Smart Shipment: An Efficient Algorithm for Packing Three-Dimensional Bins

Y. Harrath, M. Aljassim, L.M. Anees

Department of Computer Science, University of Bahrain, Kingdom of Bahrain
yharrath@uob.edu.bh, manar.aljassim@gmail.com and lyebaanees35@gmail.com

Keywords: Bin-Packing, 3D, Smart Packing, Rotation.

Abstract

The 3D Bin Packing Problem is a well-known NP hard problem which aims at minimizing the number of bins for packing a given number of boxes. The tackled problem includes the case where boxes are allowed to be freely rotated and be placed in one of their six distinct phases, all for the purpose of utilizing the bin's space. A number of algorithms have been proposed by various researchers in the hope of providing optimal or near-optimal solutions for the problem. This paper proposes a new algorithm that was implemented and successfully tested and have provided improved results compared to two previously published algorithms. A theoretical lower bound was also used for the purpose of validation and comparison. The time taken by the new algorithm to pack the boxes in the bins is very small (less than one minute) even for large instances of 100000 boxes. One of the major achievements of the proposed algorithm is the maximum utilization bins' spaces.

1 Introduction

A bin is a container which accommodates items or boxes inside it for the purpose of their transportation from one location to another. The way to fill boxes inside these containers is the primary focus of this paper. Packing a box in a bin may seem to be a simple task, as one can start from any area and place the item anywhere. When the bin is first empty, there is a freedom of choosing any location inside it to situate the box in, in any possible rotation. This matter can continue for some boxes but reaching a case of improper arrangements can lead to having huge gaps and wasted spaces inside the bin. The likelihood of losing this freedom in placement of remaining items is tremendous. Taking a first look into this issue, may seem as a minor problem; if a box does not fit inside the bin, then a change in the placements of existing items can be applied so that it can fit in, or as a last resort, another bin can be used to pack the item. Both solutions are in fact reasonable to an extent, however, there are some constraints to be recognized: the burden of rearranging the items already placed and the cost of using an extra bin, as each added bin can potentially increase the cost. So ultimately, time, effort and expenses are at risk. Packing items in an effective way from the starting point is crucial, since it can potentially save a great deal of resources. The impact of previously mentioned problems is not something that can be overlooked easily if the packing matter is on a large scale (commercially, for companies). Therefore, having an algorithm that fixes these issues by packing the boxes in a smart way is significant.

2 Literature Review

The two bin packaging algorithms used in [1] were the First Fit Decreasing and The Best Fit. These algorithms were chosen over many other algorithms because of their fast running time and their ability to produce optimal solutions. They considered rotations of the three directions, knowingly, the width, depth and height directions to pack the boxes in the minimum number of bins. The authors emphasize on the final number of bins used and highlight an interesting point regarding the wasted space and the used space, which states that the value of the wasted space is only an indicator to the user of the possibility of packing more items in the bin, and not a scale for determining if the solution is better.

The authors in [2] provided an approach known as "binpack3d" with the aim of minimizing the number of rectangular bins while packing them with n number of boxes along with rotations. Their research follows a depth-first strategy of an algorithm called "rec_binpack", which is recursively performed to check if the bins are filled or if there is still a possibility of placing another box to completely fill the bin. The authors use another algorithm to check the feasibility of the assignment of boxes, which is known as the "onebin_decision". However, it was noted that the best performance was given by the algorithm called "onebin_general" which was by sorting the boxes in their non-increasing volumes and then branching them into pairs. The relative position of the largest boxes was obtained in the beginning of the algorithm and infeasibility could be known at the earliest. Their research shows that most of the time of their algorithms was spent in deciding the routines for filling the bin and hence they proposed that future work can be done in solving this decision-making problem.

A novel algorithm was proposed in [3] which considers the volume, weight and optimizes the number of bins required to pack different sized boxes. The algorithm packed items in horizontal layers and the solution followed a layer-based approach. The algorithm showed improved results in terms of number of bins and execution time. However, the authors did not consider rotations of items and layering of items of mixed types.

