Computers & Industrial Engineering 188 (2024) 109814

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Solving a 3D bin packing problem with stacking constraints☆

Weiwang Zhu, Sheng Chen, Min Dai, Jiping Tao *

School of Aerospace Engineering, Xiamen University, Xiamen, Fujian, China
Key Laboratory of Big Data Intelligent Analysis and Decision-making of Fujian Province, Xiamen University, Xiamen, Fujian, China

ARTICLE INFO

Keywords:
Three-dimensional bin packing problem
Stacking constraints
Branch-and-price
Encoder–decoder

ABSTRACT

We consider a novel variant of the three-dimensional bin packing problem (3DBPP) arising from a maritime shipping port. Unlike the classical bin packing problem, our 3DBPP requires all items to be stacked vertically during the packing process. To tackle this problem effectively, we divide it into two subproblems: the stack packing problem (SPP) and the two-dimensional bin packing problem (2DBPP). We first formulate the SPP as an integer programming model to minimize the total bottom area of the stacks, which is further solved by a branch-and-price method. Then, in the 2DBPP, to pack all stacks into a minimum number of bins, we follow the left-down most extreme point placement strategy. An encoder–decoder model was developed to optimize the packing sequence and orientation of the stacks simultaneously. Industrial instances from shipping ports were collected and extended. The computational results show that our two-stage algorithm performs well in solving the stacking-constrained 3DBPP.

1. Introduction

The classical three-dimensional bin packing problem (3DBPP) involves the orthogonal packing of a set of cuboid-shaped items into a minimum number of three-dimensional bins (Faroe et al., 2003). This problem is widely recognized as strongly NP-hard, and has been applied to various logistics and transportation systems, including container shipping, air cargo loading, and rail transport. The 3DBPP in these scenarios usually incorporates industry-related constraints and customized objectives, making it challenging to solve using existing exact and heuristic algorithms.

In this study, the 3DBPP with stacking constraints was solved. The problem originates from maritime shipping scenarios. The items are weakly heterogeneous and categorized by their length d_i, width w_i, and height h_i. The bins were ISO containers with dimensions $D \times W \times H$. Our goal is to pack all items of m categories into a minimum number of 3D containers with stacking constraints. Specific constraints are described in the following section.

As depicted in Fig. 1, the items packed into the containers must be stacked in a constrained manner to ensure stability and satisfy the packing restrictions. Specifically, each item must be fully supported by another item or the bin itself. Furthermore, rotation of the x- and y-axes is not allowed because the items must be kept upright and cannot be placed horizontally. In addition, the cumulative height of the items within the same stack must not exceed the height of the bins. These constraints are concretized from the operations and requirements of real ports. By organizing items into stacks, we address the support and rotation constraints while minimizing the risk of item damage and improving the efficiency of automation and handling. Similar packing restrictions have been explored in previous studies such as Haessler and Talbot (1990), Bettinelli et al. (2008), and Shengming et al. (2021).

As shown in Fig. 2, we divided the constrained 3DBPP into two subproblems and solved them separately. We first define and solve a stack packing problem (SPP) to build vertical stacks that satisfy all the constraints. We then solved the classical two-dimensional bin packing problem (2DBPP) to pack all stacks into a minimum of three-dimensional containers. The remainder of this paper is organized as follows. In the next section, we provide a brief review of related studies. Section 3 presents our SPP formulation and its branch-and-price algorithm. In Section 4, we introduce our novel encoder–decoder model to solve the 2DBPP. In Section 5, the detailed computational simulations of industrial instances are presented. Finally, the conclusions are presented in Section 6.

2. Related work

The 3DBPP has been extensively studied over the past few decades. Exact methods, heuristics, and learning algorithms are the main research methods for 3DBPP. Exact algorithms can solve the established

☆ This research is supported by the Natural Science Foundation of Fujian province, China [Grant No. 2020J01053].
* Corresponding author at: School of Aerospace Engineering, Xiamen University, Xiamen, Fujian, China.
E-mail addresses: zhuweiwang@stu.xmu.edu.cn (W. Zhu), taojiping@xmu.edu.cn (J. Tao).

https://doi.org/10.1016/j.cie.2023.109814
Received 13 January 2023; Received in revised form 15 September 2023; Accepted 2 December 2023
Available online 4 December 2023

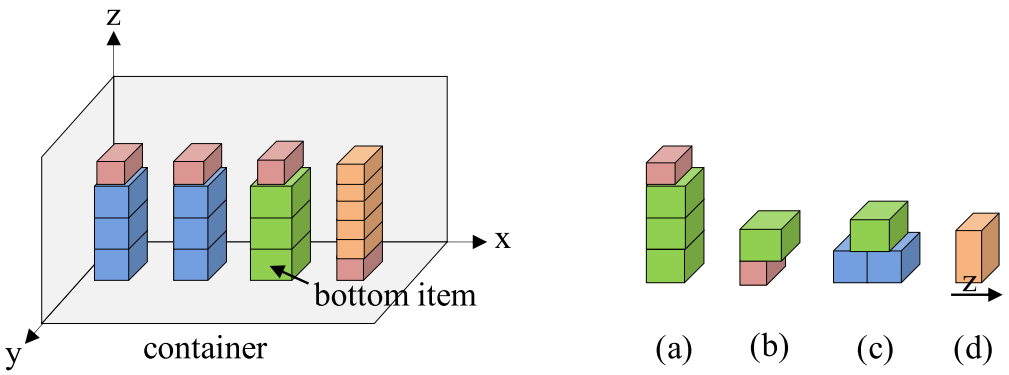

Fig. 1. Items inside containers must be stacked. (a) valid stack; (b) nonvalid stack, the upper item is not fully supported; (c) nonvalid stack, the upper item can only be supported by exactly one item or bin; (d) nonvalid stack, the z-axis of the item must be kept upright.

