ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/226249396

A New Heuristic Algorithm for the 3D Bin Packing Problem

Chapter · August 2008

DOI: 10.1007/978-1-4020-8735-6_64

CITATIONS
10

READS
10,348

3 authors:

Wissam FAYSAL Maarouf
American University of Science and Technology
1 PUBLICATION **10** CITATIONS
SEE PROFILE

Aziz M. Barbar
American University of Science and Technology
37 PUBLICATIONS **81** CITATIONS
SEE PROFILE

Michel Owayjan
American University of Science and Technology
61 PUBLICATIONS **217** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Parking Management System View project

All content following this page was uploaded by Aziz M. Barbar on 27 October 2015.

The user has requested enhancement of the downloaded file.

A New Heuristic Algorithm for the 3D Bin Packing Problem

Wissam F. Maarouf, Aziz M. Barbar, Michel J. Owayjan
American University of Science and Technology, Beirut, Lebanon
Department of Computer Science
wmaarouf@aust.edu.lb, abarbar@aust.edu.lb, mowayjan@aust.edu.lb

***Abstract*- 3D bin packing is a classical NP-hard (Nondeterministic Polynomial-time hard) problem where a set *N* of 3D boxes is to be packed in a minimum number of containers (bins). 3D bin packing is used in many industrial applications; hence computer scientists are challenged in designing practical and efficient approaches for the problem. This paper presents a new heuristic algorithm called Peak Filling Slice Push (PFSP) for 3D bin packing. The algorithm recursively divides the container into smaller slices and then fills each slice with boxes before pushing them to minimize the wasted space. Experimental results showed significant performance improvements over other current approaches.**

I. INTRODUCTION

Many industrial applications use packing. Examples include scheduling television programs, stacking cargo in a semi-truck, to loading airplanes and placing chips on circuit boards [1]. 3D bin packing is one type of packing problems where we are given a set *N* of 3D boxes and an unlimited number of containers (bins). The problem is to pack all the boxes into the minimum number of containers. 3D bin packing is a classical NP-hard (Nondeterministic Polynomial-time hard) problem; therefore exact solution cannot be achieved in polynomial time [1, 2]. Thus designing practical and efficient approaches to the problem is a challenge. The performance of 3D bin packing algorithm is largely affected by the strategy of packing boxes and the techniques used in minimizing wasted space. This paper presents a new heuristic algorithm for 3D bin packing. Experiments were conducted to test the performance of the algorithm and are compared with current tools like Robot packing [3].

II. LITERATURE REVIEW

Several methods have been used to solve 3D bin packing. Depending on the problem requirements, the techniques may attempt to minimize wasted space, minimize number of containers, maximize profit or stabilize the balance of containers. Being a combinatorial problem, 3D bin packing is usually solved using either optimization or heuristic algorithms. Optimization algorithms try to deliver an optimal solution [4], while heuristic algorithms deliver a good (acceptable) solution in a relatively acceptable time (that is linear time with respect to the input size) [4]. Wang presented an approach to two-dimensional rectangular packing by successively "gluing" together pairs of rectangles to produce a set of feasible sub-solutions [5].

For the non-rectangular packing, the geometric complexity of placing the pieces directly onto the stock sheet is generally prohibitive. Adamowicz and Albano and Israni and Sanders proposed an approach to first nest the pieces into regular modules [6, 7]. The wall-building approaches (George and Robinson, 1980, Bischoff and Marriott, 1990) are the common methods to deal with 3D cuboids packing problems [6]. Sections of a container across the full width and height are packed first. Identical items are grouped together to develop layers. An ordering of boxes based on decreasing volume, introduced by Gehring et al. is also used to develop layers [8].

Z. Dai and J. Cha [9] proposed a heuristic algorithm for the generation of 3D non-cuboids packing. An octree representation was used to approximate the geometry of the components. The packing algorithm is based on the idea of matching the octree nodes to identify the proper order and orientation of the components. The objects are packed into the container sequentially, depending on the number of items involved.

III. PROPOSED ALGORITHM

In 3D packing problem, we are given a set *N* of rectangular boxes and an unlimited number of containers. The task is to pack the *N* boxes into the minimum number of containers. Each box b_i has width w_i, length l_i, and height h_i. The containers are of the same size and have width *W*, length *L*, and height *H*. We present a heuristic algorithm called Peak Filling Slice Push (PFSP). This new approach benefits from the slicing mechanism introduced by Sweep [3]. The proposed algorithm has two main steps. First, the container is divided into slices having same height and width as the container. Fig. 1 shows a graphical explanation of the slicing mechanism. Each slice is filled using Peak Filling technique. Second, the filled slices are pushed in order to compress the boxes and minimize the wasted space. For efficiency purposes, the boxes are sorted in decreasing order by height, width, length. Slicing the container has a considerable influence on reducing packing time. It also makes the algorithm easily parallelizable.

