss, Santiago Enrique Conant-Pablos, and Hugo Terashima-Marín. 2019. A Simulated Annealing Hyper-heuristic for Job Shop Scheduling Problems. *2019 IEEE Congress on Evolutionary Computation (CEC)* (2019), 57–64.

[11] Fernando Garza-Santisteban, Roberto Sánchez-Pámanes, Luis Antonio Puente Rodríguez, Iván Amaya, José carlos Ortíz-Bayliss, Santiago Enrique Conant-Pablos, and Hugo Terashima-Marín. 2019. A Simulated Annealing Hyper-heuristic for Job Shop Scheduling Problems. *2019 IEEE Congress on Evolutionary Computation (CEC)* (2019), 57–64.

[12] José Fernando Gonçalves. 2007. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *Eur. J. Oper. Res.* 183 (2007), 1212–1229.

[13] Eleni Hadjiconstantinou and Nicos Christofides. 1995. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research* 83 (1995), 39–56.

[14] Jie Jia, Huiliang Shang, and Xiong Chen. 2022. Robot Online 3D Bin Packing Strategy Based on Deep Reinforcement Learning and 3D Vision. *2022 IEEE International Conference on Networking, Sensing and Control (ICNSC)* (2022), 1–6.

[15] Yuan Jiang, Zhiguang Cao, and Jie Zhang. 2021. Learning to Solve 3-D Bin Packing Problem via Deep Reinforcement Learning and Constraint Programming. *IEEE transactions on cybernetics* PP (2021).

[16] Jukka Jylänki. 2010. A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing. *retrived from http://clb. demon. fi/files/RectangleBinPack. pdf* (2010).

[17] R. Kamalakannan, R. Sudhakara Pandian, and Pl. Sivakumar. 2019. A simulated annealing for the cell formation problem with ratio level data. *International Journal of Enterprise Network Management* (2019).

[18] T. Kampke. 1988. Simulated annealing: Use of a new tool in bin packing. *Annals of Operations Research* 16 (1988), 327–332.

[19] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[20] Wouter Kool, Herke van Hoof, and Max Welling. 2018. Attention, Learn to Solve Routing Problems!. In *International Conference on Learning Representations*.

[21] Olyvia Kundu, Samrat Dutta, and S. Kumar. 2019. Deep-Pack: A Vision-Based 2D Online Bin Packing Algorithm with Deep Reinforcement Learning. *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)* (2019), 1–7.

[22] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Seungjai Min, and Youngjune Gwon. 2020. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. *ArXiv* abs/2010.16011 (2020).

[23] T. W. Leung, Chi Kin Chan, and Marvin D. Troutt. 2003. Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. *Eur. J. Oper. Res.* 145 (2003), 530–542.

[24] Shengcai Liu, Ke Tang, and Xin Yao. 2020. Memetic Search for Vehicle Routing with Simultaneous Pickup-Delivery and Time Windows. *ArXiv* abs/2011.06331 (2020).

[25] Andrea Lodi, Silvano Martello, and Daniele Vigo. 2004. Models and Bounds for Two-Dimensional Level Packing Problems. *Journal of Combinatorial Optimization* 8 (2004), 363–379.

[26] Vahid Mahmoodian, Armin Jabbarzadeh, Hassan Rezazadeh, and Farnaz Barzinpour. 2019. A novel intelligent particle swarm optimization algorithm for solving cell formation problem. *Neural Computing and Applications* 31 (2019), 801–815.

[27] Silvano Martello and Daniele Vigo. 1998. Exact Solution of the Two-Dimensional Finite Bon Packing Problem. *Management Science* 44 (1998), 388–399.

[28] Óscar Oliveira, Dorabela Gamboa, and Elsa Silva. 2022. An introduction to the two-dimensional rectangular cutting and packing problem. *International Transactions in Operational Research* (2022).

[29] Zeping Pei, Zhuan Wang, and Yiwen Yang. 2019. Research of Order Batching Variable Neighborhood Search Algorithm based on Saving Mileage. *Proceedings of the 3rd International Conference on Mechatronics Engineering and Information Technology (ICMEIT 2019)* (2019).

[30] Aaron Valero Puche and Sukhan Lee. 2022. Online 3D Bin Packing Reinforcement Learning Solution with Buffer. *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2022), 8902–8909.

[31] Jakob Puchinger and Günther R. Raidl. 2007. Models and algorithms for three-stage two-dimensional bin packing. *Eur. J. Oper. Res.* 183 (2007), 1304–1327.

