December 2016

Simulation of Online Bin Packing in Practice

Kalpana Rajagopal
University of Nevada, Las Vegas, rajagopa@unlv.nevada.edu

SIMULATION OF ONLINE BIN PACKING IN PRACTICE

by

Kalpana Rajagopal

Bachelor of Technology
Jawaharlal Nehru Technological University
2007

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
December 2016

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

December 8, 2016

This thesis prepared by

Kalpana Rajagopal

entitled

Simulation of Online Bin Packing in Practice

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Wolfgang Bein, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Ajoy K. Datta, Ph.D.
Examination Committee Member

Laxmi Gewali, Ph.D.
Examination Committee Member

Venkatesan Muthukumar, Ph.D.
Graduate College Faculty Representative

ABSTRACT

by

Kalpana Rajagopal

Dr. Wolfgang Bein, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

The bin packing problem requires packing a set of objects into a finite number of bins of fixed capacity in a way that minimizes the number of bins used. We consider the online version of the problem where items arrive over tine and a decision has to be made as soon as an element is available. Online algorithms can be analyzed in terms of competitiveness, a measure of performance that compares the solution obtained online with the optimal offline solution for the same problem, where the lowest possible competitiveness is best. Online processing is difficult due to the fact that unpredictable item sizes may appear. A number of online algorithms such as First-Fit, Next-Fit, Best-Fit, Worst Fit and Harmonic have been proposed and studied in the literature. In this thesis, we examine how well these algorithms perform in practice. Our results that practical performance is differs substantially from the worst case online measure.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Wolfgang Bein for all the support, encouragement and knowledge that I have received from him throughout all stages of my thesis. I am deeply grateful to Dr. Ajoy K Datta for his valuable advice that has made my master's degree possible. I thank Dr.Laxmi Gewali for his valuable time and advice. I would like to thank Dr. Muthukumar Venkatesan for having accepted to serve as a graduate college representative for my thesis and offering constructive comments.

I take this opportunity to thank my colleagues for their kindly support.

I must thank my family who have provided me with their enormous love, support and consideration for my life.

I thank almighty God for his blessings and providing me this opportunity.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

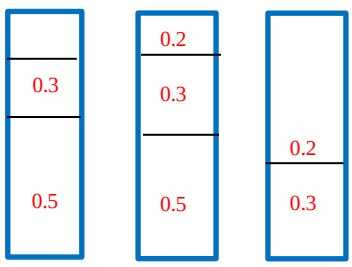
1.1 What is Bin Packing?

The bin packing problem asks for the minimum number k of identical bins of capacity C needed to store a finite collection of weights w_1, w_2, w_3,…, w_n so that no bin has weights stored in it whose sum exceeds the bin's capacity. Traditionally the capacity C is chosen to be 1 and the weights are real numbers which lie between 0 and 1. This problem is known as 1-dimensional bin packing problem.

Bin Packing is considered both when an algorithm has the whole input in advance and when items are coming one by one and each must be packed immediately and irrevocably into a bin without any knowledge of future items. It is important to keep in mind that weights are to be thought of as indivisible objects rather than something like oil or water. For oil one can imagine part of a weight being put into one container and any leftover being put into another container. Here we cannot split an object.

One way to visualize the situation is as a collection of rectangles which have height equal to the capacity C and a fixed width, whose exact size does not matter. When an item is put into the bin it either falls to the bottom or is stopped at a height determined by the weights that are already in the bins.

FIGURE 1: AN EXAMPLE OF THREE BINS: THE FIRST BIN HAS LEVEL OF 0.8, THE SECOND BIN IS FULL AND THE THIRD BIN HAS LEVEL 0.4



Bin packing problem is an optimization problem with incomplete information. In Bin Packing, a sequence of items of size up to 1 arrives to be packed into bins of unit capacity which lie between 0 and 1. The goal is to minimize the number of bins used. In other words, we need to partition items into the minimum number of bins such that the sum of sizes of items in each bin is at most one.

1.2 Background of Bin Packing

Bin packing problem is one of the oldest and most thoroughly studied problems in computer science. Bin packing problem is NP – hard optimization problem. One way to deal with this problem is to find an algorithm which gives a worse solution in polynomial time. The field of approximation algorithms deals with such problem. The bin packing problem was proposed by Johnson [1] in early seventies and it was studied extensively since then in both online and offline settings. In fact, bin packing was the first problem being investigated properly in both of these settings and moreover Johnson in 1973 introduced comparing the quality of the solution by an online algorithm against an optimal strategy.

1.3 Online vs Offline Computation

Generally, online computation deals with situations when the input is revealed to the algorithm as the time goes on and the algorithm must deal with an incoming part of the input immediately after it comes without any knowledge of the future [3] [4]. On the other hand, in the offline computation an algorithm knows the whole input before making any decision.

An offline algorithm simply repacks everything each time an item arrives, where as in online algorithm each item is packed immediately and irrevocably without any knowledge of the next items in input sequence. Offline bin packing is not easy if we have only polynomial amount of time. Packing large items is difficult with online bin packing.

1.4 Competitive Analysis

Competitive analysis is a type of worst-case analysis. Analyzes the performance of an algorithm by comparing it with the best optimal solution. In this thesis, we compare the performance of different online

algorithms with the best optimal solution. This comparison is done by using a ratio called 'Competitive Ratio'. The best online algorithm for bin packing is the one that has the lowest possible competitive ratio and this ratio is at least 1.

