THE BIN PACKING PROBLEM

Harsh Jain, Nikhil Bharat Rai
17BCB0062, 17BEC0449

ABSTRACT-

The BIN PACKING PROBLEM concentrates on placing of objects subject to various constraints (example length, breadth, width, height volume etc.) into a finite number of bins, minimalizing the space wastage. The arrangement of bins is done keeping in mind the fact that one unit of a kind of objects, stays together and is not further split. The algorithm developed finds many applications such as loading containers, filling trucks etc. The problem is an example of an NP-hard problem; hence algorithms are to be developed keeping in mind to provide if not perfect, optimal solution.

INTRODUCTION-

The single dimensional bin packing problem (BPP) can be described as finding a way to assign a list of n items L, each with varying sizes or weights is, into the smallest number of identical bins, each of capacity B. Usually, the item sizes are normalized to the range [0,1] with unit bin capacities. Given as an (integer) linear program, the problem can be stated as follows:

$$\text{Minimize:} \sum_{j=1}^{n} y_j$$

$$\text{Subject to:} \sum_{i=1}^{n} s_i x_{ij} \leq B y_j, \quad \forall\, j \in \{1, ..., n\}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall\, i \in \{1, ..., n\}$$

Where the y_j and x_{ij} are each {0,1} indicator variables denoting whether we need to use a bin j (up to n since we need at most n bins in any solution) and whether item i is included in bin j, respectively. This can be extended to multidimensional problem descriptions in different ways as well as variable sized bins, though even in the single dimensional case it is an NP hard problem. Yet, the problem has basic applications to industry in the context of packing real physical objects in boxes, resources assignment such as VMs to physical machines, and task assignment such as in the context of multiprocessor systems, and has such has generated continued interest over the years. Additionally, it has served as a test bed for various advancements in the techniques used in competitive analysis. It is also closely related to the cutting stock and partitioning problems. In this paper we focus on a survey of several common heuristics for greedy approaches to BPP such as First Fit (FF), Best Fit (BF), Harmonic, and Sum of Squares (SS). We also conduct a brief comparative study of certain aspects of them on our case study's data such as the repacking displacement effect for items that may change in size, and load balance, which is typically consider to be a separate problem. We conclude with an overview of bin packing's uses in some related work and describe future possible areas of exploration.

LITERATURE SURVEY-

The most classical bin packing problem addressed by the GBPP is the BPP. The BPP is the simplest mono-dimensional bin packing problem, which consists in finding the minimum number of bins (all having the same capacity) in order to accommodate a set of items satisfying capacity constraints. A noteworthy pioneering work has been conducted proposed and studied preliminary algorithms, some of them still in the vanguard. In particular, in 1973 Next Fit (NF) algorithm was proposed and proved that its performance ratio is 2. In 1974 it was proposed that First Fit (FF), Best Fit (BF), First Fit Decreasing (FFD), and Best Fit Decreasing (BFD) algorithms and showed that their performance ratios are, 17/10 for FF and BF and 11/9 for FFD and BFD. FFD and BFD are still very used nowadays, sometimes combined with local improvement heuristics.

As a result of evolution in all fields, man is turning back to mother nature for ideas. Bio-Inspired Algorithm is an example of such an idea.

All the newly developed algorithms take into consideration these basic algorithm's with a pinch of new idea so as to minimise the cost and time complexity.

