Three Dimensional Bin-Packing Issues and Solutions

Seth Sweep
University of Minnesota, Morris
swee0119@mrs.umn.edu

Abstract

This paper reviews the three-dimensional version of the classic NP-hard bin-packing, optimization problem along with its theoretical relevance and practical importance. Also, new approximation solutions are introduced based on prior research.

The three-dimensional bin-packing optimization problem concerns placing box-shaped objects of arbitrary size and number into a box-shaped, three-dimensional space efficiently. Approximation algorithms become a guide that attempts to place objects in the least amount of space and time. As a NP-hard problem, three-dimensional bin-packing holds much academic interest to computer scientists. However, this problem also has relevance in industrial settings such as shipping cargo and efficiently designing machinery with replaceable parts, such as automobiles. Industry relies on algorithms to provide good solutions to versions of this problem.

This research project adapted common approaches to one-dimensional bin-packing, such as next fit and best fit, to three dimensions. Adaptation of these algorithms is possible by dividing the space of a bin. Then, by paying attention strictly to the width of each object, the one-dimensional approaches attempt to efficiently pack into the divided compartments of the original bin. Through focusing entirely on width, these divisions effectively become multiple one-dimensional bins.

However, entirely disregarding the volume of the bin by ignoring height and depth may generate inefficient packings. As this paper details in more depth, generally cubical objects may pack well but exceptionally high or long objects leave large gaps of empty space unpacked. However, two modifications of the one-dimensional algorithms partially alleviate this problem. These modifications include sorting the objects and packing around the higher and longer objects.

In the course of this research, six one-dimensional approaches—next fit, first fit, best fit, worst fit, last fit, and almost-worst fit—were implemented as three-dimensional bin-packing algorithms. These algorithms were tested against random sets of objects. Also, each algorithm's speed and packing efficiency was compared against each other and a persistent random algorithm.

Introduction

Since the advent of computer science over thirty years ago, bin-packing remains one of the classic difficult problems today. Computer scientists and discrete mathematicians have analyzed and studied this computational puzzle for decades [1], yet none have obtained an algorithm which derives the optimal solution in reasonable amount of time. However, fortunately, no one has yet disproved the possibility of such a solution. Theorists place bin-packing problems and other potentially "unsolvable" problems in a class of mathematical problems known as NP-hard and NP-complete. At this time, no optimal solutions to the bin-packing problem may be derived by a computer without deriving nearly every possible solution. In other words, finding a perfect solution to one non-trivial instance of the bin-packing problem with even the most powerful computer may take months or years.

However, while theorists study, ponder, and test, real-world industry relies on solutions to these special problems, even imperfect solutions. Industry uses bin-packing for everything from scheduling television programming to stacking cargo in a semi-truck to designing automobiles and airplanes. In many cases, if a computer can find a near-perfect solution in a reasonable amount of time, that solution will suffice for industrial use. Therefore, theorists and scientists developed approximation algorithms to bin-packing problems.

This analysis will specifically show new, original approximation solutions to the three-dimensional version of bin-packing. Bin-packing in three dimensions, or box packing, only received significant attention in computer science during the past decade [2] despite its great importance to industry and computer science.

To better understand these new approximation algorithms, this paper will first define the three dimensional bin-packing problem and describe its practical importance. Then this paper will examine in-depth six new approximation algorithms derived from original research. Finally, these algorithms's packing and computational efficiency against a random algorithm will be shown.

The Classic Bin-Packing Problem

In the classic, one dimensional bin-packing problem, a finite collection of objects with varying width is packed into one or more bins or baskets. Each bin can hold any subset of the collection of objects, but cannot exceed its capacity [3]. Each object is packed into the bin or bins so that the least amount of space remains. This problem may be thought of as either a decision problem or an optimization problem. In the decision problem, a determination must be made: do all of the objects fit into the bins? In other words, the decision problem states that enough free space exists to hold the objects or not. Alternatively, the optimization problem attempts to minimize the number of bins wasted or minimize the amount of wasted bin capacity. This way of framing the problem attempts to compress the bins most efficiently [2]. As the decision problem has a boolean answer, the answer is less complex. Conversely, the optimization problem has a numeric answer, so more complexity exists. Mathematicians and computer scientists consider the decision problem NP-complete, while the optimization problem—which this paper focuses on—is considered NP-hard.

