mathematics

Article

Space Splitting and Merging Technique for Online 3-D Bin Packing

Thanh-Hung Nguyen * and Xuan-Thuan Nguyen

School of Mechanical Engineering, Hanoi University of Science and Technology, No. 1, Dai Co Viet Road, Hanoi 112400, Vietnam; thuan.nguyenxuan@hust.edu.vn
* Correspondence: hung.nguyenthanh@hust.edu.vn; Tel.: +84-936-220-198

Abstract: This paper introduces a novel method for online 3-D bin packing, which is a strongly NP-hard problem, based on a space splitting and merging technique. In this scenario, the incoming box is unknown and must be immediately packed. The problem has many applications in industries that use manipulators to automate the packing process. The main idea of the approach is to divide the bin into spaces. These spaces are then categorized into one of two types of data structures: main and secondary data structures. Each node in the main data structure holds the information of a space that can be used to fit a new box. Each node in the secondary data structure holds the information of a space that cannot be used to place a box. The search algorithm based on these two data structures reduces the required search effort and simplifies the organizing and editing of the data structure. The experimental results demonstrate that the proposed method can achieve a packed volume ratio of up to 83% in the case of multiple bins being used. The position of a placed box can be found within milliseconds.

Keywords: 3-D bin packing; online bin packing; heuristic algorithm; space splitting and merging

MSC: 90-08

Citation: Nguyen, T.-H.; Nguyen, X.-T. Space Splitting and Merging Technique for Online 3-D Bin Packing. *Mathematics* **2023**, *11*, 1912. https://doi.org/10.3390/math11081912

Academic Editors: Shiv Raj Singh, Dharmendra Yadav and Himani Dem

Received: 23 March 2023
Revised: 13 April 2023
Accepted: 16 April 2023
Published: 18 April 2023

1. Introduction

In an automated warehouse system, one of the most critical tasks is to efficiently place products into containers. The product placement must be optimal to reduce the number of containers used. A bin packing algorithm is typically used to solve this task by arranging items of different sizes into containers to achieve the minimum number of containers used. This problem is known as strongly NP-hard [1], indicating it is a challenging problem to solve.

The logistics and e-commerce industries are rapidly growing and require efficient and reliable packing solutions. Although the possibility of fully autonomous packing manipulators is on the horizon, robotic packing has not received the necessary attention. Packing algorithms can assist robot packers in finding optimal methods to fill containers and boxes, and it is essential to research this area to improve the efficiency of these industries. This paper introduces an online 3-D bin packing algorithm that can enhance the accuracy and speed of the packing process and address the rising demand for automated logistics and e-commerce.

Bin packing can be classified as one-dimensional, two-dimensional, or three-dimensional. One-dimensional bin packing [2–4] divides a set of values such as mass, length, etc., into subgroups such that the number of subgroups is minimal and the total value in a subgroup does not exceed the predefined values. Two-dimensional bin packing [5–8] arranges items on the two-dimensional regions so that the number of regions is minimal. Three-dimensional bin packing packs all the boxes into the smallest number of bins.

Traditional three-dimensional bin packing assumes that the information of all items is known in advance and the items can be placed into the bin in an arbitrary order. This bin packing problem is called offline bin packing. Three-dimensional offline bin packing can be classified into two groups: heuristic- and learning-based methods. The heuristic-based methods solve the 3-D bin packing problem by dividing the bin into layers or slices [9–11], selecting the box with maximum bottom area [12], applying the hybrid genetic algorithm [13–15] and search framework [16,17], or using a mathematical model [18]. These methods are easy to implement and can be applied to many different types of boxes and bins with very high efficiency. However, they have difficulty handling complex practical constraints such as optimization of box-to-bin space or packing stability. In contrast, the learning-based methods can better meet the various complicated constraints than the heuristic-based methods [19]. These methods apply deep reinforcement learning to optimize the sequence and orientations of items to be packed into the bin [20–23]. However, the learning-based methods do not easily converge [24] and need to be retrained when applied to new data.

In many real-world applications, the collection of items to be packed is unknown, and the current item must be put into the bin immediately before the upcoming item appears. These applications are referred to as 3-D online bin packing. Karabulut and Inceoglu [25] proposed the deepest-bottom-left (DBL) with fill method for regular 3-D strip packing. Ha et al. [26] employed the DBL order of empty spaces to place the incoming item. Zhao et al. [24] proposed an effective constrained deep reinforcement learning strategy to solve the 3-D online bin packing problems that are defined as a constrained Markov decision process. Verma et al. [27] applied a search heuristic and deep reinforcement learning to create PackMan, which is a two-step approach. Zhao et al. [19] developed a packing configuration tree and employed a deep reinforcement learning approach to enhance the practical applicability of online 3-D bin packing.

The objective of this study was to develop an innovative online 3-D bin packing approach for robot packing applications, building upon our prior conference proceedings paper [28]. The proposed approach involves dividing the bin into spaces, which are then categorized as either main or secondary data structures. The main data structures are composed of nodes that contain information about spaces suitable for accommodating new boxes, while the secondary data structures hold information about spaces that cannot be used for placing boxes. By leveraging a search algorithm based on these data structures, the approach significantly reduces search effort and streamlines data structure organization and editing.

In this paper, we provide further details on how to locate fitted spaces for incoming boxes, achieve main and secondary spaces, and split and merge spaces for 3-D online bin packing tasks. We also introduce two additional steps to optimize empty spaces and minimize spatial fragmentation. Specifically, one step involves merging two consecutive spaces if no space in the main data structure can accommodate the incoming box, while the other step involves reorganizing the main data structure immediately after splitting the fitted space.

The previous method was thoroughly evaluated and tested using small and large bins with our own dataset. In this study, we evaluated and benchmarked the proposed method using the existing dataset by Zhao et al. [24] to compare it with other methods and verify its strengths and weaknesses. Our experiments demonstrated that our proposed method delivers high utilization and low time consumption, making it a competitive option for packing boxes.

The paper is organized as follows: Section 2 presents the proposed approach for online 3-D bin packing problem. Section 3 describes the experimental results and analysis. Finally, the conclusions are summarized in Section 4.