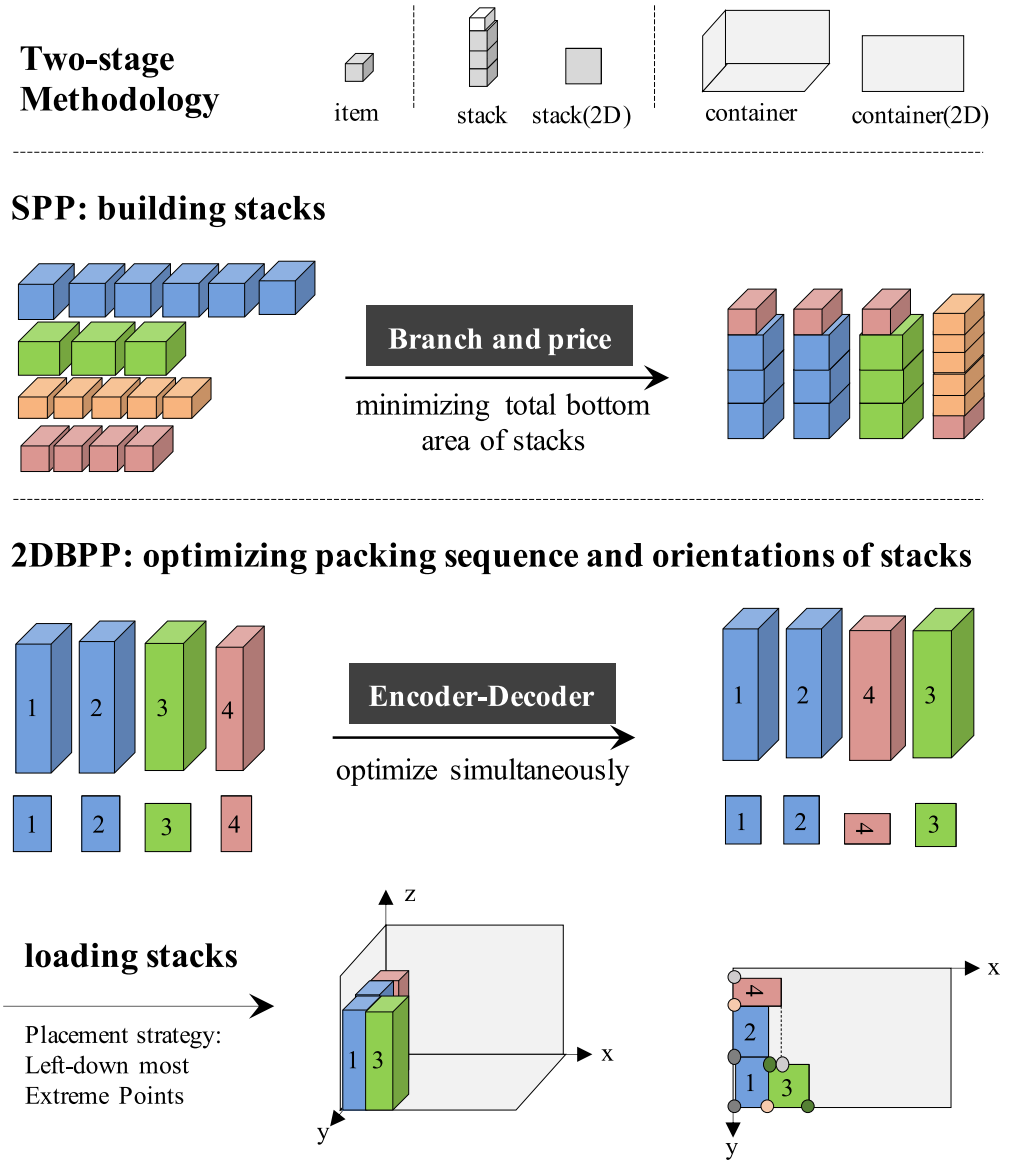

Fig. 2. Overview of the two-stage algorithm. In stage one, we build stacks by solving SPP. In stage two, we optimize the packing sequence and orientations of the built stacks and load them into containers.

mathematical models and obtain optimal solutions. However, when the size of the instance increased, the solution time of the algorithm increased exponentially. Chen et al. (1995) proposed an accurate 0–1 mixed integer programming model for 3DBPP, and Hifi et al. (2010) enhanced its solution efficiency by introducing effective inequalities. Martello et al. (2007) employed constraint programming to

address the 3DBPP with guillotine-cutting constraints. Mahvash et al. (2018) designed a column generation-based heuristic for a 3DBPP with rotation and used a heuristic method to solve the pricing subproblem in the column generation structure.

Heuristics were developed to obtain suboptimal solutions within an acceptable solution time. Simple construction heuristics such as first-fit decreasing (Johnson, 1973) add one item at a time until all items are packed. Complex metaheuristics iteratively search for better solutions. A guided local search algorithm was proposed in Faroe et al. (2003). In Crainic et al. (2009) and Liu et al. (2011), tabu search algorithms were developed. Hopper and Turton (1999), Gonçalves and Resende (2013), and Kucukyilmaz and Kiziloz (2018) studied genetic evolution.

Researchers have recently become interested in the use of learning methods to solve bin packing problems. Hu et al. (2017) modified the pointer network to solve the 3DBPP, which is an encoder–decoder model developed by Vinyals et al. (2015) for solving the traveling salesman problem (TSP). Subsequently, Duan et al. (2018) constructed a multitasking framework based on selected learning that can optimize the packing sequence and rotation of items. Jiang et al. (2021)designed a multimodal encoder to simultaneously determine the packing sequence, rotation, and placement position. Balaji et al. (2019) and Zhang et al. (2021) converted a packing problem into a Markov decision process that was trained using deep reinforcement learning.

A 3DBPP in the real world usually incorporates customized constraints and objectives (Côté et al., 2014, Yaakoubi et al., 2020, Gomez & Terashima-Marín, 2018, and Duan et al., 2022). The most common constraints are rotation, support, and guillotine-cutting. Other constraints, including weight limits, positioning, and operational limitations, are well defined in Bortfeldt and Wäscher (2013). Decomposing complicated bin packing problems by building layers (George & Robinson, 1980, Bortfeldt & Gehring, 2001, and Lodi et al., 2002), stacks (Liu et al., 2014), strips, and blocks (Fanslau & Bortfeldt, 2010, Wei et al., 2014) are common methods for dealing with complex constraints. The simplest method is to construct these structures using empirical sorting rules and specific placement strategies. Subsequently, Liu et al. (2014) used dynamic programming to build better vertical strips, whereas Bettinelli et al. (2008) developed a column generation method to generate optimal levels in a 2D level strip packing problem.