Bin-Packing In Two Dimensions

Expanding this bin-packing problem into to two dimensions adds further complexity to this problem. Each object in one dimensional bin-packing had fixed width and variable height or variable width and fixed height. In the two dimensional version, both sides vary. The two dimensional bin-packing problem may be thought of as placing rectangles on a flat surface. Rectangle packing equates to efficiently fitting varying sized flat rectangles into a larger flat rectangular space most efficiently. The classic video game *Tetris* relates to the problem of rectangle bin-packing. However, unlike *Tetris*, classic two dimensional packing requires each object to orthogonal, so objects may not be rotated.

Bin-Packing in Three Dimensions

The three-dimensional bin-packing problem retains the difficulty of lesser dimensional bin-packing problems, but holds unique and important applications. As one would expect, each object and bin exists in three dimensions. These objects and bins represent triplets containing three values: width, length, and height. Each box should fit into a bin or bins most efficiently. Like 2D bin-packing, each box must retain stay orthogonal, or maintain its orientation in the packing. If two dimensional bin-packing equates to rectangle-to-floorplan packing, three dimensional bin-packing equates to box-to-room packing.

3D bin-packing may involve a single bin or multiple bins. The singular bin-packing problem involves only one bin with either definite or infinite volume. Bins with infinite volume are defined with finite length and width, but with height extending to infinity. This allows packing solutions to pack until the set of boxes are exhausted. Solutions dealing with infinitely sized bins generally care most compressing objects effectively. Another way to approach this problem is by considering multiple bins. Each bin has definite volume. In this way, if the volume of the objects exceeds the volume of the room, an algorithm must make choices of which boxes to include in the packing and which to throw away. This approach is good for deterministic approaches to the box packing problem. These problems ask: “Do the bins hold enough volume to fit these objects?”

Like all bin-packing problems, extra constraints may be added to the problem to create a more real-world-like problem. In warehouses, gravity is a constraint so all boxes must rest on the floor, box, or other platform (such as a pallet) below it. Another constraint may be weight distribution. Real objects have mass and weight, so a packing may want equal weight distribution in the storage space. Placing all the weight in one location could cause a boat to tip or a truck to become unstable. Also, we may not want a large, heavy object—such as a 2000 kg automobile in a crate—on top of a smaller, lighter object—such as a 1 kg shoebox [4]. Time can be another deadline; perhaps we want a schedule that will quickly allow us to pack a room. Perhaps certain classes of objects require priority. Clearly, the constraints are endless. The classic-bin packing problem does not take these constraints into account, because each of these more complex problems may be simplified.

Applications

The three dimensional bin-packing problem holds importance to many industrial situations. Shipping and moving industries, architecture, engineering, and design are all areas where three dimensional bin-packing could apply.

Recently, the Institute for Algorithms and Scientific Computing in Germany used three-dimensional packing for research in molecular biology and chemistry and also with automobile design for manufacturer Mercedes-Benz [5]. In the latter case, three dimensional approximation packing algorithms were used to efficiently place automobile components in a new design of vehicle.

In New Zealand a manufacturer of electrical cable requested a way to efficiently pack drums and pallets of cable into standard shipping containers for transport overseas. In the past, scheduling was done manually which caused for some large orders to be packed inefficiently, losing the company time and money as some orders would need to be sent in two shipments. The University of Canterbury discovered an approximation solution that minimized the number of special containers used. However, this problem presented several constraints which included drums needing to be modeled as cylindrical and stacked so they would not roll around during transport and be able to be easily taken on and off a ship by a forklift. However, despite these constraints, computer scientists developed an effective heuristic for this special case [4].