2. Online 3-D Bin Packing Algorithm

In order to determine the position of a box in the bin, the Cartesian coordinate system is attached to the bin at the bottom-left-front corner. The x, y, and z axes correspond to the length L, width W, and height H of the bin, respectively. During the packing time, the container is divided into many spaces. The position of each space in the coordinate system is $C_K(x_K, y_K, z_K)$. The dimensions of the space in the x, y- and z directions are the length L_K, width W_K, and height H_K of the space, respectively. Each space K is represented by a set of 7 parameters including x_K, y_K, z_K, L_K, W_K, H_K, and S_K, where $S_K = L_K \times W_K$, and $H_K = H - z_K$. Depending on the L_K, W_K, and S_K parameters, the space can be classified into main or secondary data structures. Figure 1 represents the coordinate system attached to the bin and position of a space K in it.

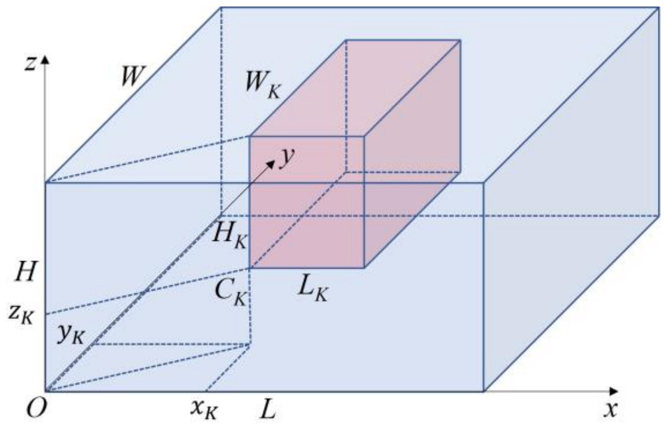


Figure 1. The position of the space K in the bin.

2.1. Main Data Structure

A space K is defined as the main space if its dimensions satisfy all three following conditions:

$$L_K \geq L_{min}, \tag{1}$$

$$W_K \geq W_{min}, \tag{2}$$

$$S_K \geq S_{min}, \tag{3}$$

where L_{min}, W_{min}, and S_{min} are the smallest length, width, and bottom area of the incoming box, respectively.

The main data structure is divided into layers. Each layer is a linked list of the main spaces that have the same coordinates in the z direction (z_K). The elements in this structure are linked using pointers as shown in Figure 2. The first node in each layer has four pointers that point to the previous, next, up, and down nodes. The other node in the layer has two pointers that point to the previous and next nodes. The layers are arranged in ascending order of parameter z_K. The data in each layer are arranged in ascending order of parameter S_K.

2.2. Secondary Data Structure

The space that fails to satisfy one of the Equations (1), (2) or (3) is classified as secondary space. This space is then added into the secondary data structure. This data structure is a linked list in which each node has two pointers that point to the previous and next nodes, as shown in Figure 3. The elements in the structure are sorted in chronological order.

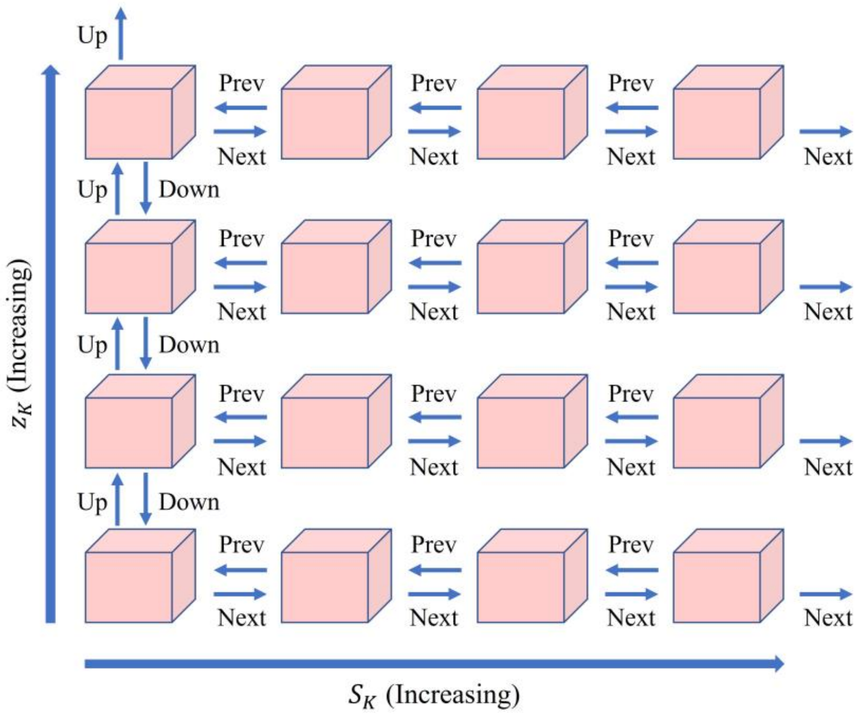


Figure 2. Main data structure diagram.

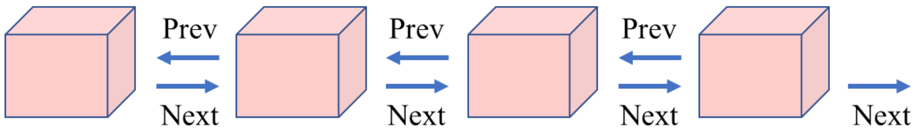


Figure 3. Secondary data structure diagram.

2.3. *Algorithm*

Whenever a box is coming, the algorithm finds a space K with the smallest z_K in the main data structure that fits the box. After that, this space is divided into 4 new subspaces. Each subspace is classified into main or secondary spaces according to its dimensions. The space K is removed from the main data structure. This structure is rearranged. The main spaces that satisfy the merging conditions are united into a new main space. In order to minimize the empty space in the bin, the secondary spaces are united with the main spaces. The main data structure is rearranged again. The flowchart of the algorithm is shown in Figure 4. The pseudo-code of the proposed method is described by Algorithm 1.

2.3.1. Finding a Fitted Space K

The space K can be used to place the incoming box if its dimensions satisfy Equations (4) or (5):

$$L_K \geq l,\ W_K \geq w,\ H_K \geq h, \tag{4}$$

$$L_K \geq w,\ W_K \geq l,\ H_K \geq h, \tag{5}$$

where l, w, and h are the length, width, and height of the incoming box, respectively.