3. Stack packing problem

3.1. Problem description

The first stage involves building constrained stacks. We defined and solved two SPPs to build high-quality stacks. SPP-1 can be described as follows: Given a set $M = \{1, 2, \dots, m\}$ of m types of items, the objective is to pack all items into the minimum number of stacks. Each item is characterized by its dimensions, represented as $depth(d_i) \times width(w_i) \times height(h_i)$, and the number of items of type i is denoted as b_i. The bins available for packing were identical, with dimensions $D \times W \times H$. SPP-2 introduced a different objective of minimizing the total bottom area of the stacks.

By formulating and solving these two stack packing problems, we aim to derive optimal or near-optimal packing solutions to construct constrained stacks. These stacks, constructed with careful consideration of the given objectives, lay the foundation for the subsequent stages of our algorithm in tackling the 2DBPP.

3.2. Problem formulation

We formulate the SPP using the following integer programming model. Without a loss of generality, we assume that all item depths are not smaller than their widths and that all input dimensions are positive integers.

$$\min_{x} \sum_{j \in J} c_j x_j \tag{1}$$

$$\sum_{j \in J} t_{ji} x_j = b_i \qquad \forall i \in m \tag{2}$$

$$x_j \in N \qquad \forall j \in J \tag{3}$$

We define set J as a given set of stack patterns, in which stack pattern j corresponds to a feasible item set that can be built into a stack. Each stack pattern is characterized by the number of each item type in pattern j, denoted as $[t_{j1}, t_{j2}, \dots, t_{jm}]$. c_j is the cost of the stack pattern j, which differs between SPP-1 and SPP-2. In SPP-2, the cost of a stack is defined by the bottom area of the stack and c_j is the maximum $d_i \times w_i$ for all items i in pattern j.

$$c_j = \begin{cases} 1 & minimizing\ the\ number\ of\ stacks \\ bottom\ area(j) & minimizing\ the\ total\ bottom\ area \end{cases} \tag{4}$$

The decision variables x_j are nonnegative integers that represent the number of patterns j used. The stack quality is considered for which we propose two greedy objective strategies: minimizing the number of stacks and minimizing the total bottom area of the stacks, corresponding to SPP-1 and SPP-2, respectively. Constraint (2) ensures that all item types are fully packed.

Because the J set is very large, we tended to solve this integer programming model by column generation (CG). By removing the integrality constraints on x variables and relaxing them, we obtain the following master problem:

$$\min_{x} \sum_{j \in J} c_j x_j \tag{5}$$

$$\sum_{j \in J} t_{ji} x_j \geq b_i \qquad \forall i \in m \tag{6}$$

The employed CG technique consists of three main components: the master problem (MP), restricted master problem (RMP), and pricing subproblem. The CG process begins by initializing the RMP. In this study, it is populated with a small set of columns obtained from the solution of a first-fit decreasing heuristic. Subsequently, new columns are added incrementally to the RMP by solving a knapsack-like subproblem. In each iteration, the RMP is solved, and the dual values associated with each item type are passed to the pricing subproblem. The pricing subproblem is then solved to identify the column with the most negative reduced cost, which is subsequently added to the RMP. This iterative procedure continues until no negative cost columns are found, indicating the termination of the process.

3.3. Pricing subproblem

The subproblem is dedicated to finding a new stack pattern p with a negative reduced cost. To solve this problem, we developed the following integer programming model:

$$\min_{\{t_{pi}, z_i\}} c_p - \sum_{i \in m} p_i t_{pi} \tag{7}$$

$$\sum_{i \in m} h_i t_{pi} \leq H \tag{8}$$

$$t_{pi} \leq b_i z_i \qquad \forall\ i \in m \tag{9}$$

$$z_i + z_j \leq 1 \qquad \forall\ i, j \in m, q_{ij} = 1 \tag{10}$$

$$z_i \in \{0, 1\} \qquad \forall i \in m \tag{11}$$

$$t_{pi} \in N \qquad \forall i \in m \tag{12}$$

h_i and p_i are the height and the dual values of item type i, respectively. q_{ij} indicates whether types i and j can be packed into the same stack under a vertical stacking constraint. $q_{ij} = 1$ indicates that types i and j cannot be placed on a single stack.

$$q_{ij} = \begin{cases} 0 & if\ \ (d_i - d_j)(w_i - w_j) \geq 0 \\ 1 & otherwise \end{cases} \tag{13}$$

The binary variables z_i were introduced to indicate whether type i was selected. If pattern p contains item type i, then $z_i = 1$, otherwise $z_i = 0$. The variable t_{pi} represents the number of types i in pattern p. The objective is to minimize the reduced cost of pattern p, which is calculated by the cost of c_p subtracting the sum of the dual item values from the cost of c_p. To minimize the number of stacks, the cost of pattern p is simply set to one. When the objective is set to minimize the total bottom area of the stacks, c_p corresponds to the maximum $d_i \times w_i$ for all item types i in pattern p. Constraint (8) is the knapsack constraint on the item height with the capacity as the bin height H. Constraint (9) limits the value range of variables t_{pi}. Constraint (10) addresses the stacking constraint: when $q_{ij} = 1$, which means that types i and j cannot be placed in the same stack, then $z_i + z_j$ is limited to 1, indicating that only one of the two types can be selected.

When solving the pricing subproblem of SPP-2, an additional constraint is incorporated into the formulation to determine the value of c_p, which represents the maximum bottom area among all selected items. We introduce c_p as a variable and include the following equation in our model:

$$c_p \geq d_i w_i z_i \qquad \forall\, i \in m \tag{14}$$

The subproblem of SPP-1 in our formulations can be classified as a disjunctively constrained knapsack problem (DCKP) (Bettinelli et al., 2017). Moreover, the problem becomes more complex when the objective is set to minimize the total bottom area of the stacks because the pattern cost c_p is variable. Solving this problem optimally at each iteration would be computationally expensive because of NP-hardness. However, it is not necessary to determine the optimal pattern for each generation. To address this issue, we propose the heuristic algorithm presented in Algorithm 1.

