

# OPTIMIZING THREE-DIMENSIONAL BIN PACKING THROUGH SIMULATION

Erick Dube  
School of Computer Science  
University of KwaZulu-Natal  
P. Bag X54001, Durban 4000  
South Africa  
[dubee2@ukzn.ac.za](mailto:dubee2@ukzn.ac.za)

Leon R. Kanavathy  
54 Cherrywood Avenue,  
Woodview, Phoenix,  
Durban, 4068  
South Africa  
[LeonK@infowave.co.za](mailto:LeonK@infowave.co.za)

## ABSTRACT

The problem addressed in this paper is that of orthogonally packing a given set of rectangular-shaped items into a minimum number of three-dimensional rectangular bins. We harness the computing power of the modern day computers to solve this NP-Hard problem that would be otherwise practically extremely difficult. The software tool that we develop using both heuristics and some knapsack problem approach, presents the solutions as a 3D graphical representation of the solution space. The visual and interactive ability of the simulation model provides a disciplined approach to solving the 3D Bin Packing Problem.

## KEY WORDS

Simulation, Optimization, Bin packing, Best fit, First fit, Rotation.

## 1. Introduction

In this paper, we describe a simulation approach to characterizing the feasible packaging and construction of optimal solutions. The complexity of finding optimal solutions for the Three Dimensional Bin Packing Problem is compounded by the difficulty of giving a useful problem formulation.

To formulate the problem we will consider each item  $i$  in the finite set  $S$  to have three dimensions  $w_i$ ,  $h_i$  and  $d_i$ . Each identical bin  $b$  has dimensions  $W$ ,  $H$  and  $D$ . The items and bins are rectangular boxes and the three dimensions correspond to the width, height and depth values. To make the solution more distinct from previous work the items are allowed to rotate orthogonally. Rotating an item simply means swapping its width ( $w_i$ ), height ( $h_i$ ) and depth ( $d_i$ ) values around in a defined ordered manner (Table 1).

Each item-box has 6 rectangular facets, but there are only 3 distinct facets because "opposite" facets are identical. Each of the three facets can be rotated orthogonally to obtain a new configuration of the box. Thus each item can have 6 different rotation configurations.

Linear programming approaches have been used for single dimensional problems.

In some approaches Evolutionary algorithms have also been used instead of heuristics algorithms [1,3]. Due to their ability to search large spaces, evolutionary algorithms could have been a suitable method to finding a solution (in this case, the solution space of the Bin packing problem), but evolutionary algorithms have a few shortcomings:

There is little continuity between solution and problem i.e. if you change the problem parameters a little, then the solution changes considerably.

Heuristics are more comprehensive and some offer worst case performances.

## 2. Problem formulation.

To find the solution for a bin  $b$  we assume without loss of generality that  $\sum_{i \in b} w_i \leq W$ ,  $\sum_{i \in b} h_i \leq H$  and

$\sum_{i \in b} d_i \leq D$ . As such it is correct to conclude that

$\sum_{i \in b} w_i \cdot h_i \cdot d_i \leq W \cdot H \cdot D$ . So for each bin  $b$  we

intend to minimize the wasted volume given by

$$W \cdot H \cdot D - \sum_{i \in b} w_i \cdot h_i \cdot d_i$$

Bin packing being an NP-Hard problem, suggests that an exhaustive search for the optimal solution is in general computationally intractable, and also that there is thus no known real computationally feasible optimal solution method for the problem. So other means to obtain a solution have to be found. Most popular are heuristic solution methods:

Items are packed one at a time with no backtracking (once an item is packed it is not repacked). The choice of an item to be packed can be done by using formal logic derived from one of the following packing algorithms.

First Fit [2,4]

Packs unassigned item into first bin that has enough space. If there is no such bin, assign item into new a bin.

### First Fit Decreasing

Almost the same as First Fit except that the items are first sorted in decreasing order before being packed.

### Last Fit

Packs unassigned item into last bin with enough space. Searching is similar to First Fit but in the reverse order of bins. If there is no such bin, assign item into new bin.

### Best Fit

The Best Fit algorithm packs an item in a bin, which is the fullest among those bins in which the item fits.

More specifically:

Items are packed one at a time in given order.

To determine the bin for an item, first determine set  $B$  of containers into which the item fits.

If  $B$  is empty, then start a new bin and put the item into this new bin.

Otherwise, pack the item into the bin of  $B$  that has least available capacity.

## 3. Solution Specification

The developed system uses a heuristic approach to perform the core of the bin packing.

The use of these heuristic approximate algorithms in the system to solve the bin packing problem:

- i. guarantees a solution to the problem,
- ii. obtains a solution in a reasonable time (i.e. solution is computationally feasible to obtain),
- iii. allows general data input,
- iv. provides continuity between the solution and the problem.