These conditions allow the box to rotate 90° about the z axis when it is placed in the bin. If these conditions are not satisfied, the algorithm merges the consecutive main spaces to form a new main space that fits the incoming box. The conditions for merging are presented in Table 1. It is noted that the conditions $H_i \leq H$ and $H_j \leq H$ are applied for all these cases.

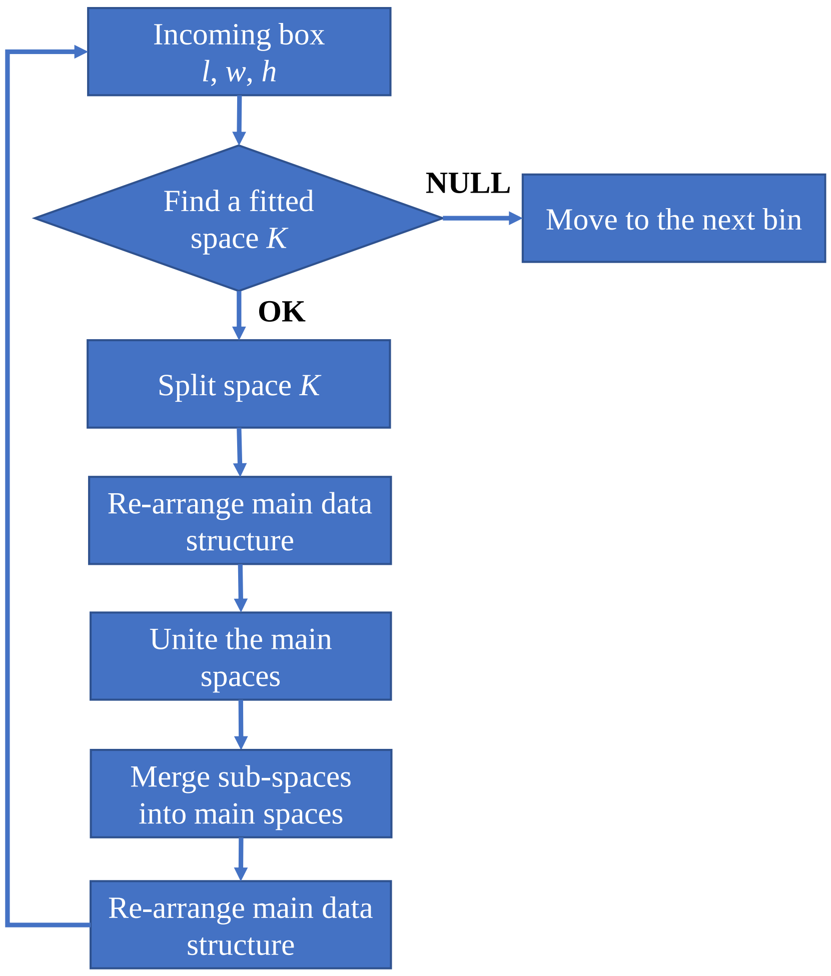
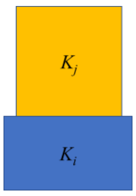
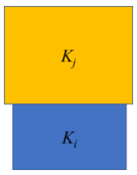
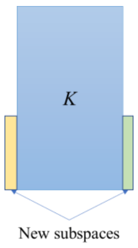
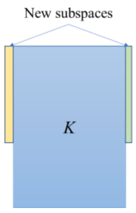
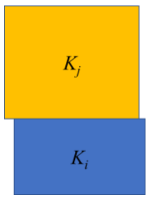
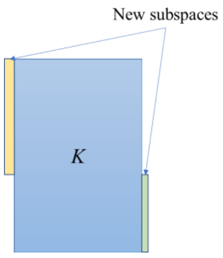
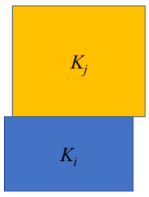
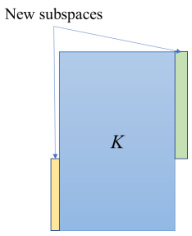
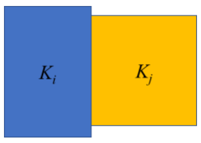
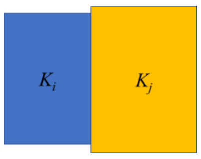
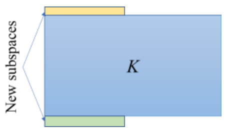
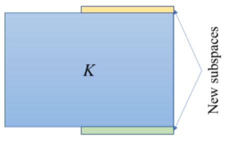
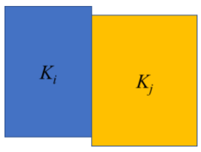
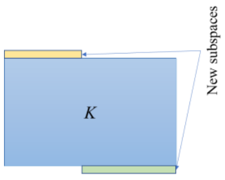
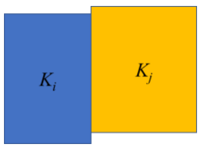
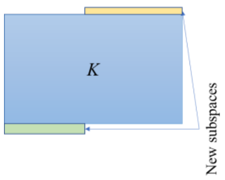


Figure 4. Flowchart of the developed online 3-D bin packing.

Algorithm 1: The 3-D online bin packing algorithm.

```
Input: incoming box $l, w, h$
Output: the position of the box in the bin
1: function PlaceBox($l, w, h$)
2:    $sp$ = FindSpace($l, w, h$) //To find the fitted space in the main data structure
3:    if $sp$ == NULL then
4:       $sp$ = FindConsecutiveSpaces(); //To find the fitted space by merging the consecutive main spaces
5:       if $sp$ == NULL then
6:          return false
7:       end if
8:    end if
9:    obtain the position of the box in the bin $x, y, z, l, w, h$
10:   SplitSpace($sp$); //To split the space sp
11:   ArrangeSpaces(); //To arrange main data structure
12:   UniteSpaces(); //To unite the main spaces
13:   OptimizeSpaces(); //To merge subspaces into main spaces
14:   ArrangeSpaces(); //To arrange main data structure
15:   return true
16: end function
```

Table 1. Merging the main spaces to find the fitted space.