Approximation Algorithms through the Level-Slice Approach

Often times the best solutions for difficult problems equate to the simplest solutions. These simple algorithms—adapted from popular approaches to the one-dimensional bin-packing problem—provide reasonable approximation solutions in polynomial time. The Next Fit and First Fit algorithms were inspired by research done by Corcoran and Wainwright [2]. The remaining four adaptation algorithms, a random algorithm, and the discussion of the "level-slice" problem and solutions are the original work of the author.

These algorithms deal chiefly with the singular bin-packing problem, where the bin has a definite volume. Also, each packing algorithm deals with the optimization bin-packing problem. In other words, for a set of boxes, each packing algorithm returns a packing that tries to come as close to the optimal solution as possible.

The Level-Slice Approach

For these algorithms, the packing space equates to a singular, finite volume bin. Each side of the bin has a specific length and objects that do not fit into the bin are discarded. All six algorithms use a similar approach for placing boxes in the bin. These approaches use a pattern to guide the placement of boxes. This approach divides the bin into several smaller bins. With the large bin represented as several smaller bins that only consider one side of the object—one-dimensional bin-packing algorithms may be applied. However, the other sides of the bin must be accounted for. This is done through a *level-slice* approach.

The level-slice approach divides the bins both horizontally and vertically. The horizontal divisions run along the height of the bin and the vertical divisions run along the length of the bin. These horizontal enclosures are called "slices" and the vertical enclosures are named "levels." These "levels" are analogous to floors of a tall building while "slices" are analogous to the slices of a bread loaf. Where these levels and slices intersect are "level-slices."

Figure 1A: A Three Dimensional Bin. The vertically shaded area is a "slice"; the horizontally shaded area is a "level." Where a level and slice intersect is considered a "level-slice."

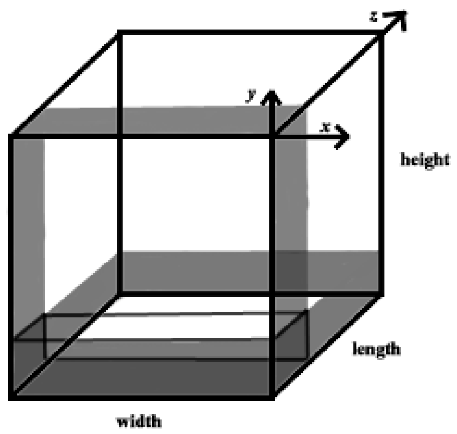
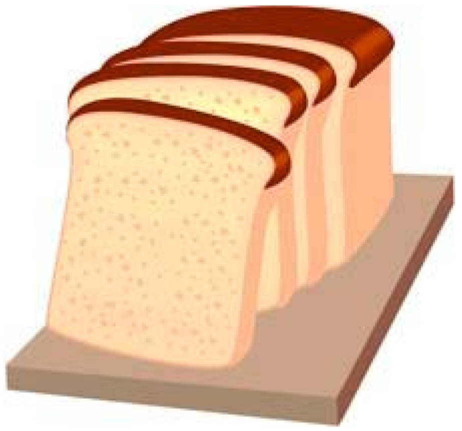


Figure 1B: Slices are analogous to slices of a loaf of bread, while levels are analogous to the floors of an office building.[i]



These level slices are the subdivisions that represent a one-dimensional bin. The adaptation algorithms place boxes in a level-slice until the remaining width in the level-slice—or the space between the previously packed box and the edge of the bin—is full or insufficient to place more boxes. As boxes are added to the level slice, the length and height of the level-slice is adjusted. For instance, if a box is taller than the height of the current level-slice (or the height of the previously tallest box in the level-slice) the height of the level-slice is adjusted to match. The same is true for boxes that are longer than the current level slice. If a box is either too wide or too long to fit in the box, a new level-slice must be defined. (Because the level-slice approach guides boxes from the bottom up, if a box is too tall to be placed in the bin, it must be discarded.)