Competitive Ratio

In the case of bin packing, the standard metric for worst-case performance is competitive ratio. For a given list of items and an algorithm A, let A(L) be the number of bins used when algorithm A is applied to list L, let OPT(L) denote the optimum number of bins for a packing of L, and $R_A(L)$ denote the absolute worst-case performance (competitive) ratio for algorithm A [3].

$$R_A(L) = A(L) / OPT(L)$$

A disadvantage of competitive analysis is that it sometimes gives an unrealistically bad impression on an algorithm, in that algorithms that perform well in practice have a high competitive ratio. Furthermore, it sometimes fails to differentiate between algorithms whose performance is observed to be very different.

1.5 Applications of Bin Packing

Bin packing problem has many real-world applications such as filling up containers, loading trucks with weight capacity, creating file backups in media [5], scheduling tasks with known execution times on a set of identical machines, storing files on disks, cutting stock problem. Online bin packing arises in many real-world problems. For example, we put goods of different weights into trucks or containers with the same weight limit and we want to use as few of them as possible, while items are packed immediately as they arrive without a possibility to change their assignment later [10]. Of course, one must simplify the problem; mainly we are assuming that the weight is more restrictive than the size of containers. Other applications are: assigning newspaper articles into columns, adding network packets of different sizes into larger blocks of the same size, assigning commercials into breaks on a television station, etc.

CHAPTER 2

OFFLINE BIN PACKING

In the classical offline bin packing problem, an algorithm receives items of size $x_1, x_2, \ldots, x_n \in (0,1]$. We have infinite number of bins, each with capacity 1, and every item is to be assigned to a bin. Further, the sum of the sizes of the items assigned to any bin cannot exceed its capacity. A bin is empty if no item is assigned to it, otherwise, it is used. The goal of the algorithm is to minimize the number of used bins. This is one of the classical NP-hard problems and heuristic and approximation algorithms have been investigated thoroughly.

In both the offline and online problems the algorithm has access to the bins in arbitrary order. The more restricted version of offline algorithms for bin packing is sequential algorithms. In this algorithm items arrive one by one but in each round the algorithm have only two possible choices: assign the given item to the open bin or to the next empty bin, and items cannot be assigned anymore to closed bins.

2.1 Next Fit Offline Algorithm

A simple offline algorithm called Next Fit is used to minimize the number of bins. Next Fit processes the items one at a time in the same order as they are given in input. The first item a_1 is placed into bin B_1. Let B_j be the last used bin, when the algorithm considers item a_i: Next Fit assigns a_i to B_j if it has enough room; otherwise a_i is assigned to a new bin B_{j+1}.

Theorem: Consider any instance x of the Minimum Bin Packing problem, algorithm Next Fit computes a solution such that: $m_{NF}(x) \leq 2m^*(x)$ [6].

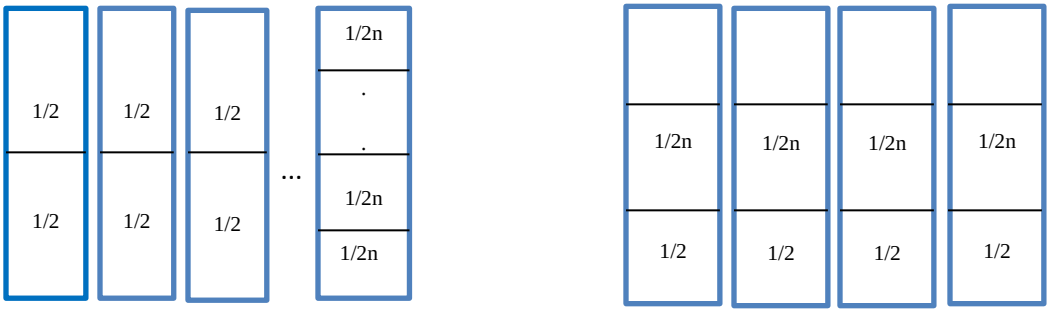
Proof: Firstly, 'A' denotes the sum of all the item sizes in the bin. In the next fit algorithm, only one bin (last used bin) is kept open and when an item doesn`t fit, that bin is closed and a new bin is opened. So at any given time only one bin is open and once the bin is closed it cannot be opened. Since the sum of the items of any two consecutive bins is always greater than 1, the number of bins used by Next fit algorithm

is less than 2[A]. This is because on an average the bins are more than half full. On the other hand, the optimal solution uses bins which are at least the total size of the items (A). So, $m^*(x) \geq [A]$. Hence, we have $m_{NF}(x) \leq 2m^*(x)$.

2.2 Is this analysis tight?

Sequence: Consider an instance of 4n items and the order of the items are as follows: {1/2, 1/2n, 1/2, 1/2n,…, 1/2, 1/2n} (each pair is repeated 2n times). This example is taken from the paper written by D.S. Johnson [1]. Fig (a) represents the optimal solution where the 2n items of size ½ are filled in n bins and the remaining 2n items of size 1/2n are filled in a single bin. Hence the optimal requires n+1 bins to fill the given sequence. Fig (b) represents the approximate solution of Next Fit algorithm where 2 items of size 1/2 and 1/2n are filled in each bin. Hence the next fit requires 2n bins to fill the given sequence.