In this algorithm, the subproblem of SPP-2 is decomposed into m sub-knapsack problems with fixed costs corresponding to each item type. Each sub-knapsack problem fixes its c_p value by pre-packing an item of type i as the bottom item. The pre-packed knapsack problems were solved using standard dynamic programming techniques. The capacity of each knapsack is reduced to $H - h_i$, and the items considered for packing in each knapsack are those with a smaller bottom area than the pre-packed item. The value of each item was set to its dual value. It is important to note that, while solving knapsack problems, we disregard disjunctive constraints to simplify the process. By examining the sorted list of partial item sets returned by the dynamic programming algorithm, we can identify the best solution that satisfies these constraints.

We construct m classical knapsack problems and solve each using dynamic programming, resulting in m candidate solutions. Finally, among all candidates, we select the solution pattern p with the minimum negative reduced cost.

In general, if the solution to the RMP is not an integer, we can obtain only a fractional solution to the MP. To obtain an integer solution, we apply a branch-and-price architecture. At every node of the branch-and-bound framework, we solve a new CG to search for integer solutions and use the fractional solution of the CG as the lower bound for pruning. The branching strategy chooses the variable closest to a decimal value of 0.5 as the branching variable. Each branch was then assigned an additional constraint corresponding to the selected branching variable. To illustrate this strategy, we consider an example in which we select the variable x_2 with a fractional value of 4.6 to branch. In the left branch, we add the constraint $x_2 \leq 4$, while in the right branch, we add the constraint $x_2 \geq 5$. This branching strategy enables the creation of new child nodes, each representing a branch added to the node list for exploration during the search process. Branching and pricing will be time-consuming if we dedicate ourselves to finding an optimal solution. In this study, we recorded the best integer solution found in the computing process and terminated when a pre-set time limit was reached.

Algorithm 1 Heuristic Algorithm for Solving Pricing Problem

Require: P: list of dual values
Require: k: the size of the solution pool
Output: a sub-optimal solution

```
Sort the item: c_1 ≥ c_2 ≥ … ≥ c_n
for each item type i do
    Set type i as the bottom item, and fix cost C ← c_i
    Generate item set IS = { j | c_j ≤ C, 0 ≤ j ≤ n}
    Solve the knapsack by standard DP with set IS, value P, and capacity H − h_i
    for each solution pattern s in the sorted solution pool of size k do
        if pattern s has negative cost c_s − Σ_{j∈m} p_j t_sj and meets stacking constraints then
            Add s to candidate set CS
Choose pattern p with the minimum reduced cost from CS
return p
```

4. 2D bin packing problem

The second stage involves loading all the stacks into 3D containers, which can be considered a classical 2D bin packing problem (2DBPP). At this stage, the input items for the 2DBPP are the stacks built in the first stage, represented by the bottom item. When solving a 2DBPP, there are typically three decision classes: the packing sequence of the stacks, the orientation of the stacks, and the position at which each stack is placed within the container. In this study, we trained an encoder–decoder model to learn the packing sequence and stack orientations simultaneously. The positions of the stacks were directly determined using a fixed strategy based on the placement of the most extreme points on the left.

4.1. Placement strategy

The performance of the 2DBPP algorithm strongly depends on the physical position and placement rules. Several studies have been conducted on the following position points: corner points (Faroe et al., 2003), extreme points (Crainic et al., 2008), and centroid points (Gu & Hao, 2018). Placement rules, including space-filling schemes (George & Robinson, 1980), bottom-left-fill heuristics (Burke et al., 2006), and distance measurements (Gonçalves & Resende, 2013), are typically handcrafted. The extreme points in 2D were generalized from the corner points. An item has two base points: the left-top and right-down points. The extreme points are the projections of these base points, as shown in Fig. 3.

To enhance the performance of the solutions, the placement strategies mentioned above often involve adjusting the given packing sequence and orientation. However, in this study, we aim to explore additional possibilities for our model by employing a fixed placement strategy without adjustments. This implies that the stacks were packed precisely according to the provided packing sequence and orientations, adhering to the left-down most extreme point placement strategy.

4.2. Encoder–decoder

A transformer network was employed to learn the sequence and orientation of the 2DBPP. The network was originally proposed by Bresson and Laurent (2021) to solve the TSP, where the input consists of city points and the output is the predicted sequence in which the points are visited. In this study, we utilized the TSP transformer to predict the packing sequence and orientation of the stacks.

As illustrated in Fig. 4, the stack is the input to the network. Each stack can be represented in two different orientations: $d \times w$ and $w \times d$, denoted by the binary values 0 and 1, respectively. The network output

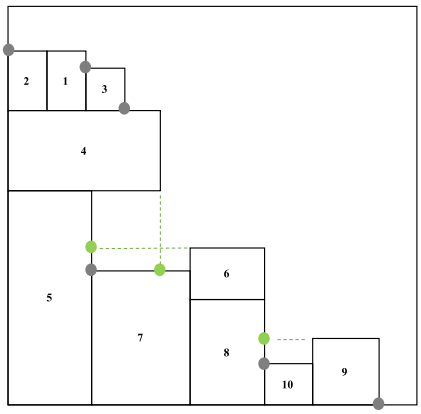

Fig. 3. 2D extreme points. All gray points are base points of items (Faroe et al., 2003). The green points are the extreme points generated by projecting the base points 4, 6, and 9 along the axis.

is a sequence indicating the order in which the stacks should be packed, along with their corresponding orientations.

The network comprises an encoder and a decoder. The encoder is a standard transformer encoder with multi-head attention (MHA) and residual connections. The MHA layer embeds the stack information and selects the most promising start stack by computing the attention scores. The decoder operates in an autoregressive manner and comprises multiple attention layers. The initial attention layer is the MHA, which encodes the selected stacks with the currently selecting stack. This step enriches the current stack using information from partial solutions. The subsequent attention layer is an encoder–decoder masked MHA that encodes the current stack (decoder) using the unselected stacks (encoder). This enables the current stack to obtain information from the remaining stacks and make an informed decision on the next most favorable stack. Unselected stacks were obtained by masking the next stack and its orientation at each step. Finally, a masked self-attention layer is employed to generate a distribution over the non-selected stacks. This distribution enables greedy sampling to select the next stack node. A more detailed description of these layers can be found in Bresson and Laurent (2021).