If there is sufficient length in the current level, a new level-slice is defined by adjusting the slice of the old level-slice. In other words, a new level-slice has the same base and height as the previous level-slice, but has length that begins at the end of the previous level-slice. The total length of the new level-slice equals the length of the first box to be placed in the level-slice. If the box is taller than the previous level, the height for all slices on this level is adjusted to match. If there is not enough length in the current level to create a new slice, a new level must be defined. The base of the new level begins at the height of the previous level and has height equal to the height of the first box to be packed. The new slice begins at the front of the bin and likewise has length equal to the length of the first box to be packed. Level-slices are defined until no more room exists to define new enclosures. When each level-slice is full, the algorithm terminates.

The 1D Algorithm Adaptations

One question defines the difference between each of the six adaptation algorithms: how does one choose which level-slice to place new boxes into? Next Fit takes the simplest approach. Next Fit adds boxes into a level-slice until it is full. A new level-slice is created and filled. The process continues until all boxes are exhausted or no more level-slices may be defined. Like the one dimensional version, three dimensional Next Fit runs in O(n) time complexity. This may be seen because adding boxes to a level-slice is a constant-time operation. Next Fit's time-complexity depends only on iteration over the list of objects [2].

Figure 2A: Given a 3D bin and a list of boxes, the Next Fit algorithm will define level-slices as a guide for placing boxes.

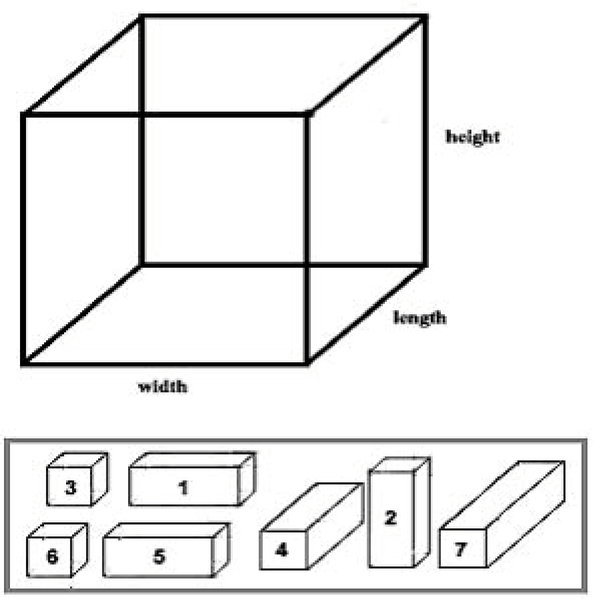
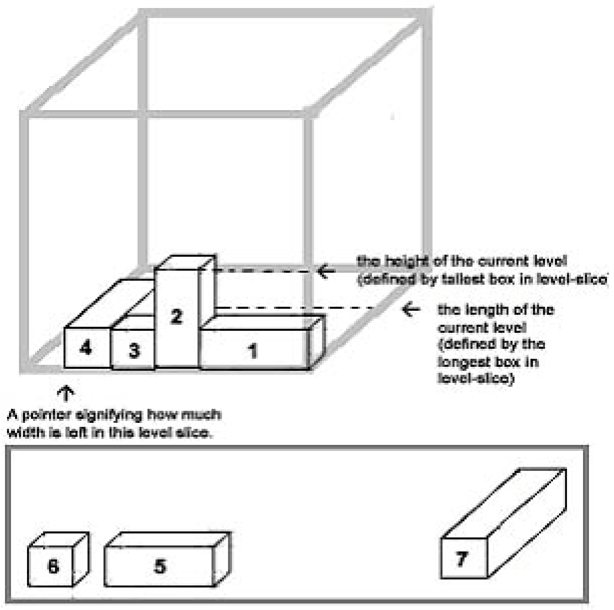


Figure 2B: After packing four boxes, there is not enough width to pack box 5, so a new slice must be defined.


