IE400
Principles of Engineering Management

Bin Packing Problem (BPP) Project Report

Deniz Ulusel	21503896
Sabit Gökberk Karaca	21401862
Oğuz Altan	21600966

17.05.2019

1. Introduction

The main goal of this project is to write two programs, an exact solver and a heuristic solver, which aim to minimize the number of bins used, as defined in Bin Packing Problem (BPP). In this problem, we are given n items of different weights w_1, w_2, …, w_n and bins S_1, S_2, …, S_m each of capacity C. The objective is to assign each item to a bin such that number of total used bins is minimized. It is assumed that all items have weights smaller than bin capacity.

In this report, the heuristic algorithm we chose to implement will be explained in detail. Then, some test results for randomly generated instances will be provided to demonstrate and discuss the differences between the heuristic algorithm, which was implemented in python, and the exact algorithm, which was implemented on Xpress Solver. Lastly, we will discuss the findings of this project.

2. Heuristic Algorithm

Two algorithms were utilized in order to come up with a solution for this integer programming problem. The first one is a heuristic algorithm, where a Best Fit approach is implemented and a sub-optimal heuristic solution is found in polynomial running time. This algorithm does not guarantee the exact solution but rather provides an approximation.

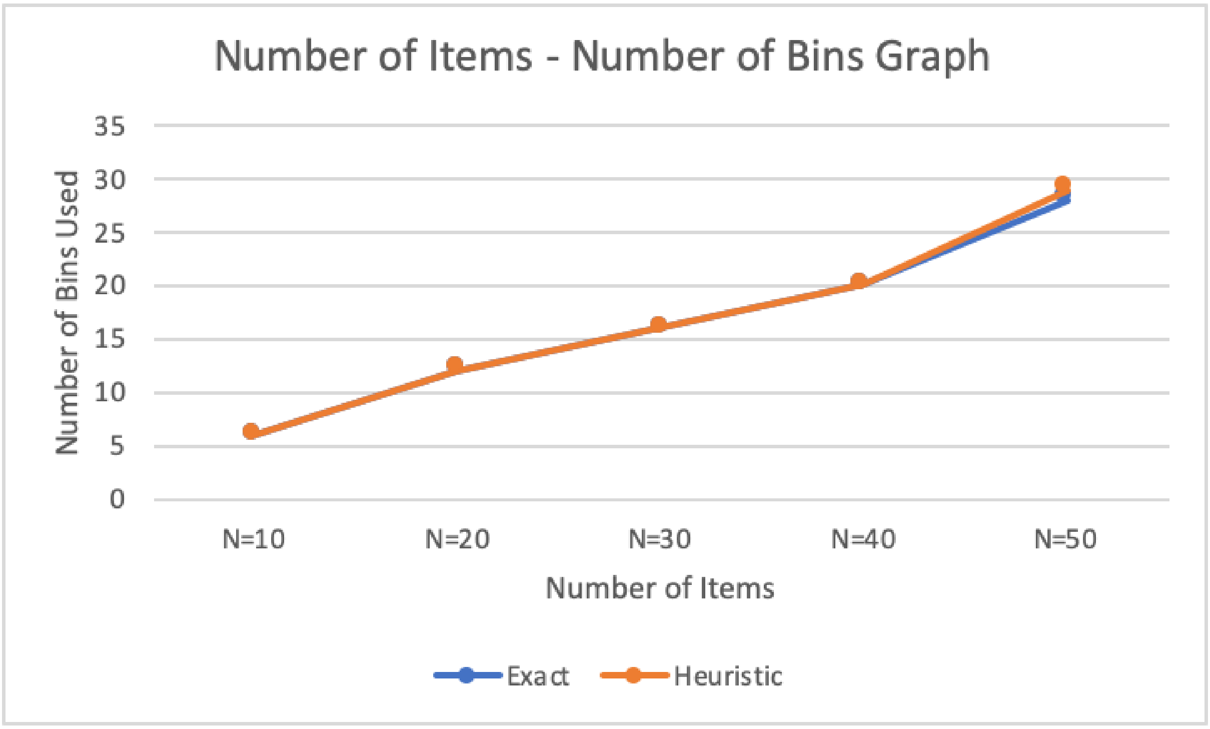
The Best Fit algorithm is an online algorithm, meaning that it processes the inputs one by one, in a serial fashion. The algorithm attempts to place the item into the bin with the smallest remaining space possible. Doing this, it guarantees that the remaining capacity after the insertion of the item into the bins is as small as possible. A random BPP instance generator was also implemented in order to find feasible solutions which would provide practical test cases for out implementation. The full code for mentioned python implementations are provided in the Appendix.