Conditions	Merging Results	
	Main Spaces to Merge	New Spaces
$L_i \geq l$, $L_j \geq l$, $W_i + W_j \geq w$. or $L_i \geq w$, $L_j \geq w$, $W_i + W_j \geq l$.	 	 
$x_j + L_j - x_i \geq l$, $W_i + W_j \geq w$. or $x_j + L_j - x_i \geq w$, $W_i + W_j \geq l$.		
$x_i + L_i - x_j \geq l$, $W_i + W_j \geq w$. or $x_i + L_i - x_j \geq w$, $W_i + W_j \geq l$.		
$W_i \geq l$, $W_j \geq l$, $L_i + L_j \geq w$. or $W_i \geq w$, $W_j \geq w$, $L_i + L_j \geq l$.	 	 
$y_j + W_j - y_i \geq l$, $L_i + L_j \geq w$. or $y_j + W_j - y_i \geq w$, $L_i + L_j \geq l$.		
$y_i + W_i - y_j \geq l$, $L_i + L_j \geq w$. or $y_i + W_i - y_j \geq w$, $L_i + L_j \geq l$.		

2.3.2. Splitting the Space K

Assuming that the incoming box is placed into the space K in the bin, the bottom-left-front corner of the box is set to coincide with the bottom-left-front corner of that space, as shown in Figure 5. After that, the space K can be divided into 4 new spaces: K_1, K_2, K_3, and K_4. Depending on the dimensions of the incoming box and space K, the results of the splitting process are as presented in Table 2.

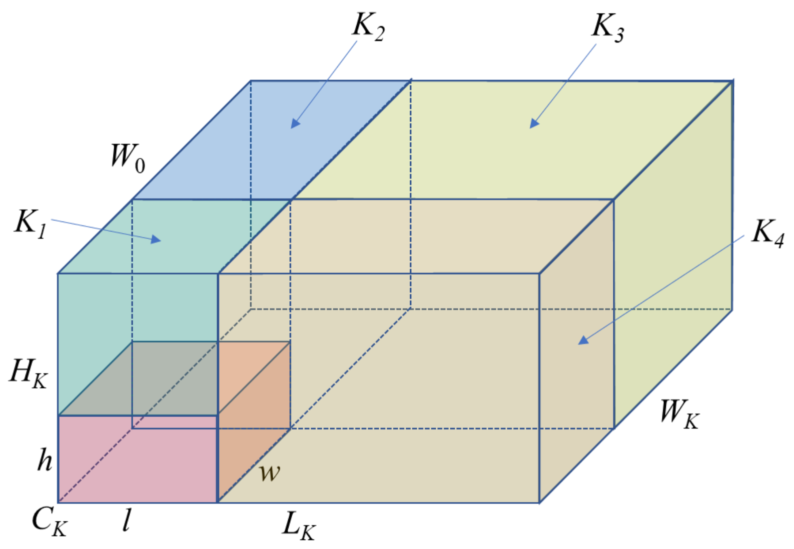


Figure 5. Splitting the space K.

Table 2. The splitting results.

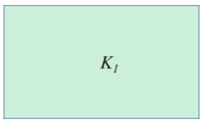
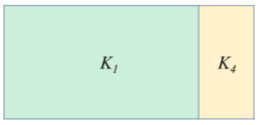
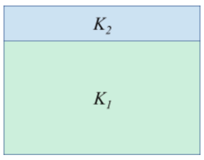
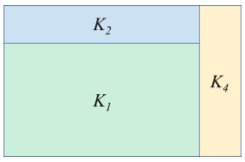
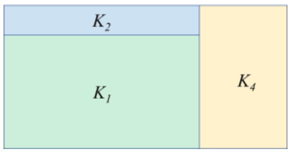
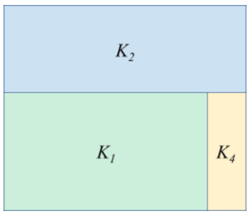
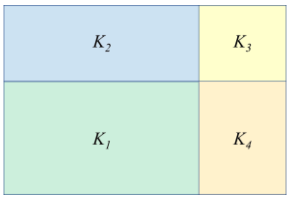
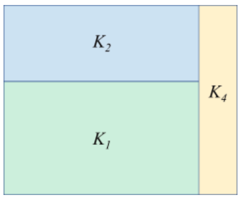
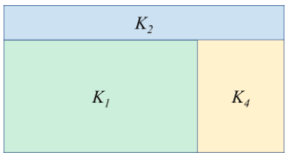
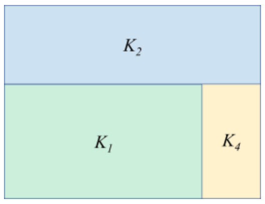
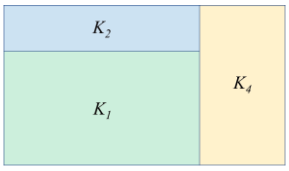
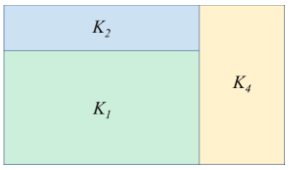
Conditions	Splitting Results	
Case 1: $L_K = l$ or $W_K = w$		
$L_K = l$, $W_K = w$.		1 main space K_1
$W_K = w$, $L_K > l$.		1 main space K_1 and 1 main/subspace K_4
$L_K = l$, $W_K \geq w$.		1 main space K_1 and 1 main/subspace K_2
Case 2: $L_K - l < L_{min}$ or $W_K - w < W_{min}$		
$L_K - l < L_{min}$, $W_K - w < W_{min}$.		1 main space K_1 and 2 subspaces K_2 and K_4

Table 2. *Cont.*

Case 2: $L_K - l < L_{min}$ or $W_K - w < W_{min}$		
$L_K - l \geq L_{min}$, $W_K - w < W_{min}$.		2 main spaces K_1 and K_4 and 1 subspace K_2
$L_K - l < L_{min}$, $W_K - w \geq W_{min}$.		2 main spaces K_1 and K_2 and 1 subspace K_4
Case 3: $L_K - l \geq L_{min}$ and $W_K - w \geq W_{min}$		
$S_2 \geq S_{min}$, $S_4 \geq S_{min}$.		3 main spaces K_1, K_2, K_4 and 1 main/subspace K_3
$S_2 \geq S_{min}$, $S_4 < S_{min}$.		2 main spaces K_1, K_2 and 1 main/subspace K_4
$S_2 < S_{min}$, $S_4 \geq S_{min}$.		2 main spaces K_1, K_4 and 1 main/subspace K_2
$S_2 < S_{min}$, $S_4 < S_{min}$, $S_2 > S_4$, $S_2 + S_3 > S_{min}$.		2 main spaces K_1, K_2 and 1 subspace K_4
$S_2 < S_{min}$, $S_4 < S_{min}$, $S_2 < S_4$, $S_3 + S_4 > S_{min}$.		2 main spaces K_1, K_4 and 1 subspace K_2
$S_2 < S_{min}$, $S_4 < S_{min}$, $S_2 + S_3 < S_{min}$, $S_3 + S_4 < S_{min}$.		1 main space K_1 and 2 subspaces K_2, K_4