FIGURE 2. EXAMPLE FOR NEXT FIT OFFLINE BIN PACKING ALGORITHM



An obvious weakness of Next Fit is that it tries to assign an item only to the last used bin. This leads to a problem where the performance of NF is 2 times worse than the optimal performance. To overcome this weakness, we suggest a new algorithm called First Fit, which processes items in the input order according to the following rule: item a_i is assigned to the first used bin that has enough available space to include it; if no bin contain a_i, a new bin is opened.

2.3 First Fit Offline Algorithm

First Fit algorithm has a better performance than Next Fit: it finds a solution this is at most 70% far away from the optimal solution. It can be shown that First Fit finds a solution with the value $m_{FF}(x)$ such that $m_{FF}(x) \leq 1.7m^{*}(x) + 2$ [1] [6].

An even better algorithm for Offline Bin Packing is First Fit Decreasing (FFD)

2.4 First Fit Decreasing Offline Algorithm

This algorithm first sort's items in non-increasing order with respect to their size and then processes items as First Fit.

Theorem: First Fit Decreasing is a 3/2-approximation for Bin Packing [2]. The algorithm runs in $O(n^2)$ time.

Proof: Let k be the number of non-empty bins of the assignment a found by First Fit Decreasing and let k^{*} be the optimal number [7]. Consider bin number j = [2/3k]. If it contains an item i with $s_i > 1/2$, then each bin $j' < j$ did not have space for item i. Thus, j' was assigned an item i' with $i' < i$. As the items are considered in non-increasing order of size we have $s_{i'} \geq s_i > 1/2$. That is, there are at least j items of size larger than 1/2. These items need to be placed in individual bins. This implies $k^{*} \geq j \geq 2/3\ k$.

Otherwise, bin j and any bin $j' > j$ does not contain an item with size larger than 1/2. Hence the bins j, j + 1, 2, ..., k contains at least $2(k - j) + 1$ items, none of which fits into the bins 1, 2, …, j − 1. Thus, we have

$$s(I) > \min\{j - 1, 2(k - j) + 1\}$$

$$\geq \min\{[2/3k] - 1, 2(k - (2/3k + 2/3)) + 1\}$$

$$= [2/3k] - 1$$

and $k * \geq s(I) > [2/3k] - 1$. This even implies

$$k^{*} \geq [2/3k] \geq 2/3\ k$$

and hence the claim.

There exists an instance of the bin packing problem for which the performance of First Fit Decreasing is 11/9 times away from optimum. By means of a detailed case analysis, the bound given in the above theorem can be substituted by $11m^{*}(x)/9 + 7/9$.

Algorithm for First-Fit Decreasing

Input: Set I of n positive rationales less than or equal to 1;

Output: Partition of I in subsets of unitary weight;

begin

Sort elements of i in non-increasing order;

(* Let (a_1, a_2, ----, a_n) be the obtained sequence *)

for i: = 1 to n do

if there is a bin that can contain a_i then

Insert a_i into the first such bin

else

Insert a_i into a new bin;

Return the partition

end

CHAPTER 3

ONLINE BIN PACKING

In the online bin packing problem, a sequence of items with sizes in the interval (0,1] arrive one by one and need to be packed into bins, so that each bin contains items of total size at most 1. Once an item is packed in a particular bin it cannot be moved. Each item must be irrevocably assigned to a bin before the next item becomes available. The algorithm has no knowledge about future items. There is an unlimited supply of bins available, and the goal is to minimize the total number of used bins.

The online processing is difficult because unpredictable item sizes may appear. In general, the performance of an online bin packing algorithm is substantially affected by the permutation of items in a given list.

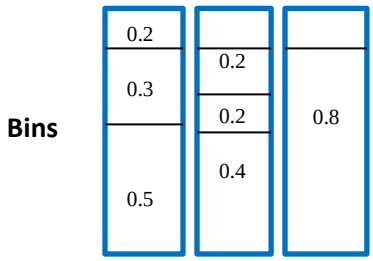
There are several well-known and often used online algorithms for bin packing. Now let us discuss some of the online algorithms for bin packing.

3.1 First Fit Bin Packing Algorithm

If we are willing to keep bins open in the hope that we will be able to fill empty space with items in the list, we will typically use fewer bins. The simplest way to evaluate this idea is known as First Fit bin packing algorithm [11]. In First Fit, we place the next item in the list into the first bin which has not been completely filled into which it will fit. When bins are filled completely they are closed and if an item will not fit into any currently open bin, a new bin is opened.

Example: Items - 0.5, 0.3, 0.4, 0.8, 0.2, 0.2, 0.2

FIGURE 3. EXAMPLE FOR FIRST FIT ONLINE BIN PACKING ALGORITHM



Algorithm

Consider bins b_j where j ϵ (1,2, … , n)

Consider an instance x containing items a_i where i ϵ (1, 2, … , n)

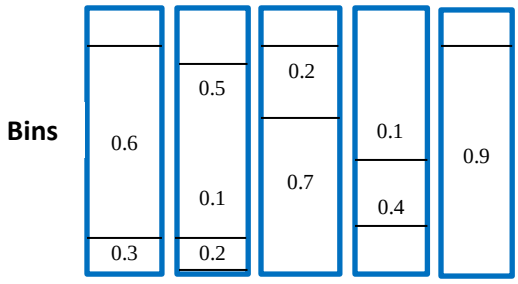
```
Begin
    For i:= 1to n do
    For j:= 1 to n do
    If item ai can fit in the bin bj
        then
            Insert ai into the bin
            Break; //exit for j loop
    // continue for i loop
End
```

3.2 Next Fit Bin Packing Algorithm

Next Fit is one of the most basic online algorithms. The idea behind this algorithm is to open a bin and place the items into it in the order they appear in the list [11]. If an item on the list will not fit into the open bin, we close the bin permanently and open a new one and continue packing the remaining items in the list. Of course, if some of the consecutive weights on the list exactly fill a bin, the bin is then closed and a new bin opened.