The above points provide solid reasoning as to why a heuristic approach was chosen because any approach which fails to satisfy any of the above conditions would not completely meet user requirements and would hence not be of any use.

It is obvious that failure to satisfy (i) and (ii) would be unsatisfactory. If the system does not satisfy (iii) then it would lose its generality and flexibility. Condition (iii) is also of importance because bins (or containers) need to be packed with items (or cargo) of *different dimensions*. Also failure to satisfy (iv) would make it difficult for users to retrieve data and test alternative solutions during the process of problem solving. Thus, failure to satisfy (iv) would mean that the system does not readily support this feature.

The two primary heuristic bin packing algorithms that were used in the system were the *First Fit Decreasing* and the *Best Fit*. They were chosen over other heuristic algorithms because they have a faster running time, as well as produce solutions that are much closer to the optimal solution than most other heuristic algorithms.

## 3.1 Analysis of the algorithms

Let  $M$  be the optimal number of bins required to pack a set of  $n$  items.

Let  $m_f$  be the number bins required when First Fit Decreasing is used to pack the items.

Let  $m_b$  be the number bins required when Best Fit is used to pack the items.

$$m_f \leq M + \left\lceil \frac{(M-1)}{3} \right\rceil$$

Then it can be shown [2] that

$$m_b \leq \left\lceil \frac{17M}{10} \right\rceil$$

and also that

Now:

$$M + \frac{(M-1)}{3} \approx \frac{4M}{3} \quad \text{for } M \text{ sufficiently large,}$$

$$\text{and thus } \frac{17M}{10} = 1\frac{7}{10}M > 1\frac{1}{3}M = \frac{4M}{3} \quad \text{for large } M.$$

Hence it appears that *Best Fit* is more likely to produce a solution closer to the optimal solution than is *First Fit Decreasing*, but seeing as these values on the upper bound for the number of bins needed are relatively close to each other, it is worth using both algorithms.

We also observe that:

$$\left[ \left( \sum_{i \in S} s_i \right) / C \right] \leq M \leq m_f \leq M + \left\lceil \frac{(M-1)}{3} \right\rceil$$

and

$$\left[ \left( \sum_{i \in S} s_i \right) / C \right] \leq M \leq m_b \leq \left\lceil \frac{17M}{10} \right\rceil$$

where  $C$  is the capacity of a bin in single units, area or volume depending on the current bin packing case under consideration.

The running time for *Best Fit* is  $O(n \log n)$  and for *First Fit Decreasing* it is  $O(n \log n)$  excluding the running time for sorting.

## 4. Software Specification

We have three directions in which to pack the items, width direction, height direction and depth direction.

As previously explained each Item has six rotation types . Consider an item:

The six rotation types can be obtained by rotating about the  $x$ ,  $y$  and/or  $z$  axis as shown in Table 1:

The bins are packed one at a time and the algorithms use a series of pivot points at which to pack the item.



```

        left corner of binItem
        break
    case 2 : Choose (pivotX, pivovY,
pivotZ ) as the back Upper
        left corner of binItem
        break
    }

    if (currentItem can be packed
in currentBin at
        position(pivotX, pivotY
,pivotZ ) ) then
    {
        Pack currentItem into
currentBin at position
        (pivotX, pivotY ,pivotZ).
        fitted=true
    }
    else
    { // try rotating item
do
        Rotate currenItem
while (currentItem cannot be
packed in currentBin at

position(pivotX,pivotY) )
        and (not all
rotations for currentItem
checked)

    if (currentItem can be packed
in currentBin at
        position(pivotX, pivotY ,
pivotZ) ) then
    {
        Pack currentItem into
currentBin at position
        (pivotX, pivotY ,pivotZ).
        fitted=true

    }else
        Restore currentItem to
its original rotation type
    }

    if (not fitted) then
        Add currentItem to the list
notPacked

    }
}
}
}

} while notPacked has at least one
Item in it (*i.e. notPacked is

```

non-empty \*)

#### 4.1.2 Worst Case Running Time

In the worst case we see that the **do-while** loop referenced by **a** above will run at most ( $n$ ) times .

Loop **b** will run at most ( $n-2$ ) times.

**c** will run 3 times.

**d** will run at most ( $n-1$ ) times because the number of items in a bin could be ( $n-1$ ).

The rotations carried out throughout the algorithm can be at most 6, so this does not significantly influence our running time.

So we have :  $O(\text{Best Fit}) = n * (n-2) * 3 * (n-1)$   
 $= O(n^3)$

Thus in the worst case the algorithm produces a solution in polynomial time.