the height of the current level (defined by tallest box in level-slice)
the length of the current level (defined by the longest box in level-slice)
A pointer signifying how much width is left in this level slice.
4
3
2
1
6
5
7

Other Adaptation Algorithms (Best Fit, Worst Fit, Almost Worst Fit, First Fit, and Last Fit)

The other algorithms choose to place a given box in a level slice based upon a condition. Best Fit places boxes in a level-slice that has the least amount of free width that will still fit it. Conversely, Worst Fit places boxes in the level-slice with the most free width. Almost Worst Fit places objects in the boxes with the second-most free width. In one-dimensional cases, Almost Worst Fit packs better than Worst Fit. In testing, a determination was made if this remains true in a three dimensional approach. First Fit places boxes in the first bin that will fit the box while Last Fit places boxes in the last bin.

Several considerations must be made in this approach. A box with length and width greater than that of a level-slice may not be packed in that level-slice, unless the level-slice is the most recently defined. To do otherwise could cause overlap into the boundaries of another level-slice. Also, because in the worst case a search must be made through n – a level slices, where n is the number of objects and a is the current number of level-slice, the complexity of these algorithms is O(n lg n).

Problems with the Level-Slice Approach

This level-slice approach used to adapt the popular one dimensional bin-packing algorithms works well on average, but comes with a significant flaw. While each of these techniques will provide reasonable packings, extremely large heights or lengths creates large "gaps" of space. In other words, when any of these algorithms place an object in a new slice, the location of that slice equates to the longest object in the previous slice. If the other objects in that previous slice were less long, much wasted space is produced. In a worst case, if the first object had equal height and equal length to the room, no other object could be added to that room. In another worst case, if we ever an object is added with height that extends to the top of the bin, no new levels may be generated. Fortunately, sorting the objects will significantly reduce wasted space.

Figure 3A: The Slice Problem. Because of the extreme length of box two, the next defined slice leaves much wasted volume.

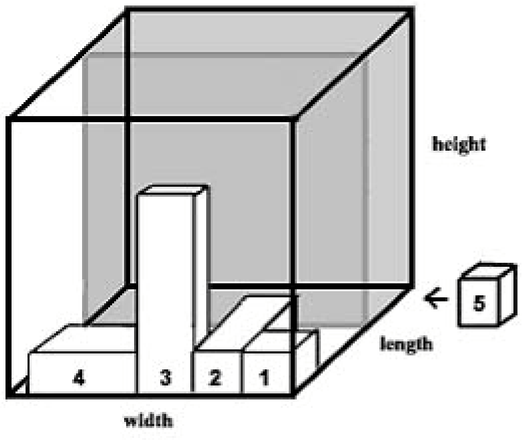
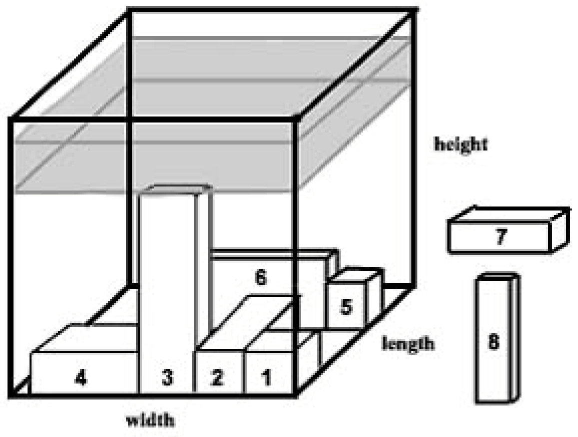


Figure 3B. The Level Problem. Because of the extreme height of box three, the next defined level leaves much wasted volume due to height.



Sorting the Boxes

Sorting the boxes by any or all of their attributes will improve packing efficiency. Particularly, sorting by height and length will reduce empty space created by the level-slice technique. The list should first be sorted from least high to highest bins so that wasted space due to levels may be reduced. Then within that sorted list, sublists of similarly high objects should be sorted by their lengths. This will reduce wasted space due to slices. If any objects hold equal or similar heights and widths, that potentially small subset may be sorted from largest width to shortest width, which may slightly improve the packing.