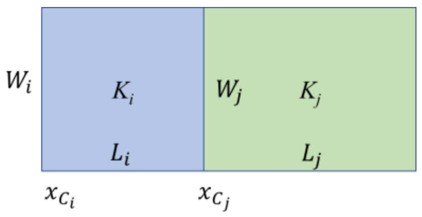
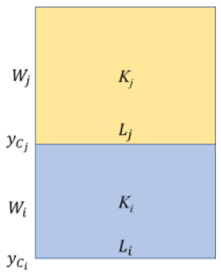
2.3.3. Rearranging the Main Data Structure

After adding data from the splitting process, the main data structure no longer retains its original organization. Therefore, this structure needs to be reorganized in a predetermined order. The sorting algorithm performs the task of traversing each layer of the main data structure. In a layer, the elements are sorted in ascending order of their areas, as shown in Figure 2.

2.3.4. Merging the Main Spaces

When the main spaces are added to the main data structure, they can be combined with other existing spaces to form new, larger spaces. The merging of these spaces helps to limit spatial fragmentation. The algorithm traverses each element in a layer until there are no more elements in the same layer that can be merged. Elements that satisfy the merging condition are united to form a new element. Elements used during the merging process are removed from the main data structure. The algorithm prioritizes merging data in the x direction, corresponding to the length of the bin first. After that, the data in the y direction corresponding to the width of bin are merged. Table 3 shows two cases that can be used to unite adjacent spaces.

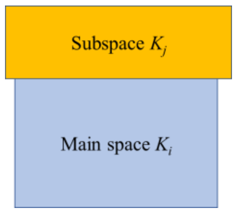
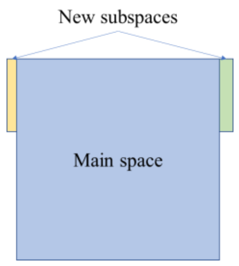
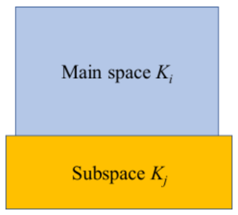
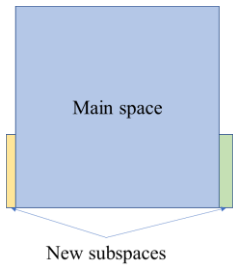
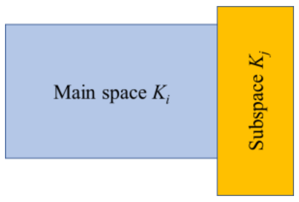
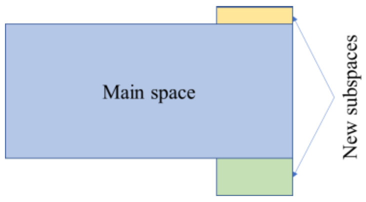
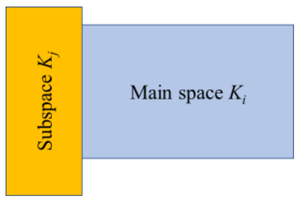
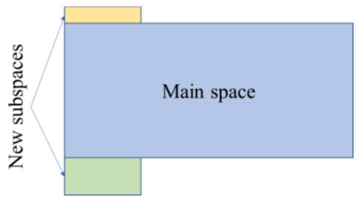
Table 3. Merging the main spaces to limit spatial fragmentation.

Conditions	Merging Results	
	Spaces to Merge	**New Spaces**
$W_i = W_j$, $y_i = y_j$, $x_j = x_i + L_i$.	 W_i K_i L_i W_j K_j L_j x_{C_i} x_{C_j}	$L_K = L_i + L_j$, $W_K = W_i = W_j$, $H_K = H_i = H_j$, $x_K = x_i$, $y_K = y_i$, $z_K = z_i$.
$L_i = L_j$, $x_i = x_j$, $y_j = y_i + W_i$.	 W_j K_j L_j y_{C_j} W_i K_i L_i y_{C_i}	$L_K = L_i = L_j$, $W_K = W_i + W_j$, $H_K = H_i = H_j$, $x_K = x_i$, $y_K = y_i$, $z_K = z_i$.

2.3.5. Merging the Secondary Spaces into Main Spaces

The secondary data structure contains elements describing small spaces, which cannot accommodate incoming boxes. These spaces cause the bin to fragment, reducing its capacity. Therefore, an algorithm was built to merge the subspace into the main space. The conditions for merging are presented in the Table 4.

Table 4. Merging the secondary spaces into main spaces.

Conditions	Merging Results	
	Spaces to Merge	New Spaces
$L_i \leq L_j$, $x_i \geq x_j$, $y_j = y_i + W_i$.		
$L_i \leq L_j$, $x_i \geq x_j$, $y_i = y_j + W_j$.		
$W_i \leq W_j$, $y_i \geq y_j$, $x_j = x_i + L_i$.		
$W_i \leq W_j$, $y_i \geq y_j$, $x_i = x_j + L_j$.		

3. Experimental Results and Analysis

To test the performance of the developed algorithm, the dataset proposed by Zhao et al. [24] was used without special declarations. There are 64 items in this dataset. The dimensions of the items range from two to five on each side. The bin sizes were set to $10 \times 10 \times 10$. The item sequences were created using random sampling (RS). Two thousand independent datasets were generated for this test. All methods were implemented in C++/Python and tested on a system with the following specifications: Intel® Core™ i7-7700HQ CPU @ 2.80 GHz (8 CPUs, ~2.8 GHz) with 8.00 GB RAM, NVIDIA GeForce GTX 1050, Windows 10 Pro 64-bit. The result of the proposed algorithm was compared with those of different methods, as shown in Table 5.

