CS325 HW8 Hao Deng ID:932912420

1). In the bin packing problem, items of different weights (or sizes) must be packed into a finite number of bins each with the capacity C in a way that minimizes the number of bins used. The decision version of the bin packing problem (deciding if objects will fit into <= k bins) is NP-complete. There is no known polynomial time algorithm to solve the optimization version of the bin packing problem. In this homework you will be examining three greedy approximation algorithms to solve the bin packing problem.

a) Give pseudo code and the running time for each of the approximation algorithms.

First-Fit algorithm (runtime: O(n^2))

There are two loops, with one inside another. The reason why this algorithm is n^2 because the number of bins and the number of items is not independent. Number of bins can be less or equal to the number of items. The outer loop is O(n) whereas the inner loop is theta n.

```
First-Fit(capacity, items, item_list[]){

    Create bin_res array with items size
    Assign num_bins to 0

    For I < items from 0, i++{
        Make an integer variable named j
        For j < num_bins from 0, j++{
            If ( bin_res at j is >=  item_list at i){
                Bin_res at j gets bin_res[j] – item_list[i]
                Break out from the current loop
            }

        }

        If(j equals to num_bins){ //adds a new bin if no bin fits the item
            Bin_res at num_bins = the capacity – item_list at i
            Num_bins add 1
        }
    }
    Return num_bins
}
```

Best-Fit Algorithm (runtime: O(n^2))

The runtime is n^2 is because it's like the first fit. The outer loop is O(n) for the number of items and the inner loops are mainly theta (n) for number of bins and the number of candidate bins for item to be packed.

```
Best_fit(capacity, items, item_list[]){

    Create bin_res array with items size
    Assign num_bins to 0
```

```
        Create a variable called smallest_ position

        For I < items from 0, i++{
                Create a candidate array with num_bins size
                Assign counter to 0

                //This loop checks the amount of bins the item can fit into
                For k < num_bins from 0, k++{
                        If (bin_res at k is >= item_list at i){
                                Candidate array at counter = k
                                Counter adds 1
                        }
                }

                Create a "j" variable
                If(counter is >= 2){ //if there are 2 or more bins that can fit the item
                        Set smallest_ position to the first candidate
                        For m < counter from 1, m++{
                                If(bin_res at candidate m is <= bin_res at the smallest
_position{
                                        The smallest position gets candidate m

                                        Break from the current loop
                                }
                        }
                        Bin_res at the smallest position gets bin_res at smallest position –
                item_list at i
                }
                Else{
                        For j < num_bins from 0, j++{
                                If ( bin_res at j is >=  item_list at i){
                                        Bin_res at j gets bin_res[j] – item_list[i]
                                        Break out from the current loop
                                }
                        }
                }
                If(j equals to num_bins){ //adds a new bin if no bin fits the item
                        Bin_res at num_bins = the capacity – item_list at i
                        Num_bins add 1
                }

        }
        Return num_bins
}
```

First-Fit-Decreasing (runtime: O(n^2))

The runtime is also n^2 because we first sort the item list with an ideal algorithm, merge sort, to have a O(nlgn) sorting time. After sorting the item list, we then pass the list to the First-Fit function, which it's O(n^2). Therefore, this algorithm is O(n^2).

```
First_fit_dec(capacity, items,item_ list_copy[]){
        Sort the list by calling merge_sort(item_list_copy, 0, items-1) function

        Return from calling the first_fit(capacity, items, item_list_copy) function

}
```

b) The README.txt and code have been submitted through Teach

c) Randomly generate at least 20 bin packing instances. Summarize the results for each algorithm. Which algorithm performs better? How often?

Description:

I first write the test cases as 30 to the random file and by using a for loop, I make a capacity to either 10 or 20; number of items is assigned randomly between 5-25 along with an item list. Inside the for loop, I created another for loop and randomly generate the weight of each item from the list between 1 to the capacity. I then write all of those to the random text file.