With the purpose of packing the objects in the minimum number of bins and ensuring that the sum of the objects' weights does not exceed the bin capacity in any bin. The BPP is similar to a decision problem called the Partition Problem. In the Partition Problem, we are given n positive integers $b1$, $b2$, …, bn whose sum B=$\sum bi$ n i=1 is even and the purpose is to know if the partition of the set of indices $\{1, 2, …, n\}$ into sets S and T such that $\sum bi$ $i \in S$ = $\sum bi$ $i \in T$, is possible. The partition problem is well-known to be NP-complete. In recent years, different versions of the problem like two and three-dimensional BPP , fragile objects , extendable bins and controllable item sizes have been discussed. On the other hand, the basic problem consists of two main versions named on-line and off-line. In the former, items are appeared without any knowledge about the sequence of them (neither the size nor the number of the items). In the latter, all of the items are given at the initial part of problem solving. Simchi-Levi proved that FFD (First Fit Decreasing), and BFD (Best Fit Decreasing) algorithms have an absolute worst-case ratio of 3 2 . These algorithms' time order is $O(n\log n)$. Zhang and Xiaoqiang provided a linear time constantspace off-line approximation algorithm with absolute worst-case ratio of 3 2 . Moreover, they presented a linear time constant-space on-line algorithm and proved that the absolute worst-case ratio is 3 2 . In 2003, Rudolf and Florian also presented an approximation algorithm for the BPP which has a linear running time and absolute approximation factor of 3 2 . It has been proven that the best algorithm for BPP has the approximation ratio of 3 2 and the time order of $O(n)$, unless $P = NP$. Noori Zehmakan presents two approximation algorithms for this problem. The first one is a 3 2 -approximation algorithm, and the second one is a modified (linear time) version of the FFD algorithm.
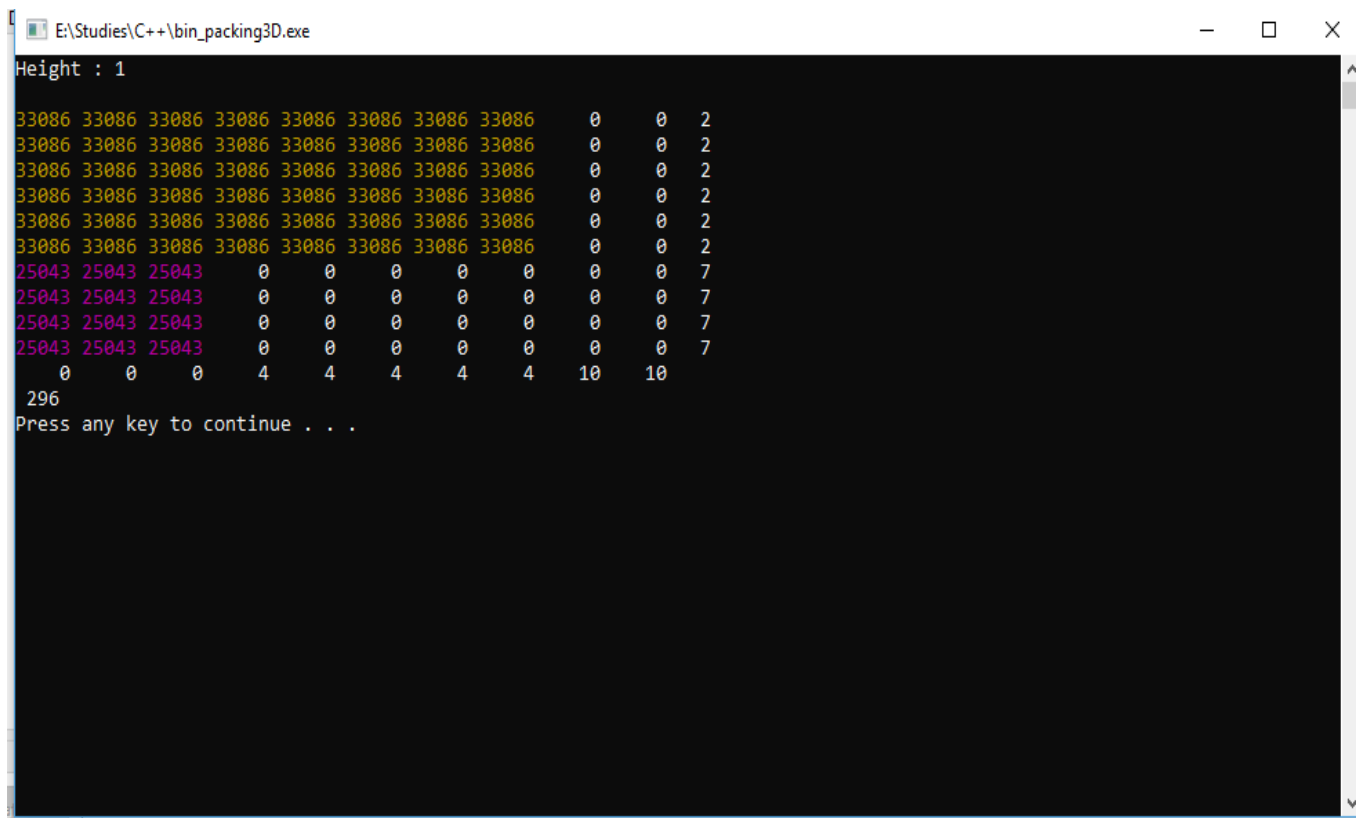
Another variant of the BPP was studied by Li and Chen , where the bins have all the same capacity but they are also characterized by a non-decreasing concave cost function. The authors proved that, for this variant of the problem, FF and BF heuristics have absolute worst case ratio equal to 2, whilst FFD and BFD have absolute worst case ratio equal to 1.5. For this problem, Leung and Li [2008] developed a polynomial time approximation algorithm such that, for any positive , the asymptotic worst case ratio is 1+ . Epstein and Levin [2012] have recently designed an asymptotic fully polynomial time approximation scheme for this problem. In their work, the authors have also proposed a fast approximation algorithm with asymptotic worst case ratio of 1.5.