Example: Items - 0.3, 0.6, 0.2, 0.1, 0.5, 0.7, 0.2, 0.4, 0.1, 0.9

FIGURE 4. EXAMPLE FOR NEXT FIT ONLINE BIN PACKING ALGORITHM



Algorithm

Consider an instance x containing items a_i where i ϵ (1, 2, … , n) and number of bins b ‹-- 1

Begin

For i: = 1 to n do

If the item a_i can fit in the opened bin then

Insert a_i into the bin

Insert the item a_i into a new bin

b: = b+1

Return b

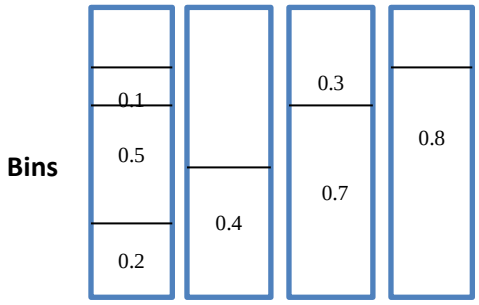
End

3.3 Best Fit Bin Packing Algorithm

Best Fit packs the input item according to the following rule: while trying to pack item a_i, the best fit algorithm assigns the item to the bin whose empty space is minimum [11]. If the item a_i is unable to fit in any of the opened bins, then a new bin is opened to pack that item a_i.

Example: Items - 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8

FIGURE 5. EXAMPLE FOR BEST FIT ONLINE BIN PACKING ALGORITHM



Algorithm

Consider bins b_j where j ϵ (1,2 … , n)

Consider an instance x containing items a_i where i ϵ (1,2 … , n) and no of bins b ‹— 1

Begin

For i := 1 to n do

Sort bins b_j in decreasing order such that the bin with minimum space available is placed first. Let the sorted sequence be { B_1,B_2 …. ,B_n}

For k:=1 to n do

if the item a_i can fit in the bin B_k then

Insert a_i into the bin

break; // exit for k loop

//continue for i loop

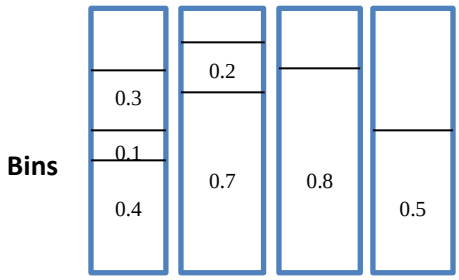
End

3.4 Worst Fit Bin Packing Algorithm

Worst Fit packs the input item according to the following rule: while trying to pack item a_i, the worst fit algorithm assigns the item to the bin whose empty space is maximum. If the item a_i is unable to fit in any of the opened bins, then a new bin is opened to pack that item a_i.

Example: Items – 0.4, 0.7, 0.1, 0.3, 0.8, 0.2, 0.5

FIGURE 6. EXAMPLE FOR WORST FIT ONLINE BIN PACKING ALGORITHM



Algorithm

Consider bins b_j where j ϵ (1,2 … , n)

Consider an instance x containing items a_i where i ϵ (1,2 … , n) and no of bins b ‹— 1

Begin

For i := 1 to n do

Sort bins b_j in decreasing order such that the bin with maximum space available is placed first. Let the sorted sequence be { $B_1, B_2 \ldots, B_n$}

For k:=1 to n do

if the item a_i can fit in the bin B_k then

Insert a_i into the bin

break; // exit for k loop

//continue for i loop

End

3.5 Harmonic Fit Bin Packing Algorithm

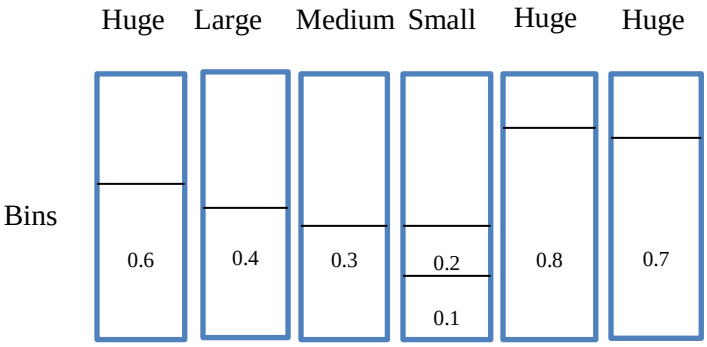
Under online algorithms for bin packing problem, we have another algorithm based on non-uniform partitioning of interval (0, 1] into M sub-intervals [8]. Consider an instance L = {it_1, it_2, it_3,… , it_n}, where $0 < s(it_i) \leq 1$, $s(it_i)$ denotes the size of item it_i in the given instance L. In this algorithm, the interval (0, 1] is partitioned into harmonic sub intervals I^M = {(0, 1/M], (1/M, 1/M-1],…., (1/2, 1]} where M is a positive integer. Now each item it_i is classified and put in one of these sub intervals based on their size. An item it_i is called I_k item, if the item size is in the interval I_k = (1/k+1, 1/k], k>1. If the item size is in the interval I_M= (0, 1/M], then the item is called I_M item. In this manner, all the items in the instance or the sequences are classified. So, the I_k filled bin (bin with all I_k items) packs exactly k items irrespective of the actual sizes of the items. Using this background, we discuss about Algorithm Harmonic.

