CS 260P Project 2

Chad Bloxham
UID: 81321335
cbloxham@uci.edu
May 24, 2019

1 Overview

The bin-packing problem can be stated as follows: given a list of items, each of which has a size, find the minimum number of bins, each with a maximum capacity, that can hold all of the items. We compare the performance of several algorithms which solve the bin-packing problem. The first is Next Fit, which simply fills a bin until the current item cannot fit, at which point the item is added into a new bin. Next is First Fit, which places the current item in the first bin that can hold it. The last algorithm we test is Best Fit, which places the current item in whatever bin it fits the tightest i.e. leaves the least amount of remaining space.

We also test variants of First Fit and Best Fit called First Fit Decreasing and Best Fit Decreasing, respectively. These are identical to First Fit and Best Fit except that the items are first sorted by size in descending order before bin-packing begins. We measure performance in terms of waste, which we define as the difference between the number of bins used and the total sum of the sizes of the items in the input list. We set the capacity of each bin to 1. The sizes of items range uniformly between 0.0 and 0.8, and we will run the algorithms on various lengths of item lists.

2 Algorithms

Below are pseudocode descriptions for each bin-packing algorithm as well as a brief description of the code used to compute the average waste for every input size considered. Item lists were generated using the uniform() function from Numpy's random library.

2.1 Next Fit

```
Next_Fit(items):
    Input: items, a numpy array of size n containing a uniform distribution
    of floats between 0.0 and 0.8.
    Output: W, the waste associated with packing the items into
    bins with capacity 1.
    item_sum = sum(items)
    num_bins = 1
    current_bin = 0
    for each item in items:
        if item fits in current_bin:
            current_bin = current_bin + item
        else:
            num_bins = num_bins + 1
            current_bin = item
    W = num_bins - item_sum
    return W
```

2.2 First Fit

First_Fit(items, decr=False):

Input: *items*, a numpy array of size n containing a uniform distribution of floats between 0.0 and 0.8.
decr, a boolean value which determines whether the items are sorted in descending order before bin-packing begins. Default value is False.
Output: *W*, the waste associated with packing the items into bins with capacity 1.

```
item_sum = sum(items)
if decr == True:
    sort items in descending order
bins = [0]       //List containing 1 empty bin.
for each item in items:
    bin_found = False
    i = 0
    //Find the first bin the item fits in. If there is none, create a new bin.
    while bin_found == False and i < len(bins):
        if item fits in bins[i]:
            bin_found = True
            bins[i] = bins[i] + item
        else:
            i = i + 1
        if i == len(bins):       //Need a new bin.
            bins.append(item)
W = len(bins) - item_sum
return W
```

2.3 Best Fit

Best_Fit(items, decr=False):

Input: *items*, a numpy array of size n containing a uniform distribution of floats between 0.0 and 0.8.
decr, a boolean value which determines whether the items are sorted in descending order before bin-packing begins. Default value is False.
Output: *W*, the waste associated with packing the items into bins with capacity 1.

```
item_sum = sum(items)
if decr == True:
    sort items in descending order
bins = [0]       //List containing 1 empty bin.
for each item in items:
    diffs = [ ]       //Will contain the remaining space in each bin assuming the item is placed.
    for each bin in bins:
        diffs.append(remaining space in bin if we add item)
    pos_diffs = all positive values in diffs
    if len(pos_diffs) == 0:       //Need a new bin.
        bins.append(item)
    else:
        bin_num = index of value min(pos_diffs) in diffs
        bins[bin_num] = bins[bin_num] + item
W = len(bins) - item_sum
return W
```

2.4 Computing Wastes

We ran our bin-packing algorithms on item list lengths which were powers of 2, ranging from $N_{min} = 2^4$ to $N_{max} = 2^{14}$. For each input list length, we ran each algorithm five times and stored the average of the resulting waste values in a numpy array.