The paper in [4] proposed a Multi objective 3D Bin packing algorithm that is focusing on simultaneous objectives that mainly concentrates on efficiency packing and balancing the weight of the bins. The only drawback faced by that algorithm was the large amount of wasted space in the bins which could not be filled. A Particle Swarm Optimization (PSO) algorithm was proposed in [5] to fit the boxes in the bins. The algorithm was mainly used to choose a suitable bin to fit the boxes based on their sizes and total volume. Moreover, the researchers also used this algorithm to decide the position of each item in the bin. They believed that their proposed algorithm provides high convergence speed in delivering the calculations of different kinds of combinations. The proposed PSO–based algorithm is believed to solve the NP hard 3D bin packing problem with strong heterogeneity of boxes and variable size of bins, but the algorithm does not cover minimizing the number of bins used for packing the boxes.

Miyazawa and Wakabayashi presented approximation algorithms for the three-dimensional bin packing problem by considering orthogonal packing's with rotations of 90 degrees [6]. The main focus is to pack the items with minimum number of bins considering that a bin has limited dimensions. By providing feasible orientations of 90 degrees, items can be packed into the bin without the need of any other orientation. The proposed algorithm uses the same strategies as used in the polynomial approaches *Hybrid First Fit* and *First Fit Decreasing Combine*.

Based on the above literature review, it was noticed that the problem has been researched for a long period and different algorithms and solutions were proposed by many renowned researchers. It was noticed that with the combinations of existing algorithms, researchers have proposed new and improved algorithms with a peculiar aim to keep in track with emerging changes in the world.

3 Problem Statement

The studied 3D bin packing problem is targeting the packing of a given number of boxes in the least number of bins efficiently, in a way that the bin's space is utilized to the maximum. The boxes given are allowed to be rotated freely in any of the six possible orientations. Boxes are placed along the three coordinates and cannot surpass the dimensions of the bin. The problem's explanation begins by describing its parameters:

- The bin's dimensions are its length, depth and height values. These values are taken as an input. It is assumed that the length, depth and height are ordered in descending order.
- The box's dimensions are its length, depth and height values. These values are taken as an input.
- Box types: The dimensions of a box will identify its type. The proposed solution considers two types of boxes only.

It will be assumed that the boxes to be packed can be rotated in any direction along the three coordinates while disregarding the maximum weight capacity of the bin.

4 Proposed Solution

A heuristic algorithm was developed to reach the goal of packing the boxes inside bins in an efficient way. The proposed solution is composed of two main parts called *setting boundaries* and *filling remaining spaces* to fill a single bin with a number of boxes. The two parts of the algorithm will be repeated until the last box is packed successfully inside a bin.

4.1 Setting Boundaries

This part of the algorithm places homogeneous layers of boxes inside the bin. These placements will be leaving areas from within the bin that will be filled with boxes in the next part of the algorithm.

The starting step of this part is the selection for the box to be placed first in the bin. In this step, a box type is selected and a rotation of it is determined. Once done, the selection will not be changed for this part. Deciding which type to select is done by examining the dimensions of each type of box. The box type that has the greatest dimensional value is selected. This can be looked as finding the maximum value of the six values given, which are the length, depth and height for both box types.

To decide the box orientation, the box type selected will be rotated so that its longest dimension is at the length of the bin, the second longest dimension is at the depth and the height will have the last dimensional value of the box placed against it. To explain this step further, imagine a box being placed at the origin of the yxz coordinates, which is at point (0,0,0). Since a box is a 3D shape, it has values (points) in all of these three dimensions. If this box is rotated, then the values in the three dimensions will be flipped, based on how the box is placed. Dube and Kanavathy in [1] explained this concept of the rotation: "Rotating an item simply means swapping its width, height and depths values around". Once the type and rotation of the box is selected, the packing starts with boxes of that type being placed along the bin's length first, depth then height, which is along the y,x and z coordinates respectively. The number of boxes placed will depend on two main points: number of available boxes to be packed of the selected type and the space it leaves of the bin unfilled after being placed. The calculations made are for computing the *possible* number of boxes to be placed. The results are then adjusted with the considerations for these two points. Starting from the length, the number of boxes possible to place denoted by n_y is computed by dividing the length of the bin L by the box's dimension d_1 which is the one placed against the length, as shown in Equation 1.