#### 4.1.3 Best Case Running Time

The Best case excluding the trivial cases is when

All the items fit into one Bin :

**a** will run 1 time.

**b** will run ( $n-2$ ) times.

**c** will run 3 times.

**d** will run ( $n-1$ ) times.

So we have :  $O(\text{Best Fit}) = 1 * (n-2) * 3 * (n-1)$   
 $= O(n^2)$

Or when each item is packed into its own bin

**a** will run  $n$  times.

**b** will run ( $n-2$ ) times.

**c** will run 3 times.

**d** will run 1 time.

So we have :  $O(\text{Best Fit}) = n * (n-2) * 3 * 1$   
 $= O(n^2)$

Thus in the best case, the performance is  $O(n^2)$ .

#### 4.2 3D First Fit Decreasing Algorithm

To pack an item one has to first decide on a packing direction. The longest side of the bin corresponds to the packing direction. Then rotate each item such that the longest side of this item is the side which is the packing direction, i.e. if we are packing by width then we want the longest side of the item to be the item's width, so for example if the packing direction is by width and the current height of the item is longer than its width, then rotate the item. If after performing the rotation(s), the item cannot fit into the bin (i.e. one or more of the dimensions of the items exceeds the bin's corresponding dimension) then we rotate the item until the *second* longest side of this item is the side which corresponds to the packing direction. If after performing the rotation(s), the item cannot fit into the bin then we rotate the item until the *third* longest side of this item is the side which corresponds to the packing direction. Next sort the items in decreasing order of width, height or depth depending on packing direction.

## 5.0 Simulation Model and Graphical display

The simulation model was implemented in C++ using an object oriented approach. The items and bins are represented by objects and a single object was used to represent the bin-packer itself. The application was developed on a Pentium II PC environment and gives acceptable response times. For portability sake, the user

interface (Fig 2) is designed such that the textual solution to the problem is completely separated from the graphical solution (Fig 3) to the problem. The textual solution contains all the bins needed, the items contained in each bin and the positions of the items in each bin.

Item Data				Solution			
Item	Width	Height	Depth	Bin Number	Item : (x,y,z) :{RotType}	Item : (x,y,z) :{RotType}	Item : (x,y,z) :{RotType}
1	50	20	20	1	4 : (0, 0, 20) :{4}	12 : (35, 0, 20) :{5}	81 : (45, 0, 20) :{5}
2	30	30	30	2	13 : (20, 0, 0) :{0}	29 : (40, 0, 0) :{1}	40 : (40, 0, 10) :{1}
3	30	10	10				
4	35	15	10				
5	5	5	5				
6	20	20	20				
7	20	10	10				
8	20	15	10				
9	25	20	15				
10	15	10	10				

Total Items 109      Width of Bin 50      Height of Bin 50      Depth of Bin 50      No. of Bins 2

Total Bin Space = 250000  
 Total Bin Space Used = 247375 = 98.95%  
 Total Wasted Space = 2625 = 1.049%  
 First Fit Decreasing algorithm used.