In this study, a reinforcement learning methodology was used to train the neural network. The output sequence is evaluated using simulated packing. By packing the stack using the sequence and placement strategy, we can obtain the exact number of bins required for packing. To increase the difference, the least packed bin is calculated by space efficiency, whereas the other bin takes the value of 1, as follows:

$$f = bin\_numbers \ - 1 \ + \frac{total\_item\_volume_{last\_bin}}{D * W * H} \tag{15}$$

5. Experiments

We tested our two-stage algorithm on weakly heterogeneous industrial instances; the bins were ISO containers with standard dimensions of 587 × 233 × 220. All the experiments were performed using a workstation with an Intel i7-7700 running at 2.4 GHz, 16 GB of memory, and Microsoft Windows 10 OS. The algorithms were coded in Python 3.6, and the mathematical solver used was Gurobi 9.0.2.

5.1. Datasets

The original data were collected from a local shipment company. It includes 20 types of different sizes, as detailed in Table 1.

We expanded the datasets by generating new instance sets for different m and n. For each m and n, we set $m = [3,\ 5,\ 8,\ 10,\ 12,\ 15,\ 20]$ and $n = [50,\ 100,\ 200]$, and built a corresponding random instance set, $RIS(m,\ n)$. As shown in Table 2, each $RIS(m,\ n)$ consists of 100 instances randomly generated according to the probability of the quantity proportion. The average performance of $RIS(m,\ n)$ was calculated and is presented in the following experiments. RIS instances are available at https://github.com/xmuMODA/3DBPP.

Table 1
Industrial data.

Type	$d_i \times w_i \times h_i$	Quantity	Type	$d_i \times w_i \times h_i$	Quantity
1	80 × 60 × 65	97	11	114 × 98 × 74	159
2	114 × 79 × 72	96	12	120 × 80 × 39	134
3	114 × 79 × 53	134	13	120 × 80 × 31	210
4	80 × 60 × 53	97	14	114 × 79 × 114	336
5	80 × 60 × 72	250	15	168 × 114 × 111	124
6	120 × 80 × 94	142	16	120 × 80 × 74	181
7	168 × 114 × 76	107	17	114 × 79 × 85	174
8	126 × 108 × 64	83	18	120 × 100 × 108	217
9	120 × 111 × 115	145	19	114 × 79 × 91	83
10	120 × 100 × 105	299	20	80 × 60 × 33	99

Table 2
Instances of $RIS(3,\ 50)$.

Instance	Types and amounts
instance 1	type 1, num 12; type 9, num 16; type 13, num 22
instance 2	type 7, num 17; type 8, num 14; type 11, num 19
	...
instance 100	type 10, num 30; type 12, num 12; type 19, num 8

Table 3
Comparison between two objective strategies on $RIS(3 \sim 20,\ 50)$.

m	LB num	*LB area	*Minimize number*			*Minimize bottom area*		
			Num	Area	f value	Num	Area	f value
3	22.01	235 230	**22.32**	240 023	1.1055	22.52	**237722**	**1.0980**
5	21.50	234 074	**22.15**	241 382	1.2289	22.26	**238411**	**1.2236**
8	19.77	221 178	**21.11**	233 692	1.3448	21.20	**227811**	**1.3292**
10	19.71	222 124	**20.94**	235 201	1.4323	21.26	**230444**	**1.3995**
12	19.49	220 628	**20.98**	234 619	1.5424	21.01	**230040**	**1.4955**
15	18.93	215 770	**20.33**	228 203	1.6094	20.4	**225642**	**1.5922**
20	18.43	207 116	**19.83**	218 041	1.6570	19.91	**216925**	**1.6355**

5.2. Experiments on the stack packing problem

In this stage, experiments are divided into four parts:

- Comparison of two objective strategies: minimizing the number of stacks and minimizing the total bottom area of the stacks.
- Comparison of solving the pricing subproblem with Gurobi and the heuristic algorithm.
- Comparisons between our branch-and-price algorithm with general heuristics.
- Experiments on different sizes of instances.

5.2.1. Comparison of two objective strategies

First, we compare the two greedy objective strategies in the SPP formulation. The pricing subproblems in the branch-and-price methods were solved using the Gurobi solver. We tested the B&P with two objectives on the $RIS(3 \sim 20, 50)$ with a time limit of one minute for each instance in the set.

For each RIS, the number and total bottom area of the stacks were recorded. To evaluate the performance of the stacks, we packed the stacks into bins with the same 2DBPP heuristics and calculated their f values.

The experimental results in Table 3 show that minimizing the total bottom area of the stacks outperforms minimizing the number of stacks in all cases, indicating a better interconnection between the total bottom area of the stacks and overall packing performance. In the follow-up experiments, the B&P algorithm was developed to minimize the total bottom area of the stacks.

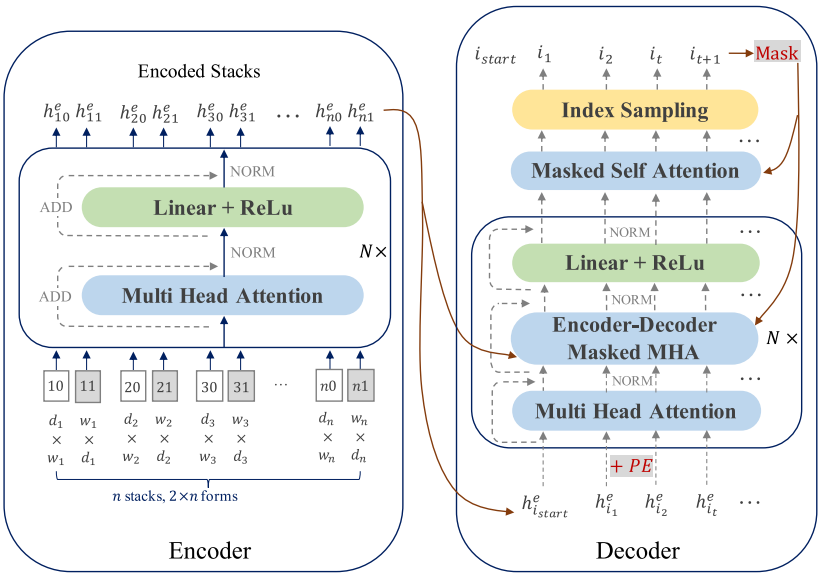

Fig. 4. Transformer for solving 2DBPP.

