**Algorithmic Experiments of Real-World Phenomena – CS 165 Project 3**

Heejin Kang (76507705)

University of California, Irvine

**Introduction**

This project involves testing various graph algorithms experimentally to determine properties of models of real-world networks. We implement three network algorithms (diameter algorithm, clustering-coefficient algorithm and degree-distribution algorithm) and test these with two different random graphs (Erdos-Renyi random graph and Barabasi-Albert random graph). We plot the result of diameter algorithm and clustering-coefficient algorithm in lin-log plots and the result of degree distribution algorithm into lin-lin plot and log-log plot, and find a function of n.

**Random Graph Generations**

**Erdos-Renyi Random Graph**

*Erdos-Renyi(n)*

*Input: number of nodes*

*Output: edges of the graph*

*p ←, v ←1, w←-1*

*while(v < n)*

*r = uniformly random number [0,1)*

*w ← w + 1 +*

*while( w>= v and v < n)*

*w ← w – v, v ← v+1*

*E←E*

The Erdos-Renyi random graph has fixed number of vertices, and every pair of nodes is connected independently with probability *p*. The function for generating Erdos-Renyi Random Graph takes the number of nodes, and we set fixed p be *p=2(ln n)/n*. We initially set the “from” vertex be 1 and the “to” vertex be -1, and r be a uniform random number between 0 and 1. Then, we increment *w* *1 + ,* and while w is bigger than or equal to *v* and *v* is smaller than *n*, we decrement *w* by *v* and simultaneously increment *v* by 1. After this, we add *v* and *w* pair into the edge set. We iterate this until *v* is greater than or equal to *n*.

**Barabasi-Albert Random Graph**

*Barabasi-Albert (n, d)*

*Input: number of nodes, min degree*

*Output: edges of the graph*

*For(v=0,…,n-1)*

*For(i=0,…,d-1)*

*M[2(vd+i)] ←v*

*r ← uniformly random number between 0 and 2(vd+i)*

*M[2(vd+i)+1] ← M[r]*

*For(i=0,…,nd-1)*

**Experiment**

We experiment with 10, 50, 100, 500, 1000, 5000, 10000, 50000 and 100,000 of nodes and generate two different types of random network graphs (Erdos-Renyi and Barabasi-Albert). For each graph, we will compute three different network algorithms, such are diameter algorithm, clustering-coefficient algorithm and degree-distribution, and plot them into graphs for visualization of its performance.

For each bin packing algorithm, we experiment with 10, 50, 100, 500, 1000, 5000 and 10000 elements of input data that are in range between 0.0 and 0.7, and these are uniformly distributed. By doing this, we can find W(A) for each algorithm for each input size. After running all the cases, we plot the result into a log-log plot whose x-axis represents the number of elements and y-axis represents the waste. Then we compute its slope k to determine their running time and a function of n.

**Next Fit Algorithm**

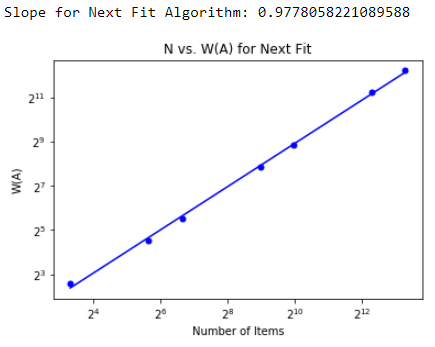
Next fit algorithm checks if the current item fits in the current bin. If the item fits, place it there, otherwise start a new bin. The algorithm is very simple but is not an optimal algorithm that creates the least number of bins to place all the items. *Figure 1* shows the log-log graph of next fit algorithm. As shown, there’s a line that fits the data. The slope of the line is 0.9778, and y-intersection is -0.5997. Therefore, we can estimate W(NF) as a function of n:

Figure Log-log plot for next fit algorithm

Also, by the definition of running time, next fit algorithm’s running time is .