Table 5. The packing results of different methods on the Zhao et al. [24] dataset.

Method	Space Utilization (%)	No. of Items
Proposed method	**51.7**	**12.7**
Zhao et al. (trained on RS) [24]	50.5	12.2
Zhao et al. (boundary rule) [24]	34.9	8.7
Karabulut and İnceoğlu [25]	43.1	10.6
Ha et al. [26]	37.1	9.1

The performance of the developed algorithm was also verified with different bin sizes. In this experiment, the dimensions of the bins were increased from 10 × 10 × 10 to 20 × 20 × 20 and 30 × 30 × 30. The packing results are presented in Figure 6 and Table 6. As shown in Table 6, the performance slightly improved when the size of the bin was larger, especially when the size of the bin was 30 × 30 × 30. In comparison with the method proposed by Zhao et al. [24], the developed algorithm can achieve higher average space utilization when the bin size is 30 × 30 × 30.

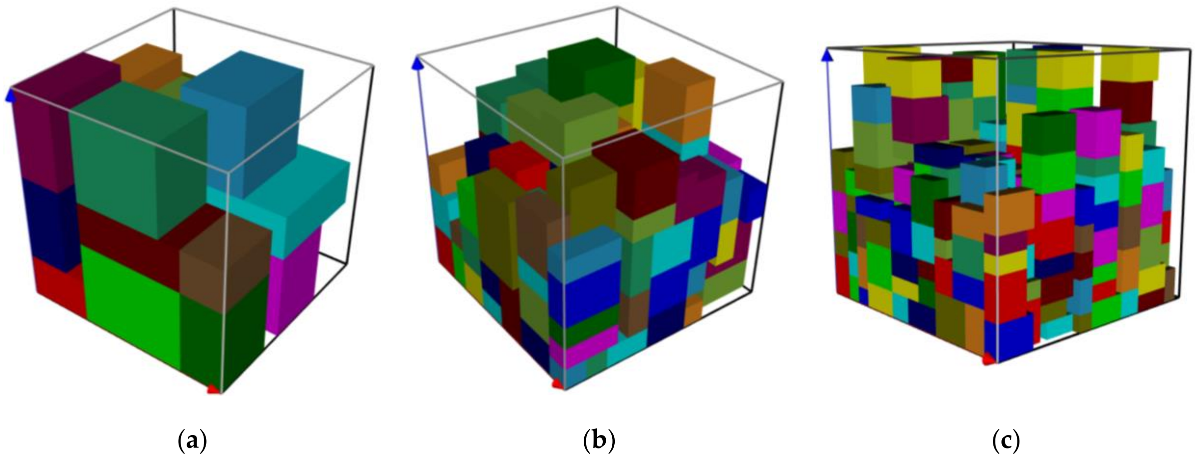


Figure 6. The packing results with different bin sizes: (**a**) bin size of 10 × 10 × 10; (**b**) bin size of 20 × 20 × 20; (**c**) bin size of 30 × 30 × 30.

Table 6. The packing results with different bin sizes.

Bin Size ($L \times W \times H$)	Space Utilization (%)		No. of Items	Time per Item (ms)
	Proposed Method	Zhao et al. [24]		
10 × 10 × 10	**51.7**	50.5	12.7	0.04
20 × 20 × 20	51.9	**58.1**	98.1	0.23
30 × 30 × 30	**60.5**	49.0	382.5	1.40

The algorithm was also tested with multiple bins in order to optimize the average space utilization of the bin. In this experiment, if an incoming item could not be placed into the current bin, the fit in the next bin was checked. The process was finished if it could not put the items into any of the existing bins. Tables 7 and 8 show the packing results for different bins when a multibin packing algorithm was applied. It can be seen that the space utilization of each bin greatly increased when the number of bins increased. However, the average computation time for each item also increased. This means that it took more time to place an item into the bin.

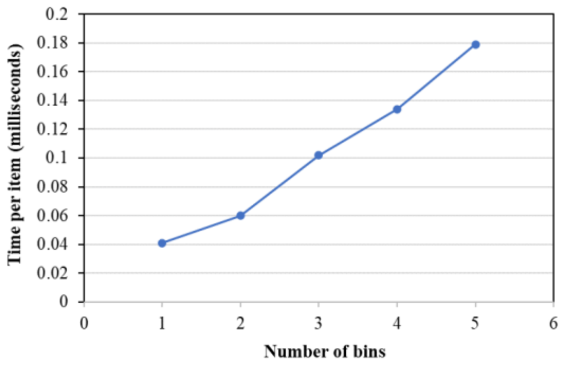
Table 7. The average percentage filled (%) of each bin in case of multibin packing.

No. of Bins	1st Bin	2nd Bin	3rd Bin	4th Bin	5th Bin	Average
1	51.7					51.7
2	70.3	61.0				65.7
3	77.5	75.9	60.3			71.2
4	80.5	81.6	75.1	60.4		74.4
5	**81.7**	**83.8**	**81.5**	**75.3**	61.0	**76.7**

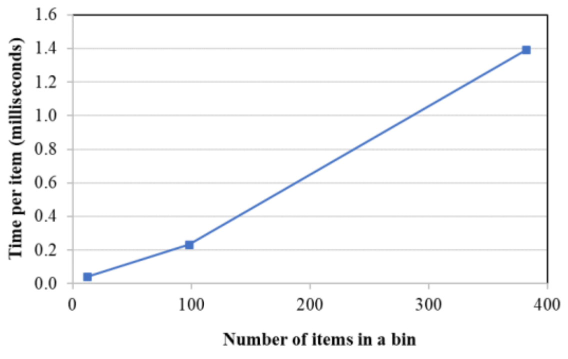
Table 8. The average number of items in each bin in case of multibin packing.