$$n_y = \lfloor \frac{L}{d_1} \rfloor \tag{1}$$

The value n_y will be checked against the aforementioned points. it is important to examine this result and consider the available boxes and remaining space that might be obtained after placing the boxes. The remaining space, r_1, is a space resulting after placing the boxes, between the last point of the placed box until the bin's end along the dimension which is often regarded as a wasted space since it cannot occupy another

box, in any rotation. This space can be depicted by the arrow found in Figure 1. The remaining space is calculated by Equation 2.

$$r_1 = L \bmod d_1 \tag{2}$$

To reach a maximum utilization, this space will be adjusted by making sure it can at least occupy any other box, in any rotation. Adjusting the space is done by changing (decrementing) the number of boxes to be placed, which is n_y. Determining if the space fits a box is done by checking r_1 with the boxes' dimensions. A space that fits a box is a space that can at least accommodate the minimum dimensional value of the two box types. If n_y is greater than the number of available boxes, then the value will be changed so that it matches the count of the unpacked boxes. Once the two points are satisfied, boxes of the selected type and orientation will be loaded into the bin along its length, forming one row and a space at the end, waiting to be filled. Figure 1 shows a visual for this step, and lines 1-4 in Algorithm 1 is the pseudo-code of this step.

Algorithm 1 Forming a layer with maximum number of boxes

1: **while** $r_1 < min$(boxes' dimensions) and $n_y >$ available boxes **do**
2: $\quad n_y = n_y - 1$
3: $\quad r_1 = r_1 + d_1$
4: **end while**
5: **while** $r_2 < min$(boxes' dimensions) and $n_y * n_x >$ available boxes **do**
6: $\quad n_x = n_x - 1$
7: $\quad r_2 = r_2 + d_2$
8: **end while**

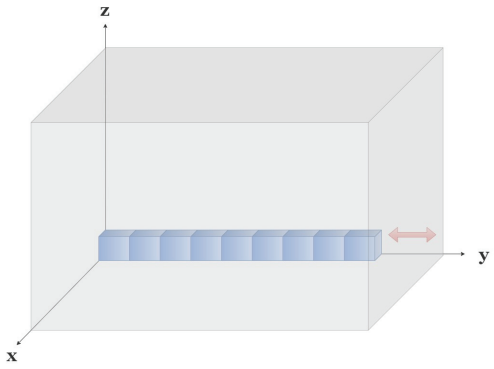

Figure 1. Placing boxes along the length of the bin.

Next comes the step of filling the bin with boxes along its depth, which can be viewed as placing several rows that are identical to the previously placed row. The number of rows n_x can be determined by dividing the depth D over the dimensional value of the box which was placed against it, that is denoted by d_2, given by Equation 3.

$$n_x = \lfloor \frac{D}{d_2} \rfloor \tag{3}$$

This result is also examined with regards to the previously discussed points, following the same approach done when the first row of boxes was situated inside the bin. Equation 4 calculates r_2 which is the remaining space while steps 5-8 of Algorithm 1 includes the pseudo-code of this step.

$$r_2 = D \bmod d_2 \tag{4}$$

Note that the number of available boxes is checked with the result $n_y * n_x$. If this result is greater than the number of unpacked boxes, only n_x will be decremented. Decrementing n_x can be visualized as removing rows of the boxes to be placed. The reason behind this step is to reach a better space utilization of the bin and to avoid having uneven layers to be placed. Once the two points are checked, the boxes will be loaded into the bin along its depth, forming one layer and a space waiting to be filled. This can be visualized in Figure 2.

