Project Two: Testing Solution Quality

Jean-Paul Nguyen

May 20, 2019

Contents

1 Introduction

1.1 Purpose

The purpose of this project is to examine and measure the performance of different bin packing algorithms by experimentally determining the quality of solutions that the algorithms produce. This quality is determined by the waste $W(a)$ of a bin-packing algorithm a, which is defined as the following:

The number of bins used by algorithm a minus the total sum of all items in the given list.

These will go over different algorithms to solve the bin-packing problem, using the observed waste for each algorithm to determine which is the most effective algorithm for this problem.

1.2 Experimental Setup

Each of the bin-packing algorithms were developed with the C++ function headers per the project requirements, and I created a function that would take a given algorithm a, a number of *reps*, and a *limit*. With these parameters, a would be run on an input of initially size 10, and the size would be multiplied by 10 up until *limit* was reached. Each input was a vector of randomized doubles from 0.0 to 1.0, non-inclusive.

Here, *reps* number of inputs were done, and a was ran on each of them, with a calculated waste that was retrieved. After each input was bin-packed, the resulting wastes were averaged, and the average waste and size of the input was stored for each a.

With all the data, the points were plotted on a log-log scale with the size of the input n as the x-axis and the measured average weight $W(a)$ as the y-axis. The linear regression of the log's were taken instead of the actual data since each of the bin-packing algorithms were expected to have a non-linear relationship between size and waste. Along with general observation of the data to determine what would be the better bin-packing algorithm, the slopes of these linear regressions were used as more concrete measurements on the potential growth of waste as size increased for each algorithm. Note that the log base used was 10.

For all the data measured in this project, I stuck to 5 *reps* and a *limit* of $100,000$ for every algorithm. These parameters were chosen with a consideration for the first fit and best fit algorithms (both of which were $O(n^2)$ and thus were the slowest). In addition, $100,000$ allowed for a given 5 points of $(n, W(a))$ for an algorithm a, which I felt was sufficient enough for the linear regression data of each a.

1.3 Outline

The paper will delve into the following algorithms:

1. Next Fit (NF)
2. First Fit (FF) and First Fit Decreasing (FFD)
3. Best Fit (BF) and Best Fit Decreasing (BFD)

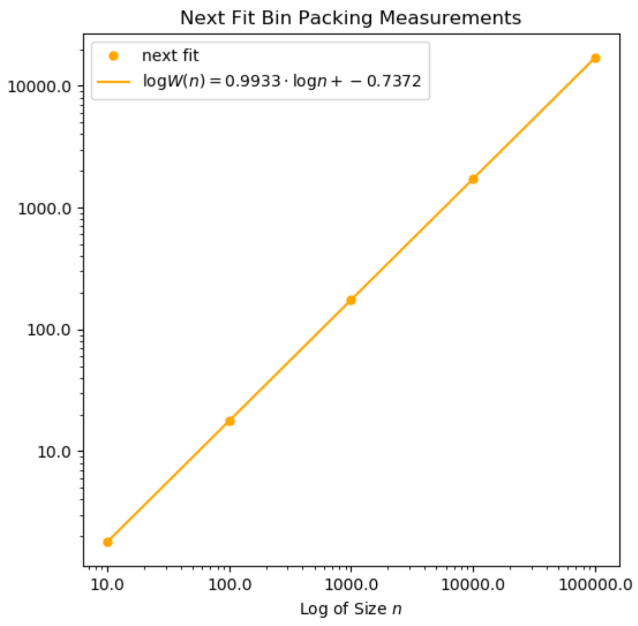
The implementation and measure of performance of each algorithm will be done, with FF and BF having counterparts with modifications on the input to be compared to. Finally, the analysis will be done, with a verdict on which algorithm is the best for the bin-packing problem at the end.

2 Next Fit

2.1 Implementation

NF is a simple bin-packing algorithm where for each item in the input, either the item fits in the *current* bin or a new one is created. For my implementation in C++, the algorithm was pretty standard, having a $O(n)$ time complexity with one for loop going through each item.

2.2 Performance



n	waste
10	1.80335
100	17.7437
1000	175.58
10000	1711.84
100000	16997

Figure 1: Graph and Data of NF

Here is a graph of NF run on input sizes from 10 to $100,000$ with calculated average wastes as well as the linear regression of the log's of the data. In addition, a table of the data from measuring the quality of NF is included.