No. of Bins	1st Bin	2nd Bin	3rd Bin	4th Bin	5th Bin	Average
1	12.7					12.7
2	20.5	10.8				15.7
3	24.2	15.9	10.4			16.8
4	26.1	18.4	15.3	10.3		17.5
5	**27.2**	**19.4**	**18.0**	**15.1**	10.3	**18.0**

The time performance of the developed algorithm is shown in Figure 7. It can be seen that the average time required to determine the location of incoming items in the bin linearly grows with the number of bins and size of the bin. In the case of a bin size of $10 \times 10 \times 10$, the average time per item is less than 0.18 milliseconds when the number of bins is 5, as shown in Figure 7a. If the bin can carry many items, it will take more time to calculate the position of the incoming item. Figure 7b presents the relationship between the number of items that can be put into a bin and the time to estimate the position of an item. The average time is less than 1.4 milliseconds while the average number of items is about 382.5. In comparison with the method proposed by Zhao et al. [24], where the time required was about 100 milliseconds for a bin size of $10 \times 10 \times 10$, the proposed method is faster.



(**a**)



(**b**)

Figure 7. The time performance of the proposed algorithm: (**a**) the average time vs. number of bins; (**b**) the average time vs. the average number of items that can be put into a bin.

The feasibility of the proposed algorithm was also tested on real data. The dimensions of the items and bins are presented in Table 9. The same items were packed into the bin. The number of items ranged from 2 to 12. In these experiments, all the bins were filled with the items, as expected.

Table 9. The packing results for different bin sizes.

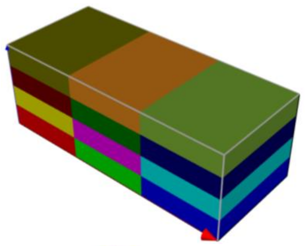
Case Study	Item Size ($l \times w \times h$)	Bin Size ($L \times W \times H$)	No. of Items	Packing Results
1	$95 \times 75 \times 20$	$225 \times 95 \times 80$	12	

Table 9. *Cont.*

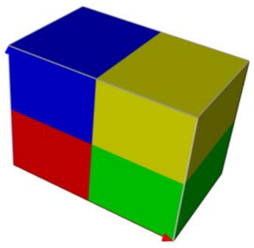
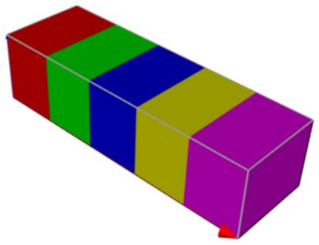
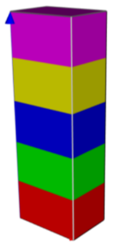
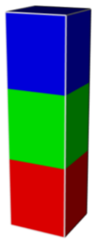
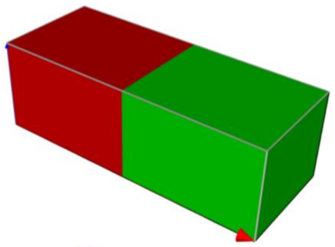
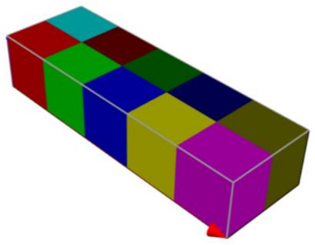
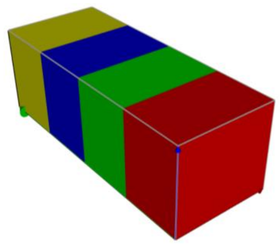
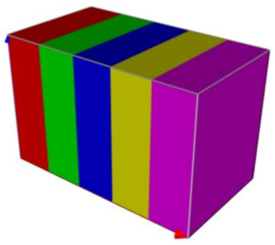
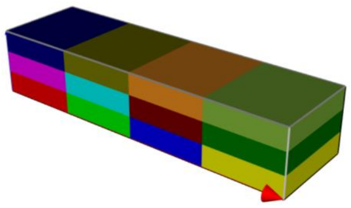
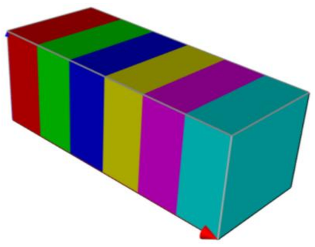
Case Study	Item Size ($l \times w \times h$)	Bin Size ($L \times W \times H$)	No. of Items	Packing Results
2	95 × 75 × 55	150 × 95 × 110	4	
3	95 × 60 × 75	300 × 95 × 75	5	
4	100 × 70 × 60	100 × 70 × 300	5	
5	90 × 75 × 70	75 × 270 × 70	3	
6	115 × 90 × 75	230 × 90 × 75	2	
7	50 × 40 × 50	250 × 80 × 50	10	

Table 9. *Cont.*

Case Study	Item Size ($l \times w \times h$)	Bin Size ($L \times W \times H$)	No. of Items	Packing Results
8	85 × 55 × 75	85 × 220 × 75	4	
9	25 × 70 × 80	125 × 70 × 80	5	
10	85 × 75 × 20	300 × 85 × 60	12	
11	85 × 35 × 75	210 × 85 × 75	6	

The above experimental results show that the developed method can pack boxes with a high utilization rate and low time consumption. However, the proposed algorithm has a limitation when the number of boxes is very large. The container is divided into too many subspaces, which increases the memory consumption of the two data structures. This affects the performance and memory requirements of the packing process.

4. Conclusions

This paper proposed an online 3-D bin packing algorithm employing a space splitting and merging technique. This technique involves categorizing the split spaces into two types of data structures. Furthermore, the merging of empty spaces helps to reduce spatial fragmentation. The search algorithm only operates on the main data structures to save computation time. The proposed method was evaluated and benchmarked with other methods on the same dataset. The experimental results showed that the proposed method could achieve better results. The developed technique was also applied to real data to verify its feasibility. The algorithm is well suited for robot bin-packing applications. Future work will focus on optimizing the empty space in each layer to increase the average percentage filled.

Author Contributions: Conceptualization, T.-H.N. and X.-T.N.; methodology, T.-H.N.; software, T.-H.N.; validation, T.-H.N. and X.-T.N.; formal analysis, T.-H.N. and X.-T.N.; investigation, T.-H.N.; resources, T.-H.N.; data curation, T.-H.N.; writing—original draft preparation, T.-H.N.; writing—review and editing, T.-H.N. and X.-T.N.; visualization, T.-H.N.; supervision, T.-H.N.; project administration, T.-H.N.; funding acquisition, T.-H.N. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the Vietnam Ministry of Education and Training (project number B2020-BKA-16).