Table 4
Stacking efficiencies of the subproblem solution strategies on $RIS(3 \sim 20,\ 50)$.

Subproblem solver	3	5	8	10	12	15	20
IP (time limit=1)	0.8306	0.8502	0.8622	0.8659	0.8662	0.8759	0.8814
IP (time limit=5)	0.8306	0.8502	0.8622	0.8662	0.8671	0.8759	0.8814
HA (time limit=1)	0.8327	0.8512	0.8684	0.8795	0.8736	0.8787	0.8871
HA (time limit=5)	**0.8327**	**0.8521**	**0.8696**	**0.8808**	**0.8763**	**0.8830**	**0.8898**

Table 5
Number of instances improved in $RIS(3 \sim 20,\ 50)$.

Sub solver	3	5	8	10	12	15	20
IP	24	48	45	29	24	18	6
HA	**32**	**59**	**74**	**80**	**70**	**50**	**31**

5.2.2. Experiments on two subproblem solution methods

In this section, we compare two methods for solving the subproblem: solving the formulated integer programming (IP) model with a mathematical solver, and solving the subproblem with the proposed heuristic algorithm (HA).

We test the B&P incorporating the two solution methods on the $RIS(3 \sim 20,\ 50)$. For each RIS, we record the average stacking efficiency calculated using the following equation:

$$stacking \quad efficiency = \frac{\sum_{i=1}^{n} d_i \times w_i \times h_i}{total\ bottom\ area \times H} \tag{16}$$

We also recorded the number of instances in the RIS on which our B&P method could find a better solution than the RMP heuristic in one minute. Furthermore, a longer time limit of five minutes is imposed to explore the potential of the algorithm.

Tables 4 and 5 report the computational results of the two strategies for solving the subproblems. From the two tables, we can see that our B&P algorithm performs better when the subproblems are solved using the heuristic method, which exhibits a larger improvement when the time limit is enlarged to five minutes. An integer programming model was used to calculate the linear lower bounds of the master problem. Solving this problem optimally at every branch node will lead to slow convergence, which will also add a large number of columns to the RMP, causing an increase in computational complexity. In the following experiments, our B&P algorithm solved the subproblem using a heuristic algorithm.

Table 6
Stacking efficiencies of FFD, DP, and B&P on $RIS(3 \sim 20, 50)$.

Algorithm	3	5	8	10	12	15	20
Upper bound	0.8483	0.8672	0.8899	0.9042	0.9046	0.9141	0.9252
FFD	0.8034	0.8036	0.8292	0.8480	0.8518	0.8657	0.8781
DP	0.8109	0.8204	0.8355	0.8593	0.8569	0.8723	0.8849
B&P	**0.8327**	**0.8512**	**0.8684**	**0.8795**	**0.8736**	**0.8787**	**0.8871**

5.2.3. Comparison between our B&P with general heuristics

To the best of our knowledge, no algorithm exists for solving the stack packing problem. We adopt the following general heuristics for comparison:

- FFD: Sort the items decreasingly with their bottom areas, then pack them into stacks by first-fit decreasing strategy.
- DP: Sort the items that decrease with their bottom areas. Build a new stack and prepack the first item in the sorted list as the bottom item of the new stack. Dynamic programming was used to select other items to maximize the stack height. This procedure was repeated until all items were packed into stacks.

Table 6 reports the computation results of B&P and other algorithms on $RIS(3 \sim 20,\ 50)$. All data in the table denote the average stacking efficiency of all instances in the RIS. The proposed B&P outperforms the other in all cases. Compared to FFD, B&P improves performance by 2.93%, 4.76%, 3.92%, 3.15%, 2.18%, 1.30%, and 0.90% on $RIS(3 \sim 20,\ 50)$, respectively. The average gap between B&P and the upper bound is 2.60%.

When m is small, the upper bound is rather low because pattern set J is small and there are few efficient stack patterns. As m increases, the number of patterns increases exponentially, and all algorithms are likely to be more efficient. The largest gap between the B&P with FFD and DP occurs when m is in the range of $5 \sim 12$, indicating an efficiency and time balance within the given time limit. Fig. 5 shows the visualization of FFD, DP, and B&P on an instance of $RIS(8,\ 50)$, which provides a better solution with fewer stacks and a smaller bottom area.

5.2.4. Experiments on different sizes of instances

Finally, we tested the B&P algorithm with $n = 100$ and $n = 200$. The computational results in Tables 7 and 8 show that there is little difference when n increases. Our SPP model was formulated for item

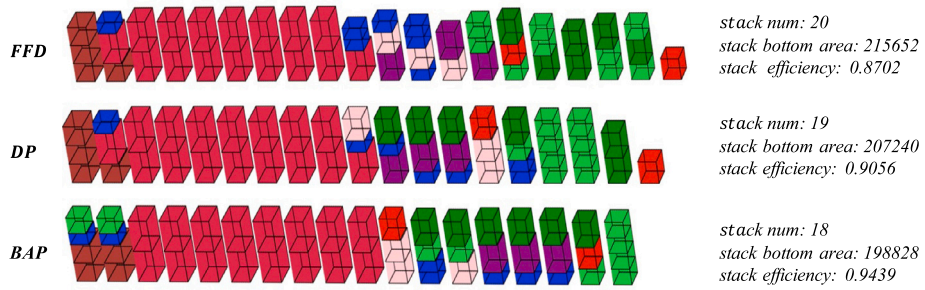

Fig. 5. Visualization of FFD, DP, and B&P on an instance of $RIS(8,\ 50)$.