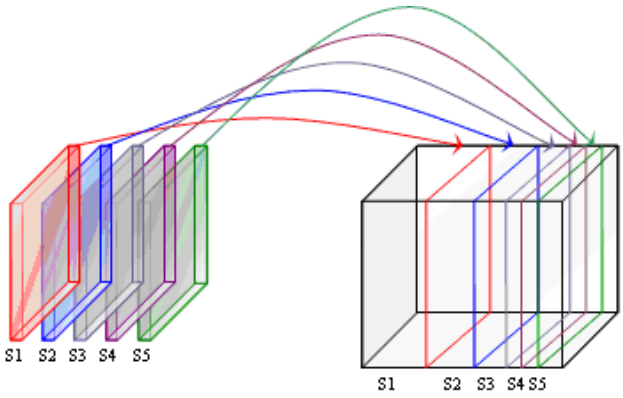

Fig. 1. Slicing mechanism.

Peak filling method is a recursive divide-and-conquer procedure. Each time a box is placed in a slice, a new sub-slice is created on the top of the placed box as shown in Fig. 2. When no more sub-slices can be created, the algorithm starts backwards and fills the left sub-slices until no usable space is left. The algorithm works by placing bigger boxes at the bottom of the container. In upper direction filling, the placed boxes are smaller or of same size as the initial box placed at the bottom. In backtracking, the chosen boxes are of smallest size.

For each slice, a stack is used to keep track of the remaining unfilled sub-slices. A reduction from 3D to 2D is done as illustrated in Fig. 3. Whenever a sub-slice is created, the dimensions of the bottom surface as well as the coordinates of the upper left corner are pushed on the stack. This is essential for the backtracking step. When the top of the slice is reached, backtracking step pops from the stack and fills the remaining unfilled space using smaller boxes. The backtracking step stops when the stack is empty meaning that there is no usable space left in the slice.

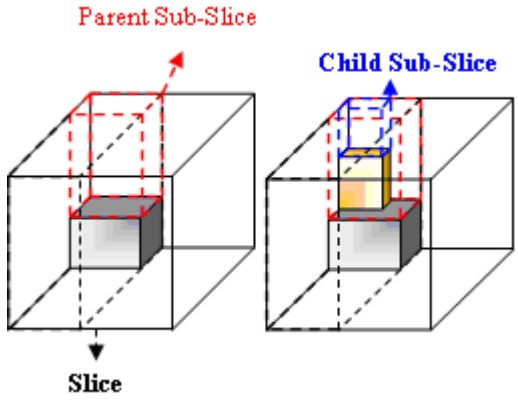

Fig. 2. Creating new sub-slice on the surface of the placed box.

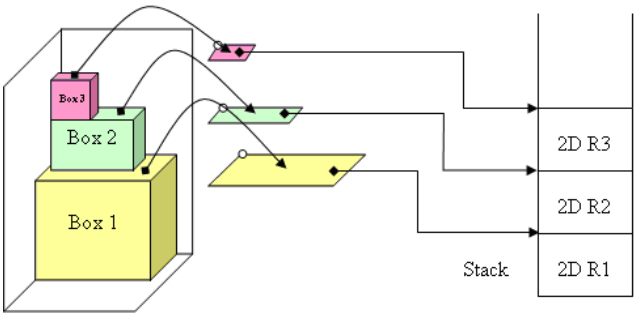

Fig. 3. Reduction from 3D to 2D.

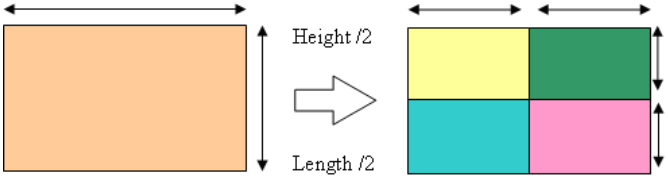

Fig. 4. Division of certain sub-slice.

Filling each slice (or sub-slice) is done by dividing the slice into four rectangular slices and using different combinations of the resulting sub-slices to determine the best arrangements of boxes inside the given slice. We have chosen to divide the slices with the ratio $r = 0.5$. Thus applying the ratio, we divide the slice into four equal sub-slices as shown in Fig. 4, even though different divisions can be used for the slice.

The algorithm then considers different combinations of resulting rectangles and tries to fill them with the available boxes. The combinations should result in rectangular shapes only thus some invalid combinations will be removed. The combination that gives the minimum wasted area is selected. Fig. 5 shows the possible combinations for the sub-rectangles. The aim of the algorithm is to reduce internal fragments thus minimizing space. Whenever a new slice is filled, a "Push" method is called. This method pushes the boxes of the new slice into the old slice without overlapping. Both slices are then combined into one larger slice. Fig. 6 shows the push mechanism.