This algorithm opens an active bin for each type i.e., one bin of I1 (Huge) type items, one bin of I2 (Large) type items, one bin of I3 (Medium) type items and one bin of I4 (Small) type items.. Hence a total of M bins are active at any given time (since M sub intervals) [12]. When an item iti belonging to sub-interval Ik (Ik item) arrives, it is packed in the corresponding active bin, if that bin is filled and has no enough space to pack item Ik then it is closed and a new bin is open for that sub-interval items. This harmonic algorithm is independent of the arriving order of the items. A disadvantage with this algorithm is when items of size > 1/2 are packed then one bin per item is used resulting in wasting a lot of free space in each single bin.

Example: Assume that the harmonic sub intervals are Huge = $(\frac{1}{2}, 1]$, Large = $(\frac{1}{3}, \frac{1}{2}]$, Medium = $(\frac{1}{4}, \frac{1}{3}]$ and Small = $(0, \frac{1}{4}]$.

Items: 0.6, 0.1, 0.4, 0.8, 0.2, 0.3, 0.7

FIGURE 7. EXAMPLE FOR HARMONIC FIT ONLINE BIN PACKING ALGORITHM



Algorithm

For a given value of M and the set of items L = {it_1, it_2, … , it_n}

Step 1: Partition the interval (0, 1] for the given M into subintervals as given below:

(0, 1/M], (1/M, 1/M-1], …. , (1/2, 1]

Step 2: Assign each item it_n to the open bin of that type (the size of it_n fits into the corresponding subinterval).

Step 3: If an item does not fit into the corresponding bin, then close it and open a new one.

Step 4: Calculate the total number of bins used of each type.

CHAPTER 4

LOWER BOUNDS

4.1 Next Fit Algorithm

Given an instance x of online bin packing, the algorithm next fit returns a result with value $m_{NF}(x)$ such that $m_{NF}(x)/m_{OPT}(x) \leq 2$ where $m_{OPT}(x)$ denotes the optimal solution for an instance x. This means the competitive ratio of next fit is 2. Here is the worst example to prove the competitive ratio for next fit.

Example: Input sequence of arbitrary length n is needed.

Sequence: k items of size 0.9 and k items of size 0.2. These can be packed in n bins by optimal.

As the sizes 0.9 and 0.2 both together cannot occupy in one bin, the items size with 0.9 are packed in n bins and the items size with 0.2 are packed in n bins. So altogether next fit packs these items in 2n bins.

So the competitive ratio for next fit is 2n/n = 2. In fact, first fit and best fit algorithms have a better competitive ratio than next fit when it comes to online algorithms for bin packing.

4.2 First Fit and Best Fit Algorithm

Given an instance x of Online Bin Packing, the algorithm First Fit returns a result with value $^{m}{}_{FF}(x) \leq 1.7$ $m_{OPT}(x) + 2$ where $m_{OPT}(x)$ denotes the optimal solution for an instance x [9]. The numeric "2" represents the additive constant. The competitive ratio for both first fit and best fit algorithms is 1.7. Here is the example to prove the competitive ratio for First Fit and Best Fit.

Example: Input sequence of arbitrary length n is needed

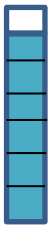
Input Sequence: 6n items of size 0.15, 6n items of size 0.34, 6n items of size 0.51

These items can be packed into 6n bins by optimal.

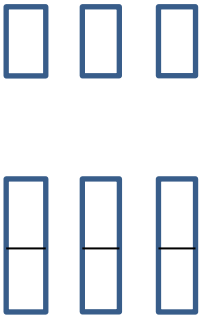
How do First Fit and Best Fit pack this input?

FIGURE 8. EXAMPLE FOR LOWER BOUND FOR FIRST FIT AND BEST FIT ALGORITHMS

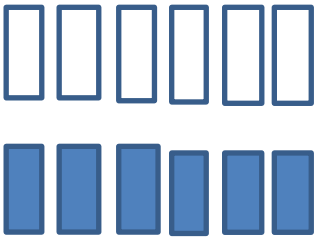
6n items of size 0.15 → packed in n bins. All these bins are 0.9 full



6n items of size 0.34 → packed in 3n bins. All these bins are 0.68 full



6n items of size 0.51 → packed in 6n bins.



- Total 10n bins are used.
- Lower bound is 10n/6n = 5/3 = 1.7

Though First Fit and Best Fit are better than Next Fit, the worst-case performance is the same for all the three algorithms. A better approximation algorithm is obtained by observing that the worst performance for First Fit and Best Fit seems to occur when smaller items appear before larger items in a given instance.

4.3 Harmonic Fit Algorithm

To find out competitive ratio we need input sequence of arbitrary length n.

Sequence: k items of size 0.50001, k items of € (epsilon). These items can be packed into k bins by optimal.

How does Harmonic Fit packs this input?

- Harmonic Fit packs all the k items of size 0.50001 in k bins and all the k items of epsilon is packed in k/2 bins. So, the harmonic fit packs the input in (k/2 + k) bins.