[32] Jin Qin, Xianhao Xu, Qinghua Wu, and T. C. Edwin Cheng. 2016. Hybridization of tabu search with feasible and infeasible local searches for the quadratic multiple knapsack problem. *Comput. Oper. Res.* 66 (2016), 199–214.

[33] Octavio Ramos-Figueroa, Marcela Quiroz-Castellanos, Efrén Mezura-Montes, and Oliver Schütze. 2020. Metaheuristics to solve grouping problems: A review and a case study. *Swarm Evol. Comput.* 53 (2020), 100643.

[34] Daniel Schermer, Mahdi Moeini, and Oliver Wendt. 2019. A hybrid VNS/Tabu search algorithm for solving the vehicle routing problem with drones and en route operations. *Comput. Oper. Res.* 109 (2019), 134–158.

[35] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *ArXiv* abs/1707.06347 (2017).

[36] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. *ArXiv* abs/1706.03762 (2017).

[37] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *NIPS*.

[38] Lijun Wei, Andrew Lim, and Wenbin Zhu. 2011. A Skyline-Based Heuristic for the 2D Rectangular Strip Packing Problem. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*.

[39] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alex Smola. 2017. Deep Sets. *ArXiv* abs/1703.06114 (2017).

[40] Hang Zhao, Qijin She, Chenyang Zhu, Y. Yang, and Kai Xu. 2020. Online 3D Bin Packing with Constrained Deep Reinforcement Learning. *ArXiv* abs/2006.14978 (2020).

# A SYNTHETIC DATASETS

The dataset **Real** contains two weeks of production data. From the statistics, we find that the number of orders of each day is between 100 and 300 and roughly follows the Gaussian distribution $\mathcal{N}(200, 40)$. Therefore, we generate the synthetic dataset by sampling the number of orders of each day from Gaussian distribution $\mathcal{N}(200, 40)$ and discard samples less than 100 or greater than 300. The items in each order are sampled from the empirical distribution of all the items in the real-world dataset. The synthetic dataset **G200** contains 1,000 days for training and 10 days for testing. **G100** is obtained from **G200** by cutting the number of items and orders of each day in half.

# B BASELINES

The methods we use for the upper-level order grouping task are as follows.

- **MG** It is described in Section 3.2 and used for the initialization in our method. It tries to minimize the bin usage by minimizing the number of groups.
- **SA** It is widely used to solve various grouping problems. We use a linear cooling schedule. At each step $t$, the temperature $T = 1 - \frac{t}{N}$ where $N$ is the total number of steps. It generates state transition proposal $s'$ by randomly applying the swap and relocate operators on current state $s_t$. The probability of accepting this transition is $p = \min(1, \exp((\text{cost}(s_t) - \text{cost}(s'))/T))$. If accepted, $s_{t+1} \leftarrow s'$; otherwise $s_{t+1} \leftarrow s_t$.
- **GGA**[6] It is one of the best-performing methods for the grouping problem. In the genetic algorithm setting, we start with a population of 16 randomly generated solutions (order grouping plans) and perform $N$ steps of evolution. At each step, we generate the child population by randomly selecting parents to perform crossover operation. The groups in the parents are recombined, removing repeated orders and reinserting missing orders to their best group (in the sense of minimum cost increase) in a greedy manner. We also use the memetic search technique to *educate* the child population through local search, which reduces the cost of the child by repeatedly finding and applying swap or relocate operator that leads to a lower cost.

# C SYSTEM SETUP AND REPRODUCIBILITY

The experiments are conducted on a Linux server with an Intel(R) Xeon(R) Gold 6342 CPU @ 2.80GHz, NVIDIA GeForce RTX 3090 GPU, and 128GB of device memory. We train the GroupRL model for 10,000 epochs (~2 days) and train the PermRL model for 5,000 epochs (~4 hours). We implement all the baselines and our method with Python3 and Pytorch. We have also released the code, trained models, and datasets to facilitate reproduction and future research: https://github.com/tsinghua-fib-lab/G2DBP.

For the parameters of the PermRL and GroupRL models, we use grid searches to determine the best settings for the final models. In PermRL, we choose the embedding dimension $M = 32$ from {16, 32, 64}, choose the number of the MHA layers $L = 3$ from {0, 1, 2, 3}, choose the number of attention heads $H = 4$ from 2,4,8. The clipping coefficient $C$ in (8) and (16) is set as 10. In GroupRL, we set the embedding dimension $M = 32$, the length of action history $L = 10$. The scoring functions $S_1$, $S_2$, and $S_3$ are 4-layered MLPs with the hidden dimensions set to be $\{32, 32\}$.