Summary:

All these results from the three different algorithms are quite similar with the 1-2 difference in the number of bins. Out of 30 test cases, it seems to me that the First Fit algorithm tends to take "more" bins than the other two because there is 1 test case where First Fit algorithm took 1 more bin than the rest. Otherwise, the First Fit and Best Fit algorithm tend to be very similar in number of bins. The best algorithm of these three I believe would be the First Fit Decreasing algorithm because it seems to me that there are couple test cases where it takes the least number of bins compare to the rest. The First Fit algorithm seems to be worst algorithm among the three with the better performance of 0/30 test cases. The Best Fit algorithm seems to be the in the middle between these algorithms with the better performance of 1/30 test cases. The First Fit Decreasing algorithm seems to be the best among these three with the better performance of 8/30 test cases. The rest of the test cases indicating that they all have the same number of bins.

2). Write an integer program for each of the following instances of bin packing and solve with the software of your choice. Submit a copy of the code and interpret the results.

a) Six items S = { 4, 4, 4, 6, 6, 6} and bin capacity of 10

The result I get is 3, which means that the exact number of bins I need for that given number of items at the capacity of 10. I need 3 bins to fit those items.

CODE

```
min S
ST
    S >= 1
    S - b1 - b2 -b3 - b4 - b5 - b6 = 0

    10b1 - 4x11 - 4x12 - 4x13 - 6x14 - 6x15 - 6x16 >= 0
    10b2 - 4x21 - 4x22 - 4x23 - 6x24 - 6x25 - 6x26 >= 0
    10b3 - 4x31 - 4x32 - 4x33 - 6x34 - 6x35 - 6x36 >= 0
    10b4 - 4x41 - 4x42 - 4x43 - 6x44 - 6x45 - 6x46 >= 0
    10b5 - 4x51 - 4x52 - 4x53 - 6x54 - 6x55 - 6x56 >= 0
    10b6 - 4x61 - 4x62 - 4x63 - 6x64 - 6x65 - 6x66 >= 0

    x11 + x21 + x31 + x41 + X51 + x61 = 1
    x12 + x22 + x32 + x42 + X52 + x62 = 1
    x13 + x23 + x33 + x43 + X53 + x63 = 1
    x14 + x24 + x34 + x44 + X54 + x64 = 1
    x15 + x25 + x35 + x45 + X55 + x65 = 1
    x16 + x26 + x36 + x46 + X56 + x66 = 1

    b1 + b2 + b3 + b4 + b5 + b6 >= 1

END

INT b1
INT b2
INT b3
INT b4
INT b5
INT b6
INT x11
INT x21
INT x31
INT x41
INT x51
INT x61
```

```
LP OPTIMUM FOUND AT STEP      21
OBJECTIVE VALUE =    3.00000000

SET       B1 TO <=      1 AT     1, BND=  -3.000     TWIN=-0.1000E+31     51

NEW INTEGER SOLUTION OF     3.00000000      AT BRANCH      1 PIVOT      51
BOUND ON OPTIMUM:  3.000000
DELETE       B1 AT LEVEL     1
ENUMERATION COMPLETE. BRANCHES=     1 PIVOTS=      51

LAST INTEGER SOLUTION IS THE BEST FOUND
RE-INSTALLING BEST SOLUTION...

       OBJECTIVE FUNCTION VALUE

       1)      3.000000

 VARIABLE        VALUE          REDUCED COST
       B1         1.000000          1.000000
       B2         0.000000          1.000000
       B3         0.000000          1.000000
       B4         1.000000          1.000000
       B5         0.000000          1.000000