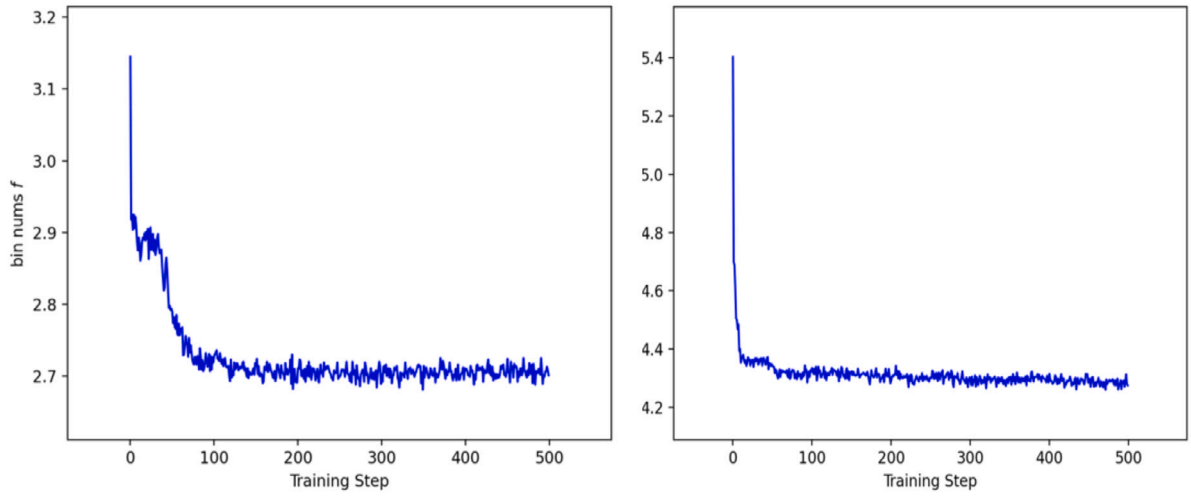

Fig. 6. Learning curves of our model for 20 and 50 stack nodes.

Table 7
Stacking efficiencies of FFD, DP, and B&P on $RIS(3 \sim 20, 100)$.

Algorithm	3	5	8	10	12	15	20
Upper bound	0.8368	0.8692	0.8879	0.9019	0.9195	0.9246	0.9352
FFD	0.8017	0.8187	0.8349	0.8492	0.8721	0.8863	0.8981
DP	0.8076	0.8350	0.8438	0.8582	0.8785	0.8934	**0.9052**
B&P	**0.8262**	**0.8648**	**0.8748**	**0.8840**	**0.8958**	**0.8993**	0.9025

Table 8
Stacking efficiencies of FFD, DP, and B&P for $RIS(3 \sim 20, 200)$.

Algorithm	3	5	8	10	12	15	20
Upper bound	0.8406	0.8685	0.8945	0.9068	0.9194	0.9278	0.9410
FFD	0.7903	0.8197	0.8486	0.8653	0.8772	0.8939	0.9135
DP	0.8126	0.8290	0.8579	0.8746	0.8799	0.8976	**0.9214**
B&P	**0.8401**	**0.8665**	**0.8845**	**0.8929**	**0.9005**	**0.9079**	0.9190

type m, and an increase in n did not impose additional complexity on the computation. Our B&P algorithm performed best in the range of $m = 5 \sim 12$, corresponding to weakly heterogeneous items. When m is increased to 20, B&P may fail to find better solutions than DP under a time limit of one minute.

In summary, we developed a branch-and-price architecture to address the stack packing problem with the objective of minimizing the number of stacks and the total bottom area of the stacks. To improve the computational efficiency, we propose a heuristic algorithm to solve the pricing subproblem within the B&P framework. Our B&P algorithm outperformed the FFD and DP approaches in most cases, particularly for instances with sizes ranging from $m = 5$ to $m = 12$. Furthermore, our algorithm demonstrated its capability to handle larger instances that exhibit weak heterogeneity. To compute the average performance, a time limit of one minute per instance was set for the proposed algorithm. However, by increasing the time limit, optimal solutions can be obtained for specific scenarios.

5.3. Experiments on 2DBPP

We proposed an encoder–decoder model to train the packing sequence and orientations of the stacks directly. To test the performance of our model, we compared it to several other sequence and orientation strategies. All algorithms were adapted to use the same placement strategy, that is, the left-down most extreme point strategy. It is worth noting that the EP heuristic itself has demonstrated effectiveness in achieving favorable packing results, even in the absence of explicit optimization of the packing sequence and orientation.

- Stochastic + EP: Stochastically sorting and rotating the stacks, place the stacks under extreme points strategy.
- BRKGA-1 + EP: Biased random key genetic algorithm by Gonçalves and Resende (2013). The algorithm was adapted for evolving only the packing sequence and orientation. The population size and number of generations were both set to 100.
- BRKGA-2 + EP: Population size = 500, number of generations = 500.

Our model was trained using 20, 50, and 100 stacks. The training sets were built by collecting stacks during the first stage. Table 9 presents the computational results of our model and other strategies. The results showed the average performance of a hundred testing instances of $RIS-20, 50, 100$. BRKGA is a state-of-the-art algorithm for the 2DBPP; however, it requires time for evolution. Our model provides a good solution within a short time. The f value was better than that of the fast BRKGA-1, and the gap with the time-consuming BRKGA-2

Table 9
Results of our model and other strategies on $RIS-20,50,100$.

Optimizing strategies	*RIS-20*		*RIS-50*		*RIS-100*	
	f	time/s	f	Time/s	f	Time/s
Stochastic + EP	3.142	0.008	5.128	0.016	11.816	0.039
BRKGA-1 + EP	2.707	114.587	4.391	280.527	10.022	700.874
BRKGA-2 + EP	2.677	1020.620	4.234	2526.0	9.764	6308.160
BPP Transformer + EP	2.706	0.016	4.248	0.039	9.838	0.074

was 1.08%, 0.33%, and 0.75% for $RIS-20,50,100$. Fig. 6 shows the learning curves of the proposed model.