3 Results

As show in Fig. 2, Next Fit yields the worst performance. This is to be expected because Next Fit is the most elementary of all the algorithms. It only has one available bin in which the current item can be placed, and the remaining space within this bin is not guaranteed to be used efficiently. For example, imagine the only available bin has 0.6 space remaining and the current item has a size of 0.7. Since this item will not fit, it will be placed into a new bin and all of that remaining space will be wasted. In the First Fit algorithm, all bins can be considered and it is possible that there will be a bin in which the 0.7 item will fit, and a new bin will not have to be opened. This explains the superior performance of First Fit over Next Fit.

Best Fit uses a more sophisticated approach which considers all bins and chooses the bin which minimizes the amount of remaining space after the current item is placed. This is an improvement over First Fit (as shown in Fig. 2) because it maximizes the amount of remaining space in all other bins, thus decreasing the probability that new bins will have to opened for future items. Because First Fit places an item in the first bin in which it fits, it is possible that there is another bin in which the item could fit more tightly. This optimal bin is always chosen with Best Fit.

The decreasing versions of First Fit and Best Fit have significantly better performance than the standard versions. Packing the largest items first may seem counterintuitive since large items are more likely to not fit in any bin, causing a new bin to be opened. However, this probability only increases as more items are added, so it is better to get the large items packed earlier. More bins are opened early on, but later items are much more likely to fit in the remaining spaces since they are smaller, and the probability of having to open a new bin for a given item plummets. Best Fit Decreasing slightly outperforms First Fit Decreasing for the same reasons explained earlier when analyzing the standard versions of the algorithms.

Algorithm	Waste
First Fit Decr.	$O(n^{0.087})$
Best Fit Decr.	$O(n^{0.106})$
Best Fit	$O(n^{0.602})$
First Fit	$O(n^{0.641})$
Next Fit	$O(n^{0.996})$

Figure 1: Algorithms ranked from best to worst performing.

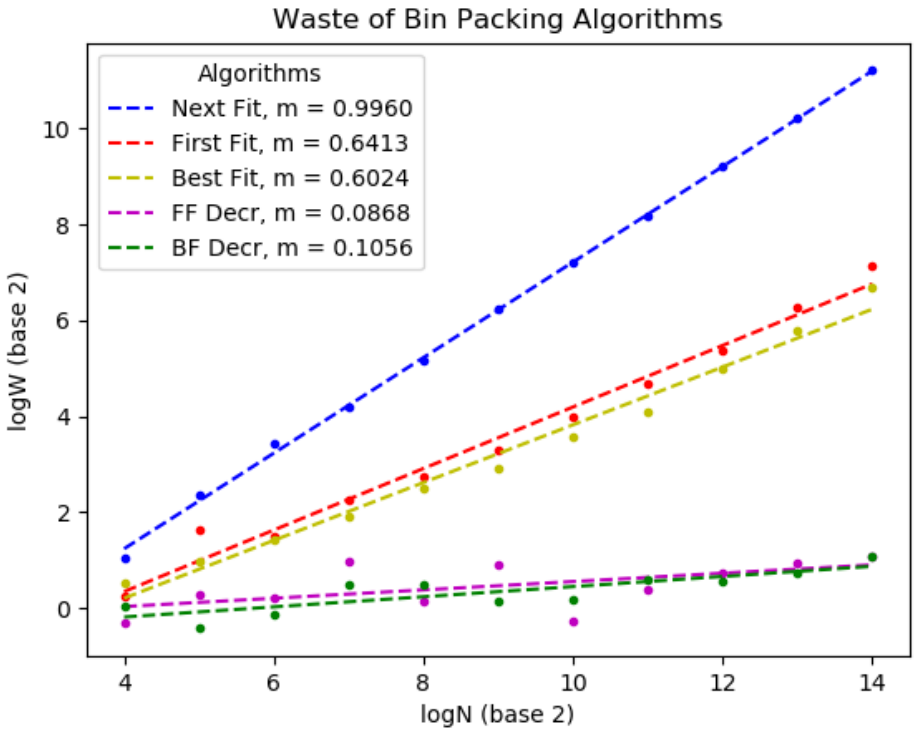

Figure 2: Waste of bin-packing algorithms as a function of number of items. Bins had a capacity of 1 and item lists consisted of uniform distributions of floats between 0.0 and 0.8.