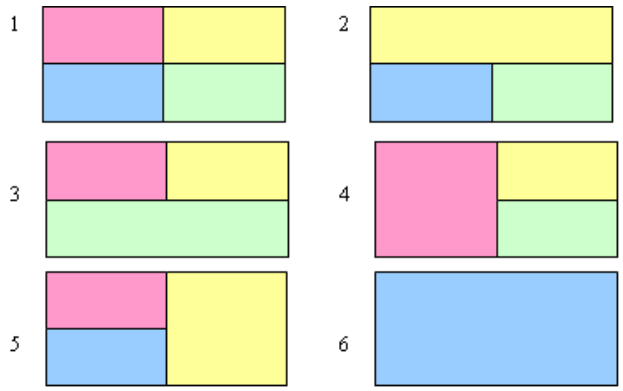

Fig. 5. Possible combinations.

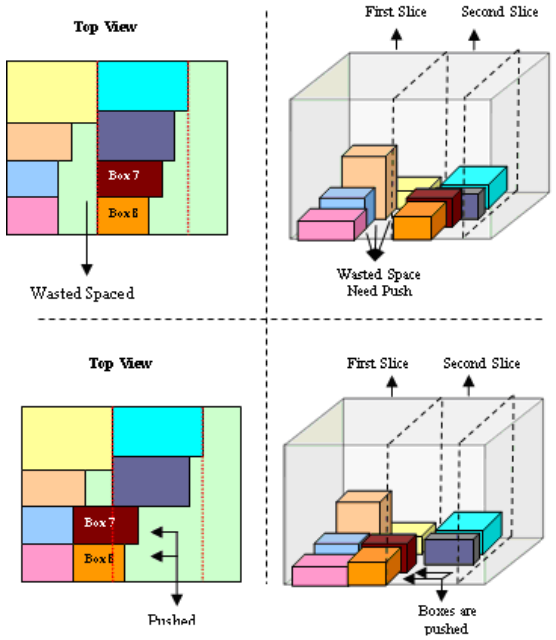

Fig. 6. Push mechanism.

The pseudo code of the algorithm can be summarized as the following:

```
Input:   Boxes Array = (Box 1, Box 2, …, Box n)
         Container = (x, y, z)
Output:  Packed Boxes Array = (Box i, Box j, … , Box m)

While (Container.FULL = False) {
         Stack.Clear()
         Temp Box = GetFirstAvailableBox(Box Array)
         Slice = CreatSlice(Temp Box)
         If (Slice = Null)   Container.FULL = True
         Rectangle = GetRectangle(Slice)
         Stack.Push (Rectangle)
         Packed Boxes = PeakFilling(Box Array, Container,
         Slice, Stack)
}
Packed Boxes = SlicePush( Packed Boxes)
```

The complexity of this algorithm is in O (N^3). The algorithm has a main "while loop" and two nested "for loops" in the peaks filling function. The result is three nested loops. It also has 2 nested loops in the function slice push. After adding them all together it would generate the following linear formula:

$$SUM = N^3+N^2+C$$

The C is a constant that represent the total additional operations. The algorithm must also take into consideration the sorting time. The sum would be:

$$SUM = N*\log(N)+N^3+N^2+C$$

The calculation of the complexity is the following:

$$O(SUM) =$$
$$O(N*\log(N)+N^3+N^2+C)=$$
$$O(N^3+N^2+C)=$$
$$O(N^3 +C)=$$
$$O(N^3)$$

IV. EXPERIMENTAL RESULTS

The algorithm has been implemented using Visual Basic programming language. It is compiled with Microsoft Visual Studio .NET v2005 Professional Edition (Microsoft Corporation, Redmond, WA, USA) and Microsoft framework 2.0 (Microsoft Corporation, Redmond, WA, USA). The graphical part of the implementation used DirectX 9.0c. The algorithm was tested on Intel Pentium® 4 CPU 3.00 GHz, 32 KB L2, 512 MB of RAM, and PM800 Motherboard. The operating system that has been used was Windows XP Professional SP2 (Microsoft Corporation, Redmond, WA, USA). All the results have been calculated in time. The operating system and other processes time has been calculated within the testing time. The sorting time is not calculated within the experiments. The experiments used a sorted box ready to be packed

The proposed algorithm was tested using four test bunches. Table 1 summarizes the information about the test bunches. The test bunches ran on a Pentium 4 processor having 256 MB of RAM. The dimensions of the boxes are chosen randomly. A box whose dimensions are larger than the container is discarded. The container dimensions can be specified by the user.