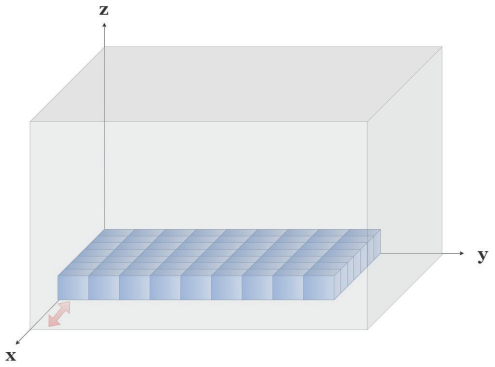

Figure 2. Placing boxes along the depth of the bin.

The final step in this part is the placement of boxes along the height of the bin. This is considered as placing a number of layers of the previously placed layer. Following the same procedure in the previous steps, the number of layers n_z is calculated by dividing the height of the bin H over the dimensional value of the box d_3, as shown in Equation 5. The remaining space here, r_3, calculated by Equation 6, along with the checking of the count of unpacked boxes, will be considered and adjusted accordingly as it can be seen in Algorithm 2.

$$n_z = \lfloor \frac{H}{d_3} \rfloor \tag{5}$$

$$r_3 = H \bmod d_3 \tag{6}$$

Algorithm 2 Forming layers with maximum number of boxes

1: **while** $r_3 < min$(boxes' dimensions) and $n_y * n_x * n_z >$ available boxes **do**
2: $\quad n_z = n_z - 1$
3: $\quad r_3 = r_3 + d_3$
4: **end while**

Once the number of layers is calculated, the boxes will be placed inside the bin as it is shown in Figure 3. The available

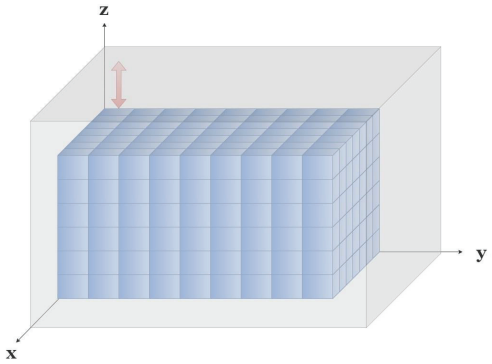

Figure 3. Placing boxes along the height of the bin.

number of boxes unpacked will be updated as shown in Equation 7.

$$available\_boxes = available\_boxes - n_y * n_x * n_z \quad (7)$$

Executing the previous steps might leave a bin not completely filled and having empty spaces surrounding the boxes placed earlier. For filling these remaining spaces, the algorithm divides them and fills each area separately. The result of this division is shown in Figures 4 and 5.

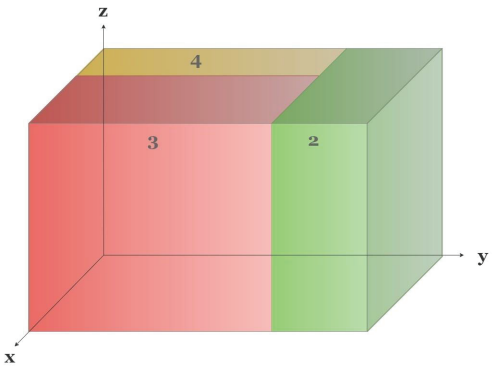

Figure 4. Bin after defining the areas' boundaries.

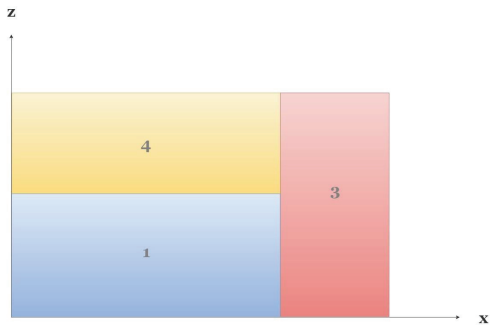

Figure 5. Side view of the bin after defining the areas' boundaries.

Figures 4 and 5 show a bin after division. Four areas are created, 3 new areas and 1 area which is the space occupying the layers placed earlier. These areas are labeled and defined inside the bin, where their boundaries are determined, which in turn explains the reason behind the name of this part.