As an enthusiast, We've not completely relied on the already developed algorithm, instead, I've tried to **develop our own algorithm** by making some changes to the existing algorithm. The algorithm lies under the nest of **Non Greedy** algorithms.

3D BIN PACKING-
My algorithm works in the following manner:

- Accept the dimensions of the parent box.
- Accept all the other boxes dimensions which are to be fit in the box.
- Calculate the individual boxes area and see that no individual area exceeds the area of the parent box.
- Once the quality of boxes is analysed, sort them according to length, breadth, height.
- The biggest box (where height is given the greater preference) among all has the highest priority, the box is supposed to be placed in two different configurations, once lengthwise, other time breadthwise.
- Using a variable check we see whether there is gap available if sufficient to fix the new given box. If it is sufficient the length breadth and height of the box is allocated the bigger box.
- All the other boxes are supposed to be placed one after the other according to space available but only in one configuration.
- In the end we have two different outputs of which we choose the optimal one.

```
E:\Studies\C++\bin_packing3D.exe                                      —    □    ×
Height : 1

33086 33086 33086 33086 33086 33086 33086 33086      0     0    2
33086 33086 33086 33086 33086 33086 33086 33086      0     0    2
33086 33086 33086 33086 33086 33086 33086 33086      0     0    2
33086 33086 33086 33086 33086 33086 33086 33086      0     0    2
33086 33086 33086 33086 33086 33086 33086 33086      0     0    2
33086 33086 33086 33086 33086 33086 33086 33086      0     0    2
25043 25043 25043     0     0     0     0     0      0     0    7
25043 25043 25043     0     0     0     0     0      0     0    7
25043 25043 25043     0     0     0     0     0      0     0    7
25043 25043 25043     0     0     0     0     0      0     0    7
    0     0     0     4     4     4     4     4     10    10
 296
Press any key to continue . . .
```

E:\Studies\C++\bin_packing3D.exe — □ ×

```
Height : 2

33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
25043 25043 25043     0     0     0     0     0     0     0    7
25043 25043 25043     0     0     0     0     0     0     0    7
25043 25043 25043     0     0     0     0     0     0     0    7
25043 25043 25043     0     0     0     0     0     0     0    7
    0     0     0     4     4     4     4     4    10    10
 296
Press any key to continue . . .
```
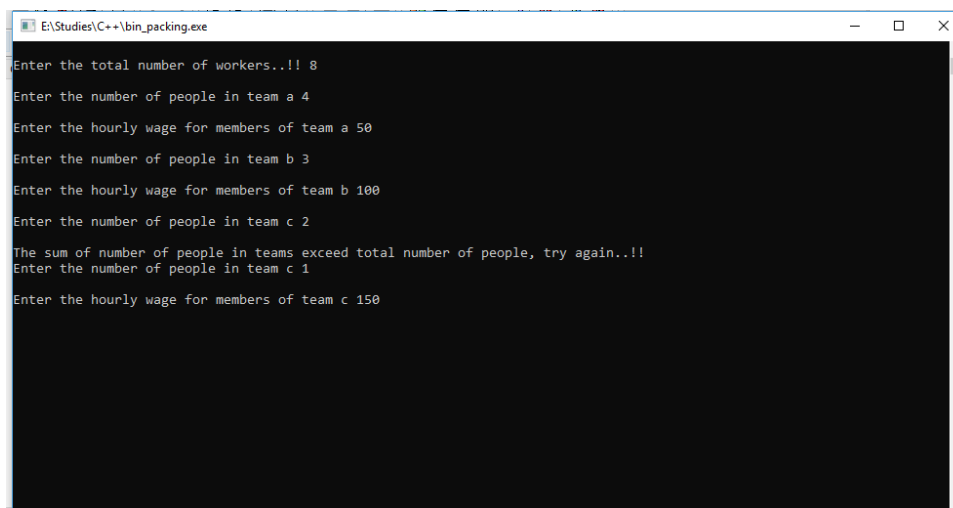
E:\Studies\C++\bin_packing3D.exe — □ ×

```
Height : 3

33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
33086 33086 33086 33086 33086 33086 33086     0     0    2
25043 25043 25043     0     0     0     0     0     0     0    7
25043 25043 25043     0     0     0     0     0     0     0    7
25043 25043 25043     0     0     0     0     0     0     0    7
25043 25043 25043     0     0     0     0     0     0     0    7
    0     0     0     4     4     4     4     4    10    10
 296
Press any key to continue . . .
```