       B6         1.000000          1.000000
      X11         0.000000          0.000000
      X21         0.000000          0.000000
      X31         0.000000          0.000000
      X41         1.000000          0.000000
      X51         0.000000          0.000000
      X61         0.000000          0.000000
        S         3.000000          0.000000
      X12         0.000000          0.000000
      X13         0.000000          0.000000
      X14         1.000000          0.000000
      X15         0.666667          0.000000
      X16         0.000000          0.000000
      X22         0.000000          0.000000
      X23         0.000000          0.000000
      X24         0.000000          0.000000
      X25         0.000000          0.000000
      X26         0.000000          0.000000
      X32         0.000000          0.000000
      X33         0.000000          0.000000
      X34         0.000000          0.000000
      X35         0.000000          0.000000
      X36         0.000000          0.000000
      X42         0.000000          0.000000
      X43         1.000000          0.000000
      X44         0.000000          0.000000
      X45         0.333333          0.000000
      X46         0.000000          0.000000
      X52         0.000000          0.000000
      X53         0.000000          0.000000
      X54         0.000000          0.000000
      X55         0.000000          0.000000
      X56         0.000000          0.000000
      X62         1.000000          0.000000
      X63         0.000000          0.000000
      X64         0.000000          0.000000
      X65         0.000000          0.000000
      X66         1.000000          0.000000


      ROW   SLACK OR SURPLUS     DUAL PRICES
       2)         2.000000          0.000000
       3)         0.000000         -1.000000
       4)         0.000000          0.000000
       5)         0.000000          0.000000
       6)         0.000000          0.000000
       7)         0.000000          0.000000
       8)         0.000000          0.000000
       9)         0.000000          0.000000
```

```
min S
ST
        S >= 1
        S - b1 - b2 -b3 - b4 - b5 - b6 = 0

        10b1 - 4x11 - 4x12 - 4x13 - 6x14 - 6x15 - 6x16 >= 0
        10b2 - 4x21 - 4x22 - 4x23 - 6x24 - 6x25 - 6x26 >= 0
        10b3 - 4x31 - 4x32 - 4x33 - 6x34 - 6x35 - 6x36 >= 0
        10b4 - 4x41 - 4x42 - 4x43 - 6x44 - 6x45 - 6x46 >= 0
        10b5 - 4x51 - 4x52 - 4x53 - 6x54 - 6x55 - 6x56 >= 0
        10b6 - 4x61 - 4x62 - 4x63 - 6x64 - 6x65 - 6x66 >= 0

        x11 + x21 + x31 + x41 + X51 + x61 = 1
        x12 + x22 + x32 + x42 + X52 + x62 = 1
        x13 + x23 + x33 + x43 + X53 + x63 = 1
        x14 + x24 + x34 + x44 + X54 + x64 = 1
        x15 + x25 + x35 + x45 + X55 + x65 = 1
        x16 + x26 + x36 + x46 + X56 + x66 = 1

        b1 + b2 + b3 + b4 + b5 + b6 >= 1

END

INT b1
INT b2
INT b3
INT b4
INT b5
INT b6
INT x11
INT x21
INT x31
INT x41
INT x51
INT x61
```