4.2 Filling remaining spaces

Once the bin is divided into four unique spaces, the packing continues starting from the second area until reaching the fourth. To start, a box type and orientation is selected. Selecting the box type and rotation is determined by one of the dimensions of these areas, which are the values of the remaining spaces, r_1, r_2 and r_3 for the 3 created areas respectively. In most cases, this value is the minimum of the other dimensions of the labeled area, which would limit the box's selection, since the box has to fit within this particular area. To select a box, the dimensions for the two types are examined: The box type that has the greatest dimensional value which fits in the area is selected. This can be demonstrated by Algorithm 3. Specifying the dimension in which the box is placed ultimately leaves the option of having two possible rotations for the box. The orientation of the box that allows for the maximum number of boxes to be placed will be selected. Just like the previous part, the selection will not change for each area.

Algorithm 3 Selecting box type to place in an area

1: **if** max(box 1 dimensions fitting in the space) $<$ max(box 2 dimensions fitting in the space) **then**
2: Select box 2
3: **else**
4: Select box 1
5: **end if**

Finding the boxes with the largest dimensional value fitting in an area is important since placing the appropriate box and the appropriate orientation will reduce the wasted space in the bin. In some cases, the value of the remaining space can fit more than one box. If the space can fit only a single box, then the algorithm continues. This calculation is done in Equation 8. a_1 and a_2 are the other area's dimensions while bd_1 and bd_2 are the other box's dimensions.

$$max(\lfloor\frac{a_1}{bd_1}\rfloor * \lfloor\frac{a_2}{bd_2}\rfloor, \lfloor\frac{a_1}{bd_2}\rfloor * \lfloor\frac{a_2}{bd_1}\rfloor) \quad (8)$$

Otherwise, a calculation will be done to find the maximum number of boxes that can be placed, by finding all possible placements of rotating the selected box type. After calculating the number of boxes and filling it in the area, the algorithm proceeds to the other two areas and calculates how many boxes can be placed following the same approach. Because the calculated number of boxes is an indicator for the possible number of boxes to be placed and not the actual count of unpacked boxes, another approach will be done when the calculated number of boxes is not available, where the algorithm will fill the area with boxes of the other type. This step will be repeated for the third and fourth area. Filling the fourth area will ultimately fill the entire bin. Once this is done, the algorithm proceeds filling the remaining boxes in other bins, following the same steps of

Type	Length	Depth	Height	Max. Weight
Bin	317.5 cm	243.8 cm	178 cm	6804 kg
Box 1	25.88 cm	40.16 cm	32.86 cm	10 kg
Box 2	33.5 cm	54.8 cm	42.1 cm	25 kg

Table 1. Input Data.

the two parts of the algorithm and stops when the last box is packed inside a bin.

5 Implementation and Results

The proposed algorithm, implemented in Python, begins computation by being provided with the dimensions of the two box types and the bin. The data used in testing is found in Table 1. The boxes' data is taken from an excel sheet. This file has the information for each box, such as the id, type and count. The results produced by the program will be listed and compared with the lower bounds of the same number of boxes as well as the results found in the algorithms of papers [3] and [4]. These two algorithms have used the same data set.

After executing the program with the given inputs, the results are listed in Table 2 along with the results of the two other algorithms. In this table, Column n_1 and n_2 under *Instances* lists the number of boxes for type 1 and 2 respectively. Column lb is for the computed lower boundary for the number of boxes combined in each entry. Columns b_1 and b_2 are listing the results of papers [4] and [3] respectively. Column b lists the results that were computed from this paper's algorithm. Observing the number of bins that each algorithm has calculated, it can be clearly seen that the proposed algorithm of this paper has the least values of them all. The algorithm has packed the boxes efficiently in less number of bins than the other algorithms. Although there is a considered difference between the results, it is important to mention that the other two algorithms of papers [3] and [4] have taken into consideration the weights of each box placed inside the bins and ensures a better weight distribution of them, while the proposed algorithm has regarded this matter as one of its limitations.