So the competitive ratio of harmonic to optimal is 3/2 = 1.5

CHAPTER 5

SIMULATION RESULTS

5.1 Next Fit Algorithm

This is a bounded-space online algorithm in which the only partially-filled bin that is open is the most recent one to be started. The first item is assigned to bin1. Items 2, …., n are then considered by increasing indices: each item is assigned to the current bin, if it fits; otherwise, it is assigned to a new bin, which becomes the current one. The time complexity of the algorithm is clearly O (n). It is easy to prove that, for any instance x of bin packing problem, the solution value $m_{NF}(x)$ provided by the algorithm satisfies the bound

$$m_{NF}(x) \leq 2\, m_{OPT}(x)$$

where $m_{OPT}(x)$ denotes the optimal solution value. Furthermore, there exist instances for which the ratio $m_{NF}(x)/m_{OPT}(x)$ is arbitrarily close to 2, i.e., the competitive ratio of NF is 2. The tables below show the closest competitive ratio and worst values.

TABLE 1. NEXT FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{10} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1	1
0.0 – 0.2	N/A	1
0.0 – 0.5	N/A	1
0.0 – 1.0	1.044	1.2

TABLE 2. NEXT FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{20} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	N/A	1
0.0 – 0.2	N/A	1
0.0 – 0.5	1.016	1.2
0.0 – 1.0	1.107	1.286

TABLE 3. NEXT FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{50} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	N/A	N/A
0.0 – 0.2	1	1
0.0 – 0.5	1.13	1.18
0.0 – 1.0	1.28	1.33

TABLE 4. NEXT FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{100} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	N/A	1
0.0 – 0.2	1	1
0.0 – 0.5	1.13	1.19
0.0 – 1.0	1.27	1.3

TABLE 5. NEXT FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{1000} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1.0201	1.0208
0.0 – 0.2	1.05	1.06
0.0 – 0.5	1.16	1.17
0.0 – 1.0	1.3	1.31

5.2 First Fit Algorithm

There is no restriction for First Fit Bin Packing. All partially-filled bins are considered as possible destinations for an item to be packed. We place an item in the first bin into which it will fit, the next item is placed in the bin which has lowest index otherwise a new bin is opened. The lower bound for First Fit is

$$m_{FF}(x)/m_{OPT}(x) \leq 5/3$$

The above lower bound occurs when the items are sorted in increasing order by size. The lower bound 5/3 is obtained by following sequence of items. The sequence has 6n items of size 0.15, 6n items of size 0.34 and 6n items of size 0.51.

Note that still worse examples and arbitrarily close to the lower bound examples can be devised using the above idea.

TABLE 6. FIRST FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{10} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	N/A	N/A
0.0 – 0.2	N/A	1
0.0 – 0.5	1.365	1.5
0.0 – 1.0	1.3	1.333

TABLE 7. FIRST FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{20} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1	1
0.0 – 0.2	1.315	1.5
0.0 – 0.5	1.27	1.33
0.0 – 1.0	1.13	1.16

TABLE 8. FIRST FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{50} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1.155	1.5
0.0 – 0.2	1.16	1.25
0.0 – 0.5	1.1	1.11
0.0 – 1.0	1.09	1.11

TABLE 9. FIRST FIT - SEQUENCE OF ITEMS WITH SIZES $S_1, S_2, \ldots\ldots, S_{100}$ IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1	1
0.0 – 0.2	1.07	1.125
0.0 – 0.5	1.04	1.05
0.0 – 1.0	1.06	1.08

TABLE 10. FIRST FIT - SEQUENCE OF ITEMS WITH SIZES $S_1, S_2, \ldots\ldots, S_{1000}$ IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1.003	1.020
0.0 – 0.2	1.007	1.01
0.0 – 0.5	1.0093	1.0095
0.0 – 1.0	1.023	1.027

Although this simple scheme for worst case examples is insufficient to characterize the worst-case behavior of First Fit, there are other algorithms for which it is more relevant. Moreover, the scheme is correct in suggesting that for First Fit to behave at its worst, the instance to which it is applied must contain relatively large items. As with Next Fit, First Fit's worst-case behavior improves dramatically as the size of the largest item declines. Moreover, it maintains its advantage over Next Fit in such situations, although the size of its advantage depends on the precise value of the item and shrinks with the size of the largest item.

5.3 Best Fit Algorithm

Now we know how crucial the packing rule used for First Fit to the improved worst-case behavior. Performance of Best Fit and First Fit is similar, but BF assigns an arriving item to the bin in which it fits best. Place the items in the order in which they arrive. Place the next item into that bin which will leave the least room left over after the item is placed in the bin. If it does not fit in any bin, start a new bin. The time complexity of BF is O (n log n). The lower bound of best fit is

$$m_{BF}(x)/m_{OPT}(x) = 5/3$$

This means best fit never uses more than 5/3 = 1.7 bins. Note that still worse examples and arbitrarily close to the lower bound examples can be devised using the above idea.