Editable input for dimensions of items

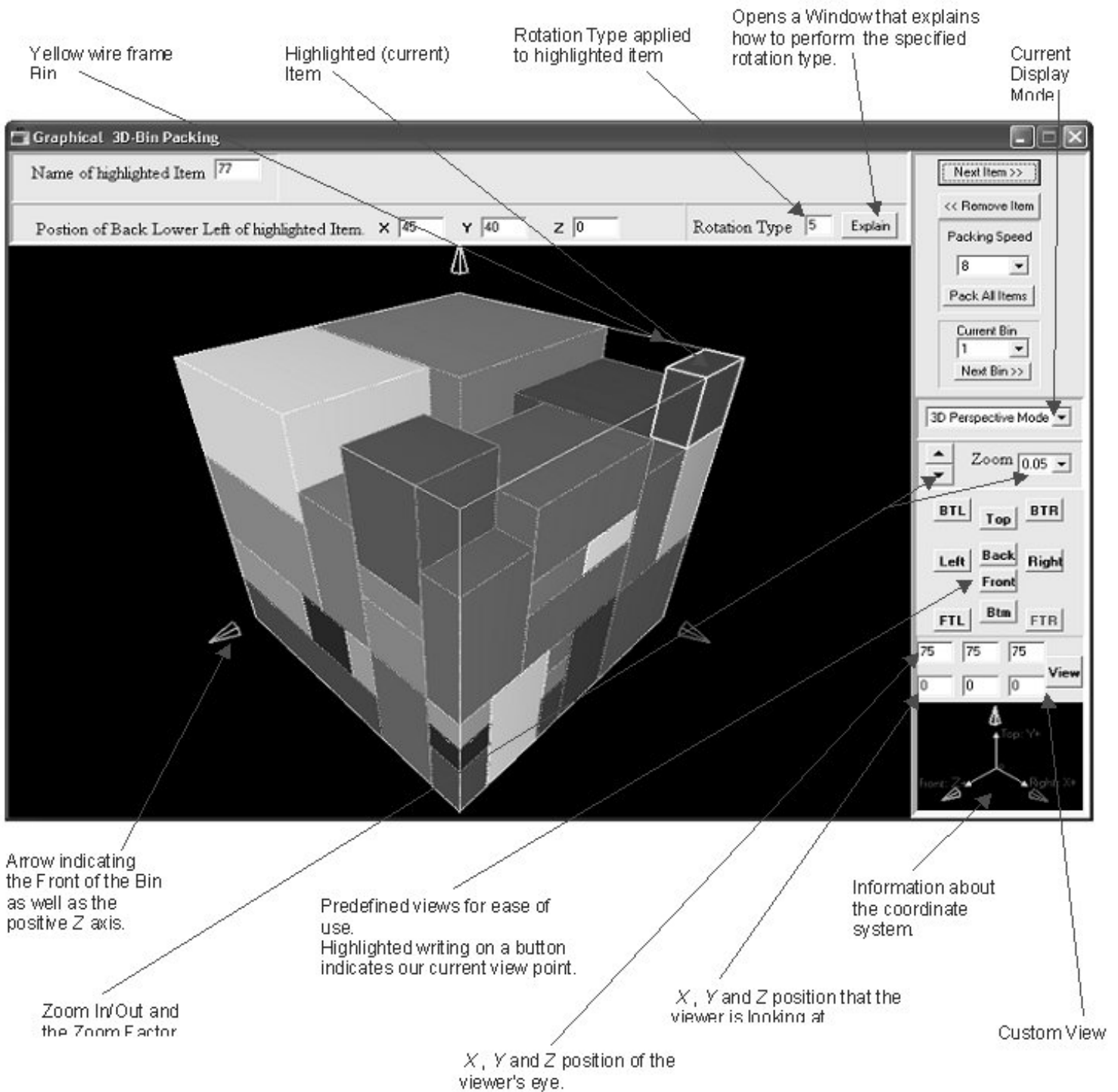
Statistical Solution Summary output

Indicates the rotation type that must be applied to the item. (see Table 1)

x, y and z position of the back lower left corner of the item positioned in the bin

Entire Textual solution output

Fig 2. Snapshot of User Interface



**Fig 3:** Graphical representation of a Packed Bin

The Graphical solution display (Fig 3) was implemented using OpenGL under a windows operating system environment. The graphical display provides users amongst other utilities the *Custom View* button that enables the user to obtain their own view point. This view point is defined by changing the X , Y and Z eye position and/or changing the X , Y and Z 'look-at' position. The

eye and look-at positions also provide information to the user about the predefined views.

The statistical solution summary data (Fig 2) provides information about both the problem and the solution undertaken. The most important piece of information of the solution here is the total number of bins used. The

wasted space and used space do not reflect on the optimality of the solution but merely inform the user that there is more space available to pack other items, if desired. This is best illustrated by the following example. Suppose all the items except one item could be packed into just one bin and the remaining item, suppose which is relatively small in volume to the bin is packed into a second bin. This item will then be the only item packed into the second bin and thus there will be a lot of wasted space left over in the second bin.

## 6.0 Conclusion

Bin packing is a very appealing mathematical model problem, yet work on this problem is surprisingly recent. In this paper, we considered the implementation of the optimization of packing 3-D boxes into a finite set of bins and demonstrated that the program will find a solution within reasonable time. The major set backs of most other implementations is that of failing to converge to a solution and thus execute “indefinitely”. Our careful design also brings in the visualisation of the solution, that is, the exact location and orientation of an item in a bin is known.

Our future work will be to use the application to determine the set of possible sizes of boxes that can be used by shipping companies so as to make better use of existing loading techniques.

## 7.0 References

- [1]Frederick Ducatelle, John Levine. Ant Colony Optimisation for Bin Packing and Cutting Stock Problems, *Proceedings of the UK Workshop on Computational Intelligence*, 2001, Edinburgh.
- [2]Emanuel Falkenauer, A hybrid grouping genetic algorithm for bin packing, *Journal of Heuristics*, 1996.
- [3]Fekete, S. P., J. Schepers, A new exact algorithm for general orthogonal 3d-dimensional knapsack problems, *Lecture Notes in Computer Science*, 1997.
- [4]Leon Kos, Joze Duhovnik, Rod Cutting Optimization with Store Utilization, *International design conference – DESIGN, Dubrovnik*, 2000.
- [5]Silvano Martello, David Pisinger and Daniel Vigo, The three dimensional bin packing problem, *Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA*, 2000.
- [6] Peter Ross, Sonia Schulenburg, Hyper-heuristics: learning to combine simple heuristics in bin-packing problem, *Discrete Applied mathematics ACM*, 2000.