Implementation Notes

These six approaches were implemented in the Java programming language and run through a fairly robust, randomly generated set of tests. Thirty-two different categories of test sets were generated, with each test set containing two-hundred unique, randomly generated tests. Test sets were so diversely generated to see if certain algorithms performed better in different situations. In each test set, the number of boxes were ensured to have greater volume than the bin itself so that each set maintained uniformity. Some of the factors manipulated in each test category included:

- Large bins (sides ranging from 40-60 units) and small bins (sides ranging from 15-20 units)
- Boxes sides equaling no greater than 100% the side of the bin or box dies equaling no greater than 50% the side of the bin
- Sorted boxes and unsorted boxes
- Similar shaped boxes and randomly shaped boxes

- Box side sizes distributed randomly, as a bell curve, exponentially larger, and exponentially smaller

In each test category, three different aspects were tested. These qualities of packing included:

- Ratio of empty spaces to full spaces in the bin after packing
- Number of unused boxes after packing
- Time needed to pack the boxes were compared

To preserve the integrity of the time tests, each test set was run on the same machine with no other user interaction and as few of outside processes running as possible.

Finally, a random algorithm was generated as a benchmark to compare these algorithms to. This algorithm persistently attempts to place each box in a randomly determined location throughout the bin. The random algorithm attempts to place the box 10000 times before giving up.

Results and Implications

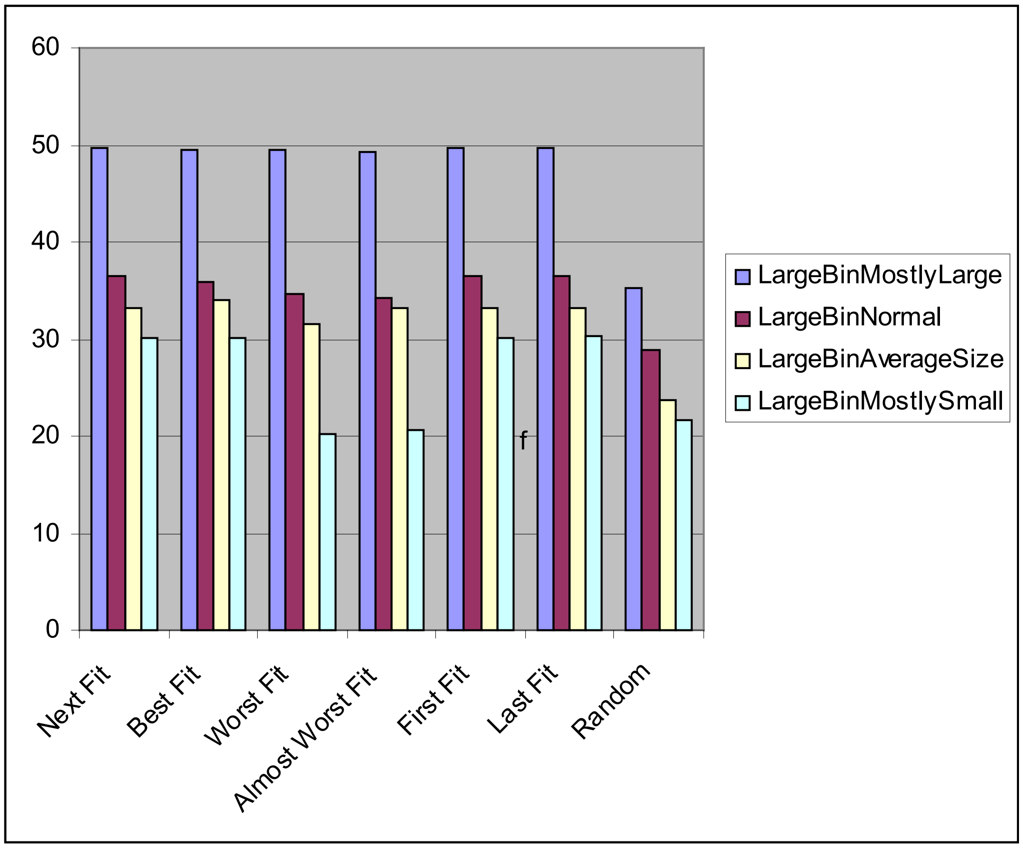
Several implications may be drawn from my results. First, pre-sorting the boxes almost always improved the efficiency of the packing by as much as 20%, except in the case of the random algorithm, where there was no effect. Also, these algorithms do not pack better than 50% in most cases. Generally, packing fullness ranged from 30%-50%. However, in the vast majority of cases when boxes were sorted, the six algorithms fared better than random chance. Interestingly, the random algorithm generally performed better when boxes were not sorted. However, in all cases the adaptation algorithms were up to 1000 times faster than random in their generations of packings.