TABLE 1
TEST SPECIFICATIONS

Test Bunch	Number of Boxes	Dimension of Container (H, W, L)
1	50	(10, 10, 15)
2	100	(10, 10, 15)
3	150	(10, 10, 15)
4	200	(10, 10, 15)

Table 2 shows the results obtained. On average, the bins are filled up to 85%, which is a satisfying result. The proposed algorithm runs in an acceptable time; less than 0.5 seconds with large input sets like 100 boxes. The results show an improvement over current tools like Robot. The percentage of filled volume from each container is shown in Table 3.

Tables 3 and 4 show best and worst case scenarios with respect to solution times (in seconds) for the general approach and Robot tool. It is obvious that our approach outperforms these two approaches by a large factor. Packing 50 boxes require at least 12.97 seconds in general approach and 11.09 in Robot, while in our approach, it takes about 0.019 seconds. Also both approaches do not pack more than 50 boxes, where as PFSP can pack up to 200 box in less than 0.5 sec.

TABLE 2
RESULTS OBTAINED

Test Bunch	Average Wasted Space	Average Time (sec)
1	16.08 %	0.019063
2	15.95 %	0.034375
3	15.14 %	0.037708
4	15.38 %	0.046953

TABLE 3
SOLUTION TIMES IN SECONDS FOR THE GENERAL APPROACH

Number of Boxes	Best Case Scenario (sec)	Worst Case Scenario (sec)
10	0.01	0.07
20	0.01	0.21
30	0.08	57.14
40	0.10	5247.60
50	12.97	55402.25

TABLE 4
SOLUTION TIMES IN SECONDS FOR ROBOT APPROACH

Number of Boxes	Best Case Scenario (sec)	Worst Case Scenario (sec)
10	0.01	0.07
20	0.01	0.21
30	0.07	37.54
40	0.11	50593.91
50	11.09	30719.51

TABLE 5
PERCENTAGE OF USED VOLUME IN CONTAINER

Number of Boxes	Best Case Scenario	Worst Case Scenario	Average
50	96 %	63 %	83.92 %
100	97 %	63 %	84.05 %
150	98 %	62 %	84.86 %
200	98 %	63 %	84.625 %

In the best case scenario, the container is filled up to 96% which is a very satisfying result according to industrial demands. Worst case scenario fills more than half of the container while on average the container is filled up to 84% which is an acceptable trade off between time and performance.

V. DISCUSSION AND CONCLUSIONS

Peak Filling Slice Push (PFSP), a new heuristic algorithm, is presented in this paper to solve 3D bin packing problem that is faced in many industrial applications. The implementation of PFSP using the Visual Basic programming language shows improvements in both performance and average time over other methods currently in use by the industry. PFSP presents a potential for saving time and money for numerous industries. As any new algorithm, PFSP has some limitations. PFSP can not load balanced containers like ships. It does not have the weight factor included. It also limited to use in a large cargo, the opportunity of using it is narrowed. On the other hand, it is practical in filling small volumes with different boxes' size of a cargo. Future work will include improvements to the algorithm like the ability to rotate the boxes for further efficiency, adding different ratios to divide a sub-slice and balanced packing. These improvements are expected to increase the performance and efficiency of the algorithm.

REFERENCES

[1] J. Haromkovic, *Algorithmics for Hard Problems*, 2nd ed., 2003.

[2] K. Kleinberg and E. Tardos, *Algorithm Design*, Addison Wesley, 2006.

[3] E.D. Boef, J. Korst, S. Martello, D. Pisinger, and D. Vigo, "A Note on Robot-Packable and Orthogonal variants of the Three-Dimensional Bin Packing Problem."

[4] S. Sweep, "Three Dimensional Bin-Packing Issues and Solutions," University of Minnesota, 2003.

[5] S. Wang, Singing voice: bright timbre, singer's formant, and larynx positions, in A. Askenfelt, S. Felicetti, E. Jansson, and J. Sundberg, eds. Proc of Stockholm Music Acoustics Conference 1983 (SMAC 83). Stockholm: Publications issued by the Royal Swedish Academy of Music 46:1, 313-322.

[6] M. Adamowicz, and A. Albano, *Nesting Two-Dimensional Shapes in Rectangular Modules*, Computer Aided Design, vol. 8, 1976, pp 27-33.

[7] S. Israni, and J.L. Sanders, "Performance Testing of Rectangular Parts-Nesting Heuristics," *International Journal of Production Research*, vol. 23, 1985, pp 437-456.

[8] H. Gehring, K. Menschner, and M. Meyer, "A Computer-Based Heuristic for Packing Pooled Shipment Containers," *European Journal of Operational Research*, vol. 44 (2), 1990, pp 277- 289.

[9] Z. Dai, and J. Cha, "A Hybrid Approach of Heuristic and Neural Network for Packing Problems," *Advances in Design Automation 1994: Proceedings of the 20th ASME Design Automation Conference*, vol. 2, 1994, pp 117-123.

View publication stats