TABLE 11. BEST FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{10} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	N/A	N/A
0.0 – 0.2	1	1
0.0 – 0.5	1.195	1.5
0.0 – 1.0	1.29	1.33

TABLE 12. BEST FIT - SEQUENCE OF ITEMS WITH ARRAY SIZES S_1, S_2 ,……, S_{20} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1	1
0.0 – 0.2	1.145	1.5
0.0 – 0.5	1.237	1.33
0.0 – 1.0	1.14	1.16

TABLE 13. BEST FIT - SEQUENCE OF ITEMS WITH ARRAY SIZES S_1, S_2 ,……, S_{50} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1	1
0.0 – 0.2	1.178	1.33
0.0 – 0.5	1.105	1.11
0.0 – 1.0	1.08	1.11

TABLE 14. BEST FIT - SEQUENCE OF ITEMS WITH ARRAY SIZES S_1, S_2 ,……, S_{100} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1.08	1.25
0.0 – 0.2	1.05	1.11
0.0 – 0.5	1.05	1.05
0.0 – 1.0	1.05	1.05

TABLE 15. BEST FIT - SEQUENCE OF ITEMS WITH ARRAY SIZES S_1, S_2 ,……, S_{1000} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1.004	1.02
0.0 – 0.2	1.007	1.01
0.0 – 0.5	1.007	1.01
0.0 – 1.0	1.01	1.02

5.4 Worst Fit Algorithm

Initially Worst Fit was presented as offline heuristics, but in fact it is online algorithm which process the items as a list. Worst Fit places the item into the most empty bin. This heuristic has the effect of spreading the slack (empty space) over the bins used. This algorithm might be useful if it is desirable to pack the bins with approximately the same weight or fill them with items of approximately the same value. Worst Fit is better than Next Fit. The lower bound of worst fit is

$$m_{wF}(x)/m_{OPT}(x) = 2$$

This means worst fit never uses more than 2 bins. Note that still worse examples and arbitrarily close to the lower bound examples can be devised using the above idea.

TABLE 16. WORST FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{10} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	N/A	N/A
0.0 – 0.2	1.1	2
0.0 – 0.5	1.68	2
0.0 – 1.0	1.34	1.5

TABLE 17. WORST FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{20} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	0.64	2
0.0 – 0.2	1.795	2
0.0 – 0.5	1.25	1.3
0.0 – 1.0	1.25	1.28

TABLE 18. WORST FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{50} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1.5	1.5
0.0 – 0.2	1.23	1.25
0.0 – 0.5	1.15	1.2
0.0 – 1.0	1.2	1.26

TABLE 19. WORST FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{100} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1.21	1.25
0.0 – 0.2	1.11	1.12
0.0 – 0.5	1.12	1.14
0.0 – 1.0	1.16	1.18

TABLE 20. WORST FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{1000} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1.04	1.04
0.0 – 0.2	1.05	1.05
0.0 – 0.5	1.09	1.1
0.0 – 1.0	1.146	1.158

5.5 Harmonic Fit Algorithm

Harmonic algorithm is one of the best of the known online heuristics for the classical bin packing problem, which was developed by C.C. Lee and D.T. Lee. They proved that the asymptotic worst-case performance ratio of this algorithm is 1.6901….

Note that still worse examples and arbitrarily close to the lower bound examples can be devised.

TABLE 21. HARMONIC FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{10} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1	1
0.0 – 0.2	1	1
0.0 – 0.5	1.45	1.5
0.0 – 1.0	1.25	1.75

TABLE 22. HARMONIC FIT - SEQUENCE OF ITEMS WITH SIZES $S_1, S_2, \ldots\ldots, S_{20}$ IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1	1
0.0 – 0.2	1.46	1.5
0.0 – 0.5	1.27	1.4
0.0 – 1.0	1.423	1.428

TABLE 23. HARMONIC FIT - SEQUENCE OF ITEMS WITH SIZES $S_1, S_2, \ldots\ldots, S_{50}$ IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1	1
0.0 – 0.2	1	1
0.0 – 0.5	1.198	1.2
0.0 – 1.0	1.33	1.42

TABLE 24. HARMONIC FIT - SEQUENCE OF ITEMS WITH SIZES $S_1, S_2, \ldots\ldots, S_{100}$ IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1	1
0.0 – 0.2	1.082	1.1
0.0 – 0.5	1.15	1.17
0.0 – 1.0	1.35	1.39

TABLE 25. HARMONIC FIT - SEQUENCE OF ITEMS WITH SIZES S_1, S_2 ,……, S_{1000} IN THE INTERVALS (0, 0.1], (0, 0.2], (0, 0.5], (0,1] WITH 100 RUNS

Items	Competitive Ratio	Worst Value
0.0 – 0.1	1.0006	1.02
0.0 – 0.2	1.0014	1.01
0.0 – 0.5	1.127	1.137
0.0 – 1.0	1.28	1.3

CHAPTER 6

IMPLEMENTATION OF TEST DATA

6.1 Optimal Solution

Start of main

initialize array size, lower and upper values

```
const granularity = 100

int size

int lower

int upper = granularity
```

6.1.1 Generate input random numbers

For generating random numbers, we declare an integer function called array[size]. A boolean function outputs the random array. We generate random numbers using srand and we use ‘for loop’ to execute a sequence of statements multiple times and abbreviate the code that manages the loop variable. Initialize and declare a variable ‘i’ in ‘for loop’ and iterate it with the array size. If the condition satisfies we use a formula to generate random numbers and store them in an array. Print the array using ‘cout’ function. The formula is:

```
array[i] = (lower + rand() % (upper – lower + 1))
```