The number of unused boxes to used boxes was almost proportional to the pack efficiency in almost all cases, except for the random algorithm, which fared just as well or significantly better than the other algorithms.

Small bins packed about 10% more efficiently than large bins. Also, similar shaped boxes and half-sized boxes produced better packings. However, this is not surprising since similar sized boxes and half-sized boxes reduce the level-slice problem. Also, Worst Fit and Almost Worst Fit seemed to perform poorly in cases when boxes were half-sized, small, or similarly shaped. With large bins, this difference in efficiency was as great as 50%. Also, there was almost no statistical difference between Worst Fit and Almost Worst Fit.

Finally, with large bins, mostly large box size distributions produce the best packings for the adaptation algorithms—up to 50%--and mostly small box size distributions produce the worst packings. Also, purely random distributions produce slightly better results than average (bell-curve) distributions. With small bins, packing efficiency ranges from best to worst in this order: mostly-large, mostly-small, pure-random, and average.

Figure 4: A sample of results from testing seven packing algorithms. The y-axis represents a ratio of empty space vs. free space.



Future Research

Several approaches to generating solutions have been discussed. Future research in this area could include testing these approaches against the optimal solution derived from a brute force algorithm. Also, adapting the level-slice approach to create level-slices that may pass through other level-slices may cut down on wasted space, but presents overlap issues that would have to be dealt with. These algorithms could also be adapted to test non-rectangular shaped objects. Three dimensional bin-packing provides almost limitless areas of research, and equally limitless desire for good solutions.

References

1. Seiden, Stephen S. and Rob van Stee. (2002). *New bounds for multi-dimensional packing. Society for Industrial and Applied Mathematics.*
2. Corcoran, Arthur L. and Wainright, Roger L. (1992). *A genetic algorithm for packing in three dimensions*. Symposium on Applied Computing. Proceedings of the 1992 ACM/SIGAPP symposium on Applied Computing: technological challenges of the 1990's.
3. Corman, Thomas H., et al. (2001). *Introduction To Algorithms*. MIT Press, Cambridge.

4. H.T. Dean, J.N. Baggaley and R.J.W. James. (1999). *Three Dimensional Container Packing of Drums and Pallets*. http://www.mngt.waikato.ac.nz/orsnz99/PDFs/James.pdf
5. "Three dimensional Packing Problems." (2003). The Institute for Algorithms and Scientific Computing (SCAI, Germany.)
6. "Bin Packing." (2003). University of Arizona. http://www.cs.arizona.edu/icon/oddsends/bpack/bpack.htm.

Acknowledgements

I would like to acknowledge and thank Rob Faux for the inspiration of this paper and Dian Lopez for advising this project.

---

[i] Graphics found at: http://www.natlsco.com/newsletter/01fall/officebuilding.jpg and http://www.ifanca.org/images/graphics/bread_09.jpg and are copyright their respective owners.