2.3 Analysis

As shown, there clearly is a linear relationship between the log of sizes and the log of wastes, and we have a slope m and an intercept b that we can work with to determine the relationship between n and $W(a)$. Indeed,

$$\begin{aligned}
\log W(a) &= m \cdot \log n + b \\
10^{\log W(a)} &= 10^{m \cdot \log n + b} \\
W(a) &= 10^{logn^m} \cdot 10^b \\
W(a) &= n^m \cdot 10^b
\end{aligned}$$

Thus, we can see that the slope indicates a power of which $W(a)$ grows as n grows. We can use this to examine the other bin-packing algorithms later in the paper.

With regards to NF, we have the linear regression equation

$$\begin{aligned} \log W(NF) &= 0.9933 \cdot \log n - 0.7372 \\ W(NF) &= n^{0.993} \cdot 10^{-0.7372} \\ \therefore &\boxed{W(NF) = O\left(n^{0.9933}\right)} \end{aligned}$$

We can see that as n grows, $W(NF)$ also grows at a close to linear rate, which is not really optimal since waste can grow really large. Nevertheless, NF acts as a good baseline for FF and BF to later improve on, in particular to have a slope m that is smaller than 0.9933.

3 First Fit and First Fit Decreasing

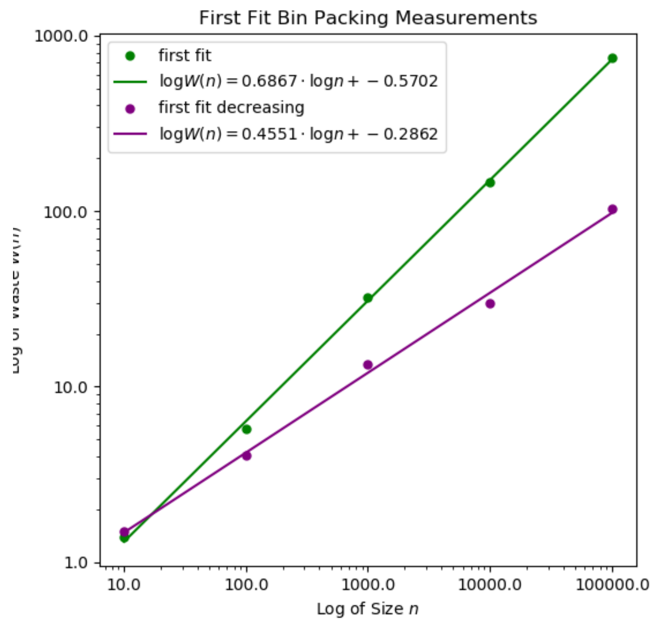
3.1 Implementation

FF is an optimization of NF that starts from the *first* created bin and checks which bin would be able to accommodate the given item (as opposed to NF where only the current bin is checked). This comes at the cost of increasing the time complexity up to $O(n^2)$; while it is possible to make the algorithm $O(n \log n)$ with the use of a balanced binary tree, I only had enought time to do the $O(n^2)$ solution.

Along with FF, FFD is the decreasing version of FF where the input is first sorted in decreasing order before FF is run. For both FFD and BFD (which is discussed later in the paper), I had the C++ implementation of std::sort be used on the input first, which has a time complexity of $O(n \log n)$. However, to maintain the indexes of the original vector, I first made an $O(n)$ pass that constructed a vector of structs, each with the value and index. From there, std::sort was called with the comparison being a "custom" greater than which only compared the values of the structs. This allowed the assignment vector to be maintained with the original indexes as opposed to the new sorted indices.

Overall, the implementations of FF and FFD were pretty straightforward since the algorithm was simple to implement using the $O(n^2)$ solution.

3.2 Performance



algorithm	n	waste
first fit	10	1.38467
first fit	100	5.77838
first fit	1000	32.2867
first fit	10000	145.632
first fit	100000	748.636
first fit decreasing	10	1.48967
first fit decreasing	100	4.03975
first fit decreasing	1000	13.3861
first fit decreasing	10000	29.9484
first fit decreasing	100000	103.234

Figure 2: Graph and Data of FF and FFD

Here is a graph for FF and FFD, which were run on input sizes from 10 to $100,000$ with calculated average wastes and linear regressions of the log's of the data. A table of the data is also included for FF and FFD.

3.3 Analysis

Like NF, both FF and FFD have linear relationships that are represented by the linear regressions of the log's, which have slopes that we can analyze. Using our relationship of n and $W(a)$ from before, we know that the complexity of $W(FF) = O\left(n^{0.6867}\right)$ and the complexity of $W(FFD) = O\left(n^{0.4551}\right)$. These slopes are less than 1, which indicate that the waste of FF grows at a smaller rate compared to NF. This means that less waste is generated in comparison to NF as size grows, and this is more optimal since we are aiming to have less waste. As such, we can FF and FFD are better than our baseline, NF.