E:\Studies\C++\bin_packing3D.exe — □ ×

```
Height : 4

    0     0     0     0     0     0     0     0     0     0   10
    0     0     0     0     0     0     0     0     0     0   10
    0     0     0     0     0     0     0     0     0     0   10
    0     0     0     0     0     0     0     0     0     0   10
    0     0     0     0     0     0     0     0     0     0   10
    0     0     0     0     0     0     0     0     0     0   10
25043 25043 25043     0     0     0     0     0     0     0    7
25043 25043 25043     0     0     0     0     0     0     0    7
25043 25043 25043     0     0     0     0     0     0     0    7
25043 25043 25043     0     0     0     0     0     0     0    7
    6     6     6    10    10    10    10    10    10    10
 296
Press any key to continue . . .
```
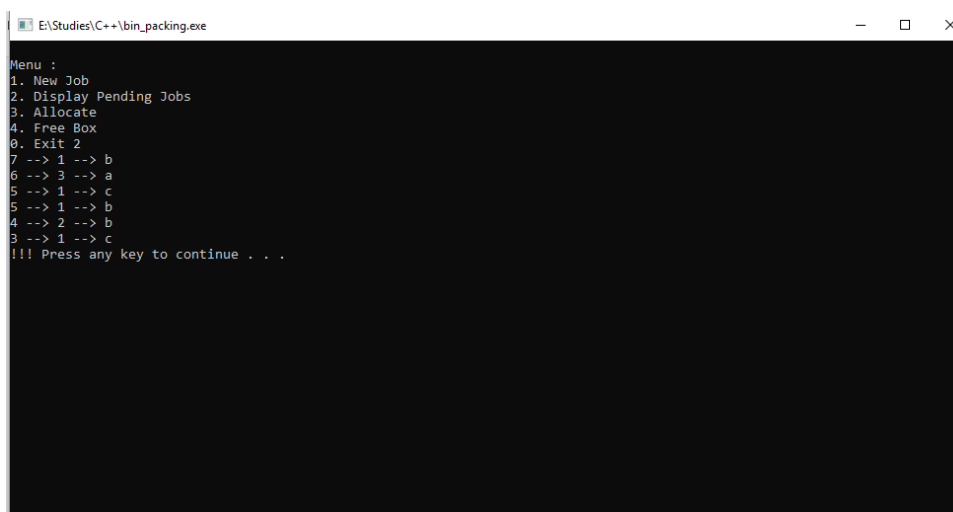
2D BIN PACKING-

My algorithm works in the following manner:

- Accept the Number of teams.
- Accept the number of people in each team.
- Accept the wages of each member of tea
- Once the quality of each worker is analysed, allocate them their respective job.
- It is allocated in such a way that lowest number of team takes the job.
- All the other jobs are supposed to be placed one after the other according to types of job provided but only in one configuration.
- In the end we have two different outputs of which we choose the optimal one.
-