3. Heuristic vs. Exact Algorithm Comparison

3.1. Manipulating the Number of Items

In the table below, the solutions provided by Xpress Solver and Best Fit heuristic algorithm for 5 random generated instances are presented, where N denotes the number of items that will be inserted into bins. We assume capacity of the bins are the same, which is 40 and the number of available bins at the beginning are equal to the number of items. Each cell shows the number of used bins for a specific combination.

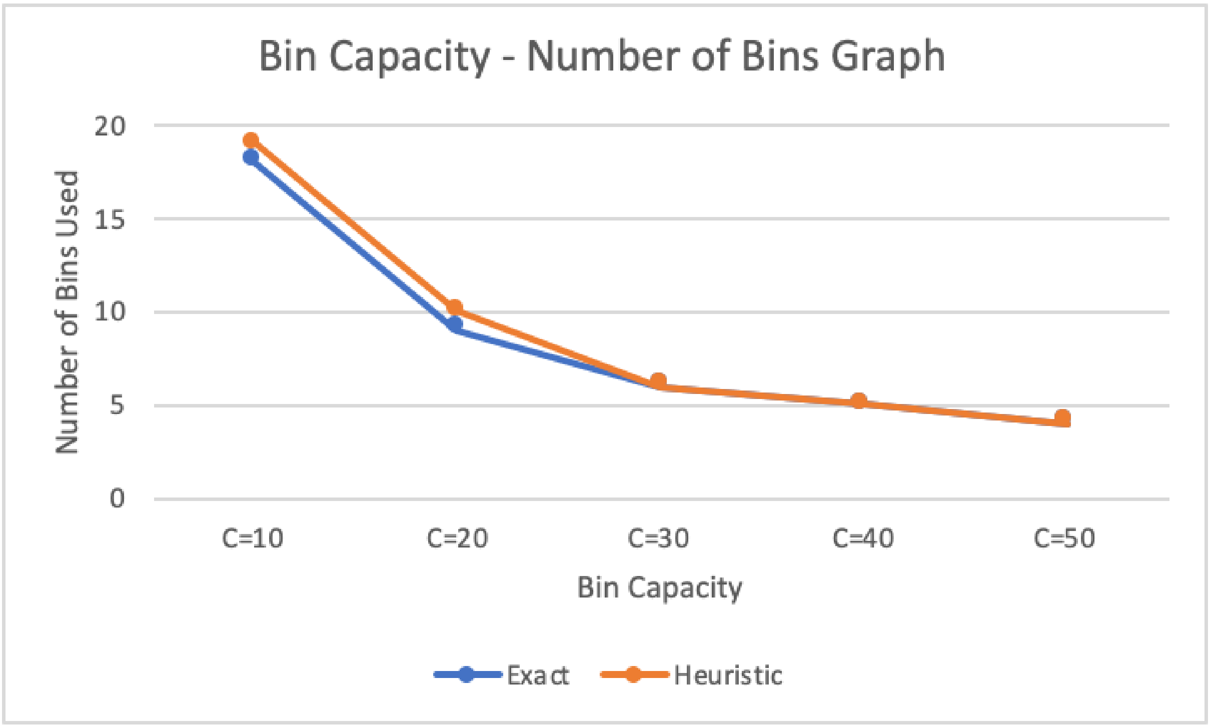
	N=10	N=20	N=30	N=40	N=50
Exact Solution	6	12	16	20	28
Heuristic Solution	6	12	16	20	29



3.2. Manipulating the Bin Capacity

In this part, we assume the number of items, which is 30, are same for 5 randomly generated instances and capacities of bins are different. The same items array is used for all cases. Again, available bins at the beginning are equal to the number of items. Each cell shows the number of used bins for a specific combination.

	C=10	C=20	C=30	C=40	C=50
Exact Solution	18	9	6	5	4
Heuristic Solution	19	10	6	5	4



4. Project Findings

Examining the “Manipulating the Number of Items” section, it can be seen that the solution provided by the heuristic algorithm starts to deviate from the optimal solution provided by the Xpress solver, as the number of items increase. For instance, for the number of items n = 20, the solutions are equal but for n = 50, the solutions differ by 1.

On the other hand, observing the “Manipulating the Bin Capacity” section, as the capacity of the bins increase, the solution provided by the heuristic algorithm approximates to the optimal solution. For the cases C = 10 and C = 20, two solutions are different. However, for the case C = 50, the solutions are exactly the same.

In addition, using Xpress Solver, for number of items n = 50, the exact solution can be obtained but as n exceeds 50, the solver refuses to compute a solution, giving the error message: “?120 Error: Problem has too many rows and columns. The maximum is 5000”.