6. Conclusion

We presented a two-stage method for addressing a novel industrial 3DBPP that incorporates stacking constraints. Our algorithm divides the problem into two subproblems, each tackled in a separate stage. In the first stage, we introduced a branch-and-price algorithm specifically designed to solve stack packing problems. The experimental results reveal that minimizing the total bottom area of the stacks is a more appropriate objective for the SPP. We also developed an integer programming model for the pricing subproblem and proposed a heuristic algorithm that significantly accelerates the branch-and-price algorithm. The stack defined in this study can also be generalized to deal with other complex constraints, such as weight limits and item disjunctions. In the second stage, we proposed an encoder–decoder model capable of predicting the near-optimal packing sequence and orientations of the stacks. We expanded our datasets and conducted extensive comparisons with general heuristics. The computational results demonstrate that the proposed model consistently generates near-optimal packing solutions within a second of processing time.

CRediT authorship contribution statement

Weiwang Zhu: Methodology, Coding, Experimental validation, Manuscript. **Sheng Chen:** Experimental validation, Reviewing, Language editing. **Min Dai:** Revising, Proofreading. **Jiping Tao:** Conceptualization, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Balaji, B., Bell-Masterson, J., Bilgin, E., Damianou, A., Garcia, P. M., Jain, A., Luo, R., Maggiar, A., Narayanaswamy, B., & Ye, C. (2019). Orl: Reinforcement learning benchmarks for online stochastic optimization problems. arXiv preprint arXiv:1911.10641.

Bettinelli, A., Cacchiani, V., & Malaguti, E. (2017). A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS Journal on Computing*, *29*(3), 457–473.

Bettinelli, A., Ceselli, A., & Righini, G. (2008). A branch-and-price algorithm for the two-dimensional level strip packing problem. *4OR*, *6*(4), 361–374.

Bortfeldt, A., & Gehring, H. (2001). A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, *131*(1), 143–161.

Bortfeldt, A., & Wäscher, G. (2013). Constraints in container loading–a state-of-the-art review. *European Journal of Operational Research*, *229*(1), 1–20.

Bresson, X., & Laurent, T. (2021). The transformer network for the traveling salesman problem. arXiv preprint arXiv:2103.03012.

Burke, E., Hellier, R., Kendall, G., & Whitwell, G. (2006). A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. *Operations Research*, *54*(3), 587–601.

Chen, C., Lee, S.-M., & Shen, Q. (1995). An analytical model for the container loading problem. *European Journal of operational research*, *80*(1), 68–76.

Côté, J.-F., Gendreau, M., & Potvin, J.-Y. (2014). An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. *Operations Research*, *62*(5), 1126–1141.

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

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

Duan, L., Hu, H., Qian, Y., Gong, Y., Zhang, X., Xu, Y., & Wei, J. (2018). A multi-task selected learning approach for solving 3D flexible bin packing problem. arXiv preprint arXiv:1804.06896.

Duan, J., Tong, X., Ni, F., He, Z., Chen, L., & Yuan, M. (2022). A data-driven column generation algorithm for bin packing problem in manufacturing industry. arXiv preprint arXiv:2202.12466.

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

Faroe, O., Pisinger, D., & Zachariasen, M. (2003). Guided local search for the three-dimensional bin-packing problem. *Informs Journal on Computing*, *15*(3), 267–283.

George, J. A., & Robinson, D. F. (1980). A heuristic for packing boxes into a container. *Computers & Operations Research*, *7*(3), 147–156.

Gomez, J. C., & Terashima-Marín, H. (2018). Evolutionary hyper-heuristics for tackling bi-objective 2d bin packing problems. *Genetic Programming and Evolvable Machines*, *19*(1), 151–181.

Gonçalves, J. F., & Resende, M. G. (2013). A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics*, *145*(2), 500–510.

Gu, S., & Hao, T. (2018). A pointer network based deep learning algorithm for 0–1 knapsack problem. In *2018 Tenth international conference on advanced computational intelligence* (pp. 473–477). IEEE.

Haessler, R. W., & Talbot, F. B. (1990). Load planning for shipments of low density products. *European Journal of Operational Research*, *44*(2), 289–299.

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

Hopper, E., & Turton, B. (1999). A genetic algorithm for a 2D industrial packing problem. *Computers & Industrial Engineering*, *37*(1–2), 375–378.

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

Jiang, Y., Cao, Z., & Zhang, J. (2021). Learning to solve 3-D bin packing problem via deep reinforcement learning and constraint programming. *IEEE Transactions on Cybernetics*.

Johnson, D. S. (1973). *Near-optimal bin packing algorithms* (Ph.D. thesis), Massachusetts Institute of Technology.

Kucukyilmaz, T., & Kiziloz, H. E. (2018). Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. *Computers & Industrial Engineering*, *125*, 157–170.

Liu, S., Tan, W., Xu, Z., & Liu, X. (2014). A tree search algorithm for the container loading problem. *Computers & Industrial Engineering*, *75*, 20–30.

Liu, J., Yue, Y., Dong, Z., Maple, C., & Keech, M. (2011). A novel hybrid tabu search approach to container loading. *Computers & Operations Research*, *38*(4), 797–807.

Lodi, A., Martello, S., & Vigo, D. (2002). Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, *141*(2), 410–420.

Mahvash, B., Awasthi, A., & Chauhan, S. (2018). A column generation-based heuristic for the three-dimensional bin packing problem with rotation. *Journal of the Operational Research Society*, *69*(1), 78–90.

Martello, S., Pisinger, D., Vigo, D., Boef, E. D., & Korst, J. (2007). Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software*, *33*(1), 7–es.

Shengming, C., Wei, Z., Zhaobin, D., & Liwen, W. (2021). *A tree search loadind algorithm for airport checked baggage*. IET.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In *Advances in neural information processing systems. Vol. 28*.

Wei, L., Tian, T., Zhu, W., & Lim, A. (2014). A block-based layer building approach for the 2D guillotine strip packing problem. *European Journal of Operational Research*, *239*(1), 58–69.

Yaakoubi, Y., Soumis, F., & Lacoste-Julien, S. (2020). Machine learning in airline crew pairing to construct initial clusters for dynamic constraint aggregation. *EURO Journal on Transportation and Logistics*, *9*(4), Article 100020.

Zhang, J., Zi, B., & Ge, X. (2021). Attend2Pack: Bin packing through deep reinforcement learning with attention. arXiv preprint arXiv:2107.04333.