-
-

-

```
E:\Studies\C++\bin_packing.exe                                    —    □    ×
a        63   63   63   63   63   63   Break    0    0    0    0   300
a        63   63   63   63   63   63   Break    0    0    0    0   300
a        63   63   63   63   63   63   Break    0    0    0    0   300
a         0    0    0    0    0    0   Break    0    0    0    0     0
b        71   71   71   71   71   71   Break   71    0    0    0   700
b        51   51   51   51   51    0   Break    0    0    0    0   500
b         0    0    0    0    0    0   Break    0    0    0    0     0
c        51   51   51   51   51   31   Break   31   31    0    0  1200


Total cost : 3300
Press any key to continue . . .
```
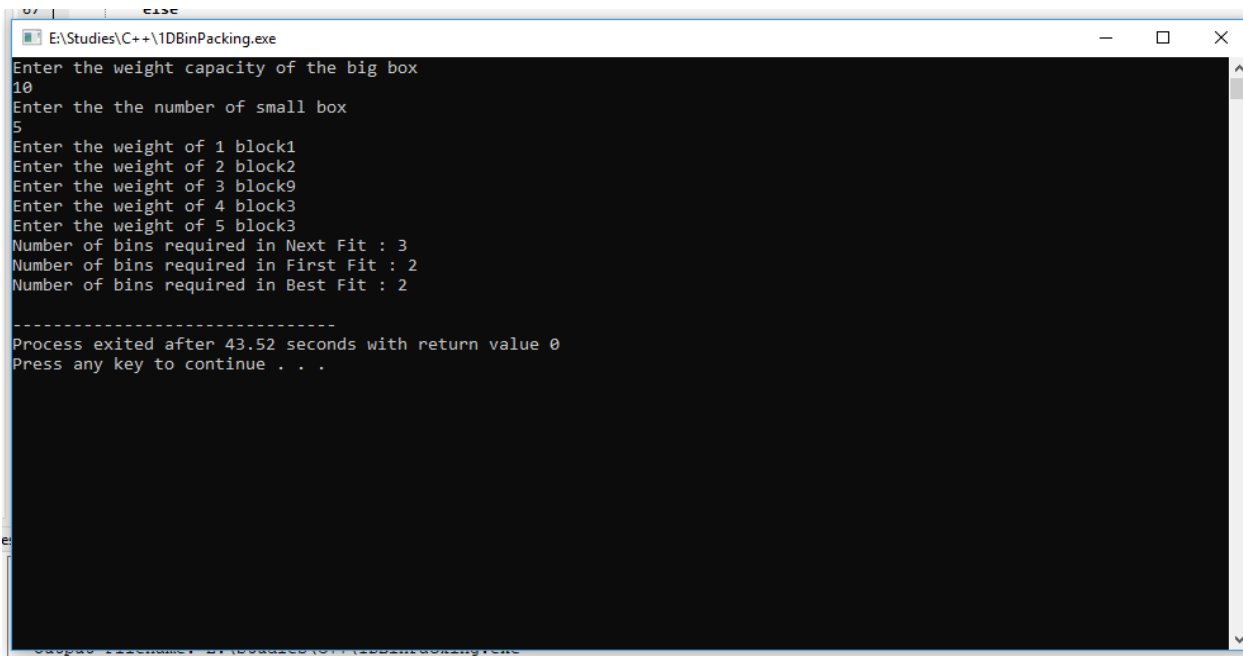
-

1D BIN PACKING-

In 1D bin packing things work a bit differently. Since there is only one parameter (weight) the way it should be placed does not matter.
So we check the lowest number of big boxes required to fit the smaller box.
The capacity that the big box can take is given by the user.
And Output according to algorithm used will be printed.

```
E:\Studies\C++\1DBinPacking.exe                                    —    □    ×
Enter the weight capacity of the big box
10
Enter the the number of small box
5
Enter the weight of 1 block1
Enter the weight of 2 block2
Enter the weight of 3 block9
Enter the weight of 4 block3
Enter the weight of 5 block3
Number of bins required in Next Fit : 3
Number of bins required in First Fit : 2
Number of bins required in Best Fit : 2

--------------------------------
Process exited after 43.52 seconds with return value 0
Press any key to continue . . .
```

REFERENCES-

1. CS787 Project: Bin Packing: A Survey and its Applications to Job Assignment and Machine Allocation by Brian Kroth.
2.https://www.researchgate.net/publication/282655741_A_memory-integrated_artificial_bee_algorithm_for_1-D_bin_packing_problems
3.https://en.wikipedia.org/wiki/Bees_algorithm
4.POLITECNICO DI TORINO SCUOLA INTERPOLITECNICA DI DOTTORATO Doctoral Program in Computer and Control Engineering
5.www.youtube.com