6.1.2 Packing random numbers into bins

for each of items in array

if sum of current total and item value less than current bin capacity

add item to current total

else

assign the remaining capacity to current bin and continue to next bin

6.1.3 Rearranging the packed items

for each elements in array

generate a random index and assign element at index to new rearranged array

assign the last element in current array to random index

6.2 First Fit Bin Packing

initialize firstfit array to zero

for each of items in new rearranged array

if item in rearranged array can fit in firstfit bin

add to firstfit bin

else

continue to next firstfit bin

print results

6.3 Next Fit Bin Packing

initialize nextfit array to zero

for each of items in new rearranged array

if item in rearranged array can fit in the opened bin

add to the opened bin

else

continue to the next new bin and close the previous bin

print results

6.4 Best Fit Bin Packing

initialize bestfit array to zero

for each of items in new rearranged array

if item in rearranged array can fit in the opened bin whose empty space is minimum

add to the opened bin

else

continue to the next new bin

print results

6.5 Worst Fit Bin Packing

initialize bestfit array to zero

for each of items in new rearranged array

if item in rearranged array can fit in the opened bin whose empty space is maximum

add to the opened bin

else

continue to the next new bin

print results

6.6 Harmonic Fit Bin Packing

Declare and initialize all bins to zero.

- This is huge bin for item sizes (1/2, 1] (maxHbin = 0)
- This is large bin for item sizes (1/3, 1/2] (maxLbin = 0)
- This is medium bin for item sizes (1/4, 1/3] (maxMbin = 0)
- This is small bin for item sizes (0, 1/4] (maxSbin = 0)

for each item sizes there exists an array, initialize all the arrays to zero

for each of items in new rearranged array

if item in rearranged array lies between (1/2, 1] then

add to the huge bin

```
    else if item in rearranged array lies between (1/3, 1/2] then
                add to the large bin
    else if item in rearranged array lies between (1/4, 1/3] then
                add to the medium bin
    else if item in rearranged array lies between (0, 1/4] then
                add to the small bin
print total number of bins
```

CHAPTER 7

SUMMARY AND RESULTS OF ONLINE BIN PACKING

We know that bin packing is one of the classic and well-studied problems in the field of computer science. Since bin packing belongs to the class of NP-hard problems, it is difficult to come up with a polynomial time algorithm which solves the problem to give an optimal solution. So as a result, approximation algorithms are presented to find the closest possible solution to the optimal. Johnson has studied the bin packing problem and showed that next fit has a competitive ratio of 2 [9]. The proof for this ratio is simple and is proved in the above chapters. He also showed that the competitive ratio of first fit is 17/10 = 1.7. Yao [13] has redefined first fit and showed that the competitive ratio of first fit is 5/3 = 1.66. Lee and Lee [6] presented a harmonic algorithm which had a better ratio of 1.635.

This thesis show better competitive ratio than the above ratios. Here we find competitive ratio and worst values using different input sizes in different intervals. For each input size and each interval there will be different competitive ratios and worst values. To get better competitive ratio we need to find ratio in all possibilities. In this thesis, we find competitive ratio and worst value for 100 runs. The ratio between online algorithm and optimal solution is Competitive Ratio. The average of all the competitive ratios gives the actual competitive ratio and maximum of all the competitive ratios gives the worst value.

We know that Next Fit is more restrictive than any other fit algorithms, since it keeps only a single bin open and puts an incoming item into it whenever the item fits, otherwise the bin is closed and a new bin is opened. We know that next fit has a competitive ratio of 2. In this thesis, we show a better competitive ratio for next fit. The competitive ratio for small items is ≈ 1 and for large items (number of items = 1000 and the size lies in between (0, 1]) the competitive ratio is 1.3.

Similarly, we know that first fit and best fit has better ratio than next fit (ratio is 1.7). In the case of FF and BF, the competitive ratio for large items is ≈ 1. In case of First Fit, for small items (number of items = 10

and the size lies in between (0, 0.5]) the competitive ratio is 1.365. In case of Best Fit, for small items (number of items = 10 and the size lies in between (0, 1]) the competitive ratio is 1.29. The proof for this is very simple. The input sequence for this competitive ratio is 6n items of size 0.15, 0.34, 0.51.

We know that worst fit has a competitive ratio of 2. Here we show that worst fit still has worst competitive ratio for any number of items and the WF has a ratio of 1.795 still not better than 2. So practically worst fit will not work for bin packing.

Lee and Lee has presented a harmonic algorithm with a competitive ratio of 1.635. We show a better ratio of 1.46. Practically harmonic fit works better for smaller items and for large items the ratio is ≈ 1. Harmonic fit has better ratio if the number of items are 20 and the size lies in between (0, 0.2].

TABLE 26. LIST OF COMPETITIVE RATIOS AND WORST VALUES FOR ONLINE ALGORITHMS.

Online Algorithms	Input Sizes	Intervals	Competitive Ratio	Worst Value
Next Fit Algorithm	$s_1, s_2, \ldots, s_{1000}$	0.0 – 1.0	1.3	1.31
First Fit Algorithm	$s_1, s_2, \ldots, s_{10}$	0.0 – 0.5	1.365	1.5
Best Fit Algorithm	$s_1, s_2, \ldots, s_{10}$	0.0 – 1.0	1.29	1.33
Worst Fit Algorithm	$s_1, s_2, \ldots, s_{20}$	0.0 – 0.2	1.795	2
Harmonic Algorithm	$s_1, s_2, \ldots, s_{20}$	0.0 – 