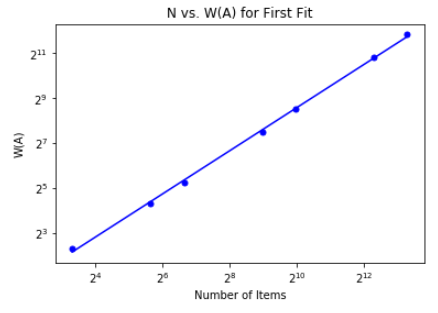
**First Fit Algorithm**

Figure Log-log plot for first fit algorithm

First fit algorithm scans the bins in order to find the first bin that is large enough to hold the current item. If a bin is found, the item is placed to the bin, otherwise it creates a new bin. To efficiently implement this algorithm, we use tuples that the key is the bin number that the item has been placed and the value is remaining capacity. Also, we keep the largest remaining capacity in the subtree rooted at the node to the value by pairing with the remaining capacity because that helps for efficient searching for the first bin that can fit the current item. By using a balanced search tree, we can implement this algorithm in . *Figure2* shows the experiment result of first fit algorithm using a balanced search tree. The slope of the line is 0.9602 and y-intersection is -0.7161. Therefore, we can estimate W(FF) as a function of n:

Also, by the definition of running time, next fit algorithm’s running time is .

**Best Fit Algorithm**

The best fit algorithm places a new item into a bin where it fits the tightest. If a bin is found, the item is placed to the bin, otherwise it creates a new bin. Similar to first fit algorithm, we use tuples to efficiently implement this algorithm. However, in this case, the key is now the remaining capacity and the value is the bins with the capacity, and this helps to find the tightest bin for the current item. Same to first fit algorithm, it can also be implemented in by using a balanced search tree. *Figure 3* shows the experiment result of best fit algorithm using a balanced search tree. This algorithm is not perfectly implemented, so there was an error when the algorithm is run with large number of random inputs. Therefore, I experimented this algorithm only with 5 cases that are 10, 50, 100, 500 and 1000. With the dataset, the slope of this line is 0.9308 and y-intersection is -0.5949, which leads to the following equation:

This may result different if the dataset was same to the other algorithms. By the definition of running time, best fit algorithm takes .

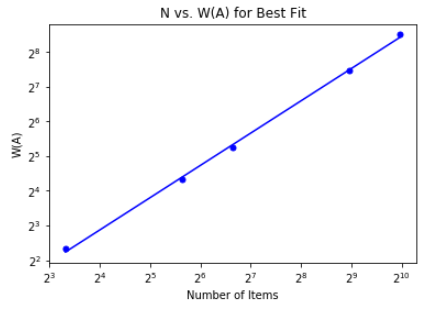


Figure Log-log plot for best fit

**First Fit Decreasing & Best Fit Decreasing**

Both first fit decreasing and best fit decreasing algorithm takes almost same process as non-decreasing version of algorithms. The only difference is that it sorts inputs in a decreasing order so that it places the largest input to the first bin and keeps creating new bins until the input is small enough to fit in the existing bin. Otherwise, it keeps creasing new bins. This algorithm can be regarded as an optimal algorithm. *Figure 4* shows loglog graphs of both first fit decreasing algorithm and best fit decreasing algorithm. For each algorithm, they can be expressed as the following equations:

Also, by the definition of running time, the first fit decreasing algorithm’s running time is , and the best fit decreasing algorithm’s running time is .

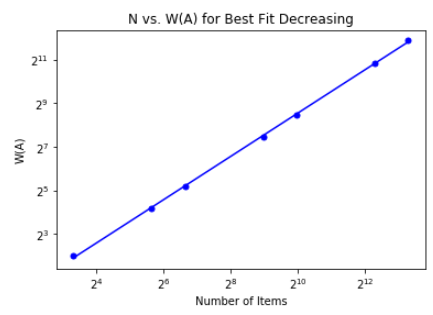
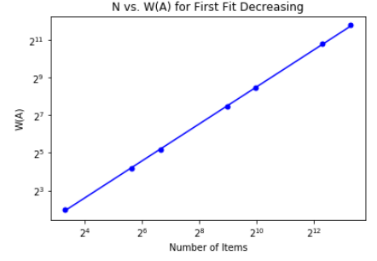


Figure 4 Log-log plot for first fit decreasing (left) and best fit decreasing (right)

**Conclusion**

We conclude that the best algorithm based on our experiments is the best fit algorithm because the slope of the graph is the smallest among all the algorithms. However, this is somewhat different than the theories of algorithms that we have learned in lectures. I believe that the best fit decreasing or the first fit decreasing must be the ideal algorithm for the bin-packing problem. That is because sorting the input in decreasing order before packing helps for the big-sized input be placed in the right bins, and then small-enough-sized inputs find their own spots. Because we have found that the best fit algorithm has some errors while running the code, there must be some problems that caused this different result. We will keep experiment and change the code to get the right result.