b) Five items S = { 20, 10, 15, 10, 5} and bin capacity of 20

The result I get is 3 as well, which it also means that I only need 3 exact number of bins to fit all these items, each bin with capacity of 20.

CODE

```
min S
ST
        S >= 1
        S - b1 - b2 -b3 - b4 - b5 = 0

        20b1 - 20x11 - 10x12 - 15x13 - 10x14 - 5x15 >= 0
        20b2 - 20x21 - 10x22 - 15x23 - 10x24 - 5x25  >= 0
        20b3 - 20x31 - 10x32 - 15x33 - 10x34 - 5x35  >= 0
        20b4 - 20x41 - 10x42 - 15x43 - 10x44 - 5x45  >= 0
        20b5 - 20x51 - 10x52 - 15x53 - 10x54 - 5x55  >= 0
        20b6 - 20x61 - 10x62 - 15x63 - 10x64 - 5x65  >= 0

        x11 + x21 + x31 + x41 + X51  = 1
        x12 + x22 + x32 + x42 + X52  = 1
```

```
    x13 + x23 + x33 + x43 + X53  = 1
    x14 + x24 + x34 + x44 + X54  = 1
    x15 + x25 + x35 + x45 + X55  = 1
    x16 + x26 + x36 + x46 + X56  = 1

    b1 + b2 + b3 + b4 + b5  >= 1

END

INT b1
INT b2
INT b3
INT b4
INT b5
INT x11
INT x21
INT x31
INT x41
INT x51
```

```
LP OPTIMUM FOUND AT STEP     28
OBJECTIVE VALUE =    3.00000000

FIX ALL VARS.(    5)  WITH RC >  0.000000E+00

NEW INTEGER SOLUTION OF    3.00000000     AT BRANCH      0 PIVOT      29
BOUND ON OPTIMUM:  3.000000
ENUMERATION COMPLETE. BRANCHES=     0 PIVOTS=      29

LAST INTEGER SOLUTION IS THE BEST FOUND
RE-INSTALLING BEST SOLUTION...

       OBJECTIVE FUNCTION VALUE

       1)      3.000000

 VARIABLE        VALUE          REDUCED COST
       B1         1.000000          1.000000
       B2         1.000000          1.000000
       B3         1.000000          1.000000
       B4         0.000000          1.000000
       B5         0.000000          1.000000
      X11         0.000000          0.000000
      X21         0.000000          0.000000
      X31         1.000000          0.000000
      X41         0.000000          0.000000
      X51         0.000000          0.000000
        S         3.000000          0.000000
      X12         0.000000          0.000000
      X13         1.000000          0.000000
      X14         0.000000          0.000000
      X15         1.000000          0.000000
      X22         1.000000          0.000000
      X23         0.000000          0.000000
      X24         1.000000          0.000000
      X25         0.000000          0.000000
      X32         0.000000          0.000000
      X33         0.000000          0.000000
      X34         0.000000          0.000000
      X35         0.000000          0.000000
      X42         0.000000          0.000000
      X43         0.000000          0.000000
      X44         0.000000          0.000000
      X45         0.000000          0.000000
      X52         0.000000          0.000000
      X53         0.000000          0.000000
      X54         0.000000          0.000000
      X55         0.000000          0.000000
       B6         0.000000          0.000000
      X61         0.000000          0.000000
      X62         0.000000          0.000000
      X63         0.000000          0.000000
      X64         0.000000          0.000000
      X65         0.000000          0.000000
      X16         0.000000          0.000000
      X26         0.000000          0.000000
      X36         0.000000          0.000000
      X46         0.000000          0.000000
      X56         1.000000          0.000000


      ROW   SLACK OR SURPLUS     DUAL PRICES
       2)         2.000000          0.000000
       3)         0.000000         -1.000000
       4)         0.000000          0.000000
       5)         0.000000          0.000000
       6)         0.000000          0.000000
       7)         0.000000          0.000000
       8)         0.000000          0.000000
       9)         0.000000          0.000000
      10)         0.000000          0.000000
      11)         0.000000          0.000000
      12)         0.000000          0.000000
      13)         0.000000          0.000000
      14)         0.000000          0.000000
```

```
min S
ST
        S >= 1
        S - b1 - b2 -b3 - b4 - b5 = 0

        20b1 - 20x11 - 10x12 - 15x13 - 10x14 - 5x15 >= 0
        20b2 - 20x21 - 10x22 - 15x23 - 10x24 - 5x25  >= 0
        20b3 - 20x31 - 10x32 - 15x33 - 10x34 - 5x35  >= 0
        20b4 - 20x41 - 10x42 - 15x43 - 10x44 - 5x45  >= 0
        20b5 - 20x51 - 10x52 - 15x53 - 10x54 - 5x55  >= 0
        20b6 - 20x61 - 10x62 - 15x63 - 10x64 - 5x65  >= 0

        x11 + x21 + x31 + x41 + X51  = 1
        x12 + x22 + x32 + x42 + X52  = 1
        x13 + x23 + x33 + x43 + X53  = 1
        x14 + x24 + x34 + x44 + X54  = 1
        x15 + x25 + x35 + x45 + X55  = 1
        x16 + x26 + x36 + x46 + X56  = 1

        b1 + b2 + b3 + b4 + b5  >= 1

END

INT b1
INT b2
INT b3
INT b4
INT b5
INT x11
INT x21
INT x31
INT x41
INT x51
```