As the number of boxes to pack increases, the calculated time taking to compute the results will also increase. The time taken for this paper's algorithm in addition to the algorithms in [3] and [4] is graphed in figure 6. Observing this figure may give the assumption that the proposed algorithm consumes more time than the other algorithms, especially for smaller number of boxes to produce the results. But as the number of boxes increases, the time recorded for the computed results is remarkably less than the time taken by the other algorithms, which can be considered as an advantage for this paper's algorithm.

6 Conclusion

The proposed algorithm has succeeded in packing boxes inside bins efficiently with the ability of rotating them. The main contribution of this research was providing an algorithm which

Instances		lb	b_1	b_2	b
n_1	n_2				
39	61	1	1	1	1
200	300	3	3	3	3
390	610	5	6	5	5
966	1534	11	15	13	12
1957	3043	22	30	26	24
2945	4555	33	46	39	36
3944	6056	44	60	52	47
9993	15007	109	153	131	117
19891	30109	219	305	261	233
29835	45165	328	457	392	350
39779	60221	437	609	522	466

Table 2. Computed Results.

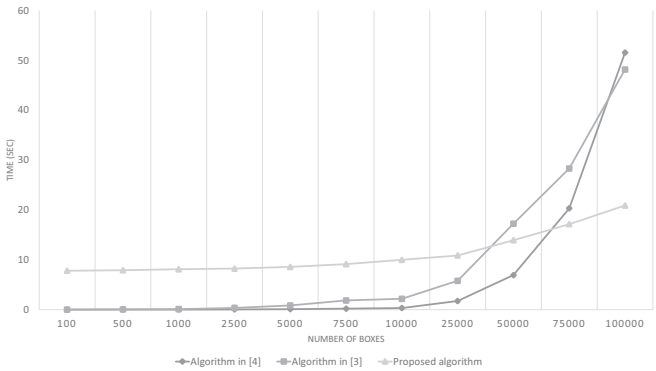

Figure 6. Time comparison of the results of the algorithms.

considerably produces results that satisfy the problem in a way such that the solutions obtained pack the given number of boxes in minimum number of bins, a number that is close to the lower bounds, which ultimately utilizes the bin's space, cost and completion time. The implemented algorithm showed better results for any given number of boxes when compared with the lower bounds and the two recently published algorithms found in [3] and [4].

A further study is intended as a future work regarding the selection of areas in the algorithm is needed to find if there are other selections for packing the boxes in a nearer perfect manner, along with balancing the weights of the bins. Also, proper adjustments to the algorithm can be done so that it is applicable for packing more than two types of boxes.

References

[1] Erick Dube, Leon R Kanavathy, and Phoenix Woodview. Optimizing three-dimensional bin packing through simulation. In *Sixth IASTED International Conference Modelling, Simulation, and Optimization*, 2006.

[2] Jens Egeblad and David Pisinger. Heuristic approaches for the two-and three-dimensional knapsack packing prob-

lem. *Computers & Operations Research*, 36(4):1026–1049, 2009.

[3] Jasim Hasan, Jihene Kaabi, and Youssef Harrath. Multi-objective 3d bin-packing problem. In *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, pages 1–5. IEEE, 2019.

[4] J Kaabi, Y Harrath, HE Bououdina, and AT Qasim. Toward smart logistics: A new algorithm for a multi-objective 3d bin packing problem. 2018.

[5] Tzuu-Hseng S Li, Chih-Yin Liu, Ping-Huan Kuo, Nien-Chu Fang, Cheng-Hui Li, Ching-Wen Cheng, Cheng-Ying Hsieh, Li-Fan Wu, Jie-Jhong Liang, and Chih-Yen Chen. A three-dimensional adaptive pso-based packing algorithm for an iot-based automated e-fulfillment packaging system. *IEEE Access*, 5:9188–9205, 2017.

[6] Flavio Keidi Miyazawa and Yoshiko Wakabayashi. Three-dimensional packings with rotations. *Computers & Operations Research*, 36(10):2801–2815, 2009.