Data Availability Statement: The data used to support the findings of this study are included within the article.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. De Castro Silva, J.L.; Soma, N.Y.; Maculan, N. A greedy search for the three-dimensional bin packing problem: The packing static stability case. *Int. Trans. Oper. Res.* **2003**, *10*, 141–153. [CrossRef]
2. Johnson, D.S.; Demers, A.; Ullman, J.D.; Garey, M.R.; Graham, R.L. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.* **1974**, *3*, 299–325. [CrossRef]
3. Martello, S.; Toth, P. Lower bounds and reduction procedures for the bin packing problem. *Discret. Appl. Math.* **1990**, *28*, 59–70. [CrossRef]
4. Delorme, M.; Iori, M.; Martello, S. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *Eur. J. Oper. Res.* **2016**, *255*, 1–20. [CrossRef]
5. Martello, S.; Vigo, D. Exact solution of the two-dimensional finite bin packing problem. *Manag. Sci.* **1998**, *44*, 388–399. [CrossRef]
6. Lodi, A.; Martello, S.; Monaci, M. Two-dimensional packing problems: A survey. *Eur. J. Oper. Res.* **2002**, *141*, 241–252. [CrossRef]
7. Zhang, D.; Kang, Y.; Deng, A. A new heuristic recursive algorithm for the strip rectangular packing problem. *Comput. Oper. Res.* **2006**, *33*, 2209–2217. [CrossRef]
8. Cid-Garcia, N.M.; Rios-Solis, Y.A. Positions and covering: A two-stage methodology to obtain optimal solutions for the 2d-bin packing problem. *PLoS ONE* **2020**, *15*, e0229358. [CrossRef]
9. Martello, S.; Pisinger, D.; Vigo, D. The three-dimensional bin packing problem. *Oper. Res.* **2000**, *48*, 256–267. [CrossRef]
10. Baltacioglu, E. The Distributer's Three-Dimensional Pallet-Packing Problem: A Human Intelligence Based Heuristic Approach. Ph.D. Thesis, Air Force Institute of Technology, Dayton, OH, USA, 2001.
11. Maarouf, W.F.; Barbar, A.M.; Owayjan, M.J. A new heuristic algorithm for the 3D bin packing problem. In *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering*; Elleithy, K., Ed.; Springer: Dordrecht, The Netherlands, 2008; pp. 342–345.
12. Liu, Y.; Vo, T.V. Bin packing solution for automated packaging application. *Appl. Mech. Mater.* **2012**, *143–144*, 279–283. [CrossRef]
13. Wu, Y.; Li, W.; Goh, M.; de Souza, R. Three-dimensional bin packing problem with variable bin height. *Eur. J. Oper. Res.* **2010**, *202*, 347–355. [CrossRef]
14. Feng, X.; Moon, I.; Shin, J. Hybrid genetic algorithms for the three-dimensional multiple container packing problem. *Flex. Serv. Manuf. J.* **2015**, *27*, 451–477. [CrossRef]
15. Linkosaari, T.; Urponen, T.; Juvonen, H.; Mäkelä, M.M.; Nikulin, Y. Three-dimensional bin packing problem with a stability rejection criterion. In Proceedings of the VII European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS Congress, Crete Island, Greece, 5–10 June 2016.
16. Faroe, O.; Pisinger, D.; Zachariasen, M. Guided local search for the three-dimensional bin-packing problem. *INFORMS J. Comput.* **2003**, *15*, 267–283. [CrossRef]
17. Crainic, T.G.; Perboli, G.; Tadei, R. TS2PACK: A two-level tabu search for the three-dimensional bin packing problem. *Eur. J. Oper. Res.* **2009**, *195*, 744–760. [CrossRef]
18. Pedruzzi, S.; Nunes LP, A.; Rosa RD, A.; Arpini, B.P. A mathematical model to optimize the volumetric capacity of trucks utilized in the transport of food products. *Gestão Produção* **2016**, *23*, 350–364. [CrossRef]
19. Zhao, H.; Yu, Y.; Xu, K. Learning Efficient Online 3D Bin Packing on Packing Configuration Trees. *Int. Conf. Learn. Represent.* **2022**.
20. Hu, H.; Zhang, X.; Yan, X.; Wang, L.; Xu, Y. Solving a new 3D bin packing problem with deep reinforcement learning method. *arXiv* **2017**, arXiv:1708.05930.
21. Duan, L.; Hu, H.; Qian, Y.; Gong, Y.; Zhang, X.; Wei, J.; Xu, Y. A multi-task selected learning approach for solving 3D flexible bin packing problem. *arXiv* **2019**, arXiv:1804.06896.
22. Hu, R.; Xu, J.; Chen, B.; Gong, M.; Zhang, H.; Huang, H. TAP-Net: Transport-and-Pack using reinforcement learning. *ACM Trans. Graph.* **2020**, *39*, 232:1–232:15. [CrossRef]

23. Zhang, J.; Zi, B.; Ge, X. Attend2Pack: Bin packing through deep reinforcement learning with attention. *arXiv* **2021**, arXiv:2107.04333.
24. Zhao, H.; She, Q.; Zhu, C.; Yang, Y.; Xu, K. Online 3D bin packing with constrained deep reinforcement learning. *arXiv* **2020**, arXiv:2006.14978. [CrossRef]
25. Karabulut, K.; İnceoğlu, M.M. A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method. In *International Conference on Advances in Information Systems*; Yakhno, T., Ed.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 441–450.
26. Ha, C.T.; Nguyen, T.T.; Bui, L.T.; Wang, R. An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the Physical Internet. In *European Conference on the Applications of Evolutionary Computation*; Squillero, G., Sim, K., Eds.; Springer: Cham, Switzerland, 2017; pp. 110–127.
27. Verma, R.; Singhal, A.; Khadilkar, H.; Basumatary, A.; Nayak, S.; Singh, H.V.; Kumar, S.; Sinha, R. A generalized reinforcement learning algorithm for online 3D bin-packing. *arXiv* **2020**, arXiv:2007.00463.
28. Nguyen, T.-H.; Tran, V.-T.; Doan, P.-Q.; Mac, T.-T. A Novel Heuristic Algorithm for Online 3D Bin Packing. In Proceedings of the 2021 21st International Conference on Control, Automation and Systems (ICCAS), Jeju, Republic of Korea, 12–15 October 2021; pp. 1993–1997.