5. Conclusion

After completing the implementation of two algorithms and testing them for different cases, we have found out that using the exact solution is a better option for small sized data since Xpress Solver can calculate the solution in a reasonable amount of time and gives a more accurate result.

On the other hand, the problem becomes unsolvable on Xpress Solver when the number of items is increased. We have observed that the solver takes a lot of time and does not give a result in a reasonable time. It even crashes for more than 50 items. Because of these reasons, we conclude that it is a better option to use the heuristic algorithm for such cases since it doesn’t deviate too much from the exact solution and it runs much faster for big number of items.

6. Appendix

6.1. Heuristic Solution

best_fit.py

```python
import sys
import random

def bestFit(weight, c):
   num_of_elems = len(weight)
   result = 0

   # Store the remaining capacities for each bin
   remaining_capacity = [c]*num_of_elems

   # Try to insert all elements
   for i in range(0, num_of_elems):
       min_space_left = c+1
       best_bin_index = 0

       # Check each bin for that item
       for j in range(0, result):
           capacity_cond = remaining_capacity[j] >= weight[i]
           min_space_cond = remaining_capacity[j] - weight[i] < min_space_left
           # Insert to the bin if the conditions are satisfied
           if capacity_cond and min_space_cond:
               best_bin_index = j
               min_space_left = remaining_capacity[j] - weight[i]

       # Create a new bin if there is not enough space
       if min_space_left == c+1:
           remaining_capacity[result] = c - weight[i]
           result += 1
       # Update the remaining capacity of the inserted bin
       else:
           remaining_capacity[best_bin_index] -= weight[i]

   return result
```

```python
def gen_random_instance(num_items, c):
    result = []
    for i in range(0, num_items):
        result.append(random.randint(1, c-1))
    return result

c = 10
num_items = 30
weights = gen_random_instance(num_items, c)
print(weights)
print(bestFit(weights, c))
```

6.2. Xpress Solver Exact Solution

bpp.mos

```
model Binpacking
 uses "mmxprs"

 parameters
  NUM_OF_ITEMS = 90
  BIN_CAPACITY = 50
 end-parameters

 declarations
  NUM_OF_BINS = NUM_OF_ITEMS
  INDICES = 1..NUM_OF_ITEMS
  WEIGHTS: array(INDICES) of integer

  BIN_CONTAINS_ITEM: array(INDICES,INDICES) of mpvar ! Either 0 or 1
  IS_BIN_USED: array(INDICES) of mpvar ! Either 0 or 1

  NUM_OF_USED_BINS: linctr ! Objective Function
 end-declarations

 ! Input array
 WEIGHTS :: [8, 9, 5, 4, 6, 9, 6, 8, 9, 5, 3, 6, 4, 7, 4, 4, 1, 7, 4, 7, 2, 9, 9,
2, 6, 6, 8, 8, 2, 3, 8, 9, 5, 4, 6, 9, 6, 8, 9, 5, 3, 6, 4, 7, 4, 4, 1, 7, 4, 7, 2,
9, 9, 2, 6, 6, 8, 8, 2, 3, 8, 9, 5, 4, 6, 9, 6, 8, 9, 5, 3, 6, 4, 7, 4, 4, 1, 7, 4,
7, 2, 9, 9, 2, 6, 6, 8, 8, 2, 3]

 ! Enforce variables to be binary
 forall(b in INDICES,i in INDICES) do
  BIN_CONTAINS_ITEM(b,i) is_binary
 end-do

 ! Enforce variables to be binary
 forall(b in INDICES) do
  IS_BIN_USED(b) is_binary
 end-do

! Make sure that each item is in at most one bin
 forall(i in INDICES) do
  sum(b in INDICES) BIN_CONTAINS_ITEM(b,i) <= 1
 end-do

! Make sure that used bin capacities are smaller than the total capacity of a bin
 forall(b in INDICES) do
  sum(i in INDICES) BIN_CONTAINS_ITEM(b,i)*WEIGHTS(i) <= BIN_CAPACITY *
IS_BIN_USED(b)
 end-do
```

```
! Make sure that each item is contained in at least one bin
 sum(b in INDICES,i in INDICES) BIN_CONTAINS_ITEM(b,i) = NUM_OF_ITEMS

! Set the objective function
 NUM_OF_USED_BINS := sum(b in INDICES) IS_BIN_USED(b)

! Minimize the objective function
 minimize(NUM_OF_USED_BINS)

 writeln("Number of used bins: ", getobjval)
end-model
```