When comparing FF and FFD, we can see that the decreasing version has a smaller slope than the original. Thus, FFD is generating less waste (and is performing better) on average compared to FF. This may be due to having the input be sorted in decreasing order; the bigger items are placed in bins first before the smaller items occupy the remaining space. Since this method turns out to be more optimal, we can infer that the ordering of the incoming items play a role in the effectiveness of FF.

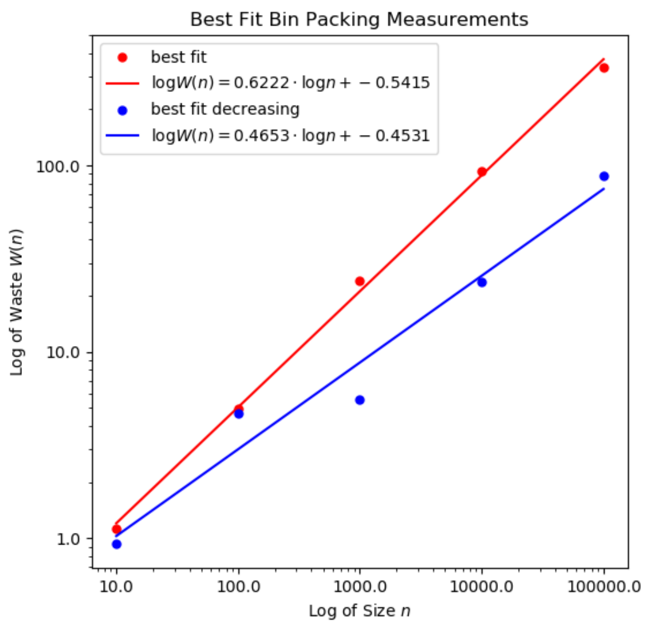
4 Best Fit and Best Fit Decreasing

4.1 Implementation

BF is another algorithm that improves on NF by making a pass on all the current bins and checking which bin fulfills a heuristic: the bin that gives the “tightest” fit for an item. Mainly. the algorithm checks which bin gives the minimum amount of waste after adding the item in the bin. Like FF, the algorithm is $O(n^2)$ in time complexity with a potential to be $O(n \log n)$ by implementing with a balanced binary tree, though I did choose to stay with the $O(n^2)$ implementation again.

BFD is the decreasing version of BF that sorts the input first in decreasing order, similar to FFD. I implemented the same sorting implementation for BFD as well to maintain the original indices of the input. The implementations of both algorithms were just as straightforward as FF and FFD.

4.2 Performance



algorithm	n	waste
best fit	10	1.13021
best fit	100	4.93017
best fit	1000	24.1481
best fit	10000	93.5517
best fit	100000	334.828
best fit decreasing	10	0.935714
best fit decreasing	100	4.70652
best fit decreasing	1000	5.58743
best fit decreasing	10000	23.8544
best fit decreasing	100000	88.1316

Figure 3: Graph and Data of BF and BFD

Here is a graph for BF and BFD, which were run on input sizes from 10 to 100,000 with calculated average wastes and linear regressions of the log’s of the data. A table of the data is also included for BF and BFD.

4.3 Analysis

Both BF and BFD have linear relationships based on the linear regressions of the log's, just like the previous algorithms. Using the relationship of n and $W(a)$, we get that the complexity of $W(BF) = O\left(n^{0.6222}\right)$ and the complexity of $W(BFD) = O\left(n^{0.4653}\right)$. These are clearly improvements over the baseline NF, and just like with FF and FFD, the decreasing version does more optimal with the smaller slope. This strengthens the idea that the order of which the items are inserted into the bins can greatly affect the effectiveness of bin-packing algorithms, depending on the heuristics of how a bin is picked for an item to be placed in.

5 Comparison of All Bin Packing Algorithms

Now, with every algorithm discussed and examined in detail, we can compare all of them with each other:

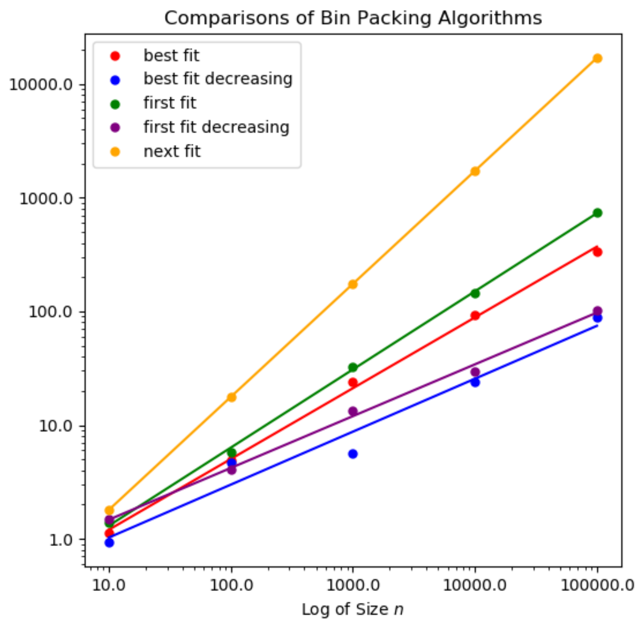
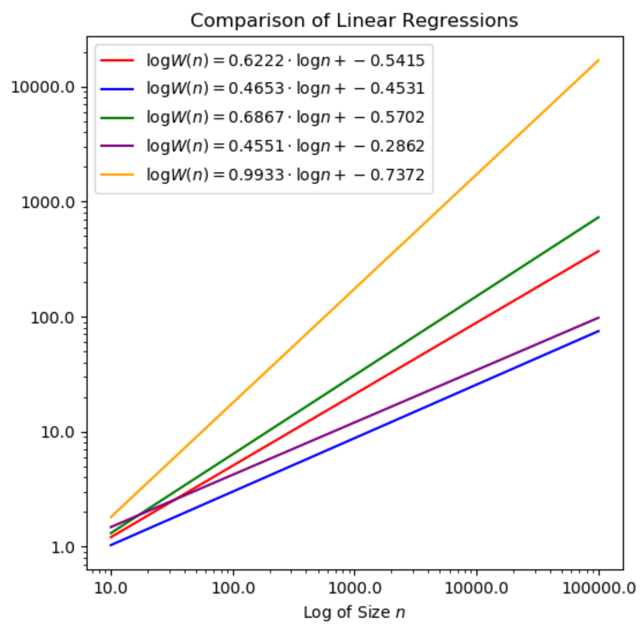


Figure 4: Graph of All Algorithms

Shown are the data points and lines of the Bin Packing algorithms in one graph, showing how each algorithm performs relative to the others. We can see that as the size increases, all the other algorithms are more optimal than next fit by a significant margin, with the decreasing algorithms being the most effective.

We can see that between FF and BF, BF ends up having less average waste as size increases. This implies that with a randomized input, BF performs better than FF. Based on how the algorithms work, this makes sense; while FF will stop immediately at the first bin that fits, BF will go through every bin and use that to determine which bin is the better choice to put the item in. This is a heuristic that works better with more information, so BF would have a better guess as to which bin is more optimal, while FF may stop at a bin that is less than optimal for an item.

As for the decreasing versions of FF and BF, it is clear that these versions perform better than their original counterparts. With a sorted, decreasing input, these algorithms do end up choosing more optimal bins to place items in; with the bigger items in the bins first, it is more likely for FFD and BFD to choose the optimal bins for the smaller items. This can be reflected in the data: With 100,000 items, the waste from FFD and BFD are less than NF's waste on just 100 items! FFD and BFD seem to be good candidates for the best algorithms.



algorithm	slope
best fit	0.6222
best fit decreasing	0.4653
first fit	0.6867
first fit decreasing	0.4551
next fit	0.9933

Figure 5: Graph and Table of Slopes of All Algorithms

Here are the lines of each of the algorithms along with a table of the slopes, which we have been using to determine the complexities of $W(a)$ given the size n. Here, we see something interesting: with the data measured, the slope of BF is better than FF, but the slope of BFD is worse than FFD. While $W(BF)$ grows at a slower rater than $W(FF)$ on randomized input, $W(BFD)$ may actually grow faster than $W(FFD)$. Even though the wastes of BFD are smaller than FFD, the data implies that with a big enough size, $W(BFD)$ can outgrow $W(FFD)$. Based on the slopes alone, **FFD would be the best algorithm**.

6 Conclusion

Based on the data, the decreasing algorithms, FFD and BFD, performed better than the other algorithms, with NF being the least-optimal in the average waste measured. With the slopes gathered from the data collected, we can say that FFD is, in practice, the best algorithm for the bin-packing problem. However, the 0.01 difference may indicate that there was not enough data to confirm this answer; there may have been outliers that affected the slopes of FFD and BFD, and the actual slopes of the linear regressions may change as the size gets bigger. With more time and a faster computer, I would increase the *limit* to larger numbers; this can confirm whether BFD eventually outgrows FFD or if BFD stays smaller than FFD.

Nevertheless, it is clear that sorted, decreasing input works with the heuristics of FF and BF to get smaller and more optimal wastes as sizes get larger. The data suggests that BF works better on randomized input, while FF will eventually work better on the sorted, decreasing input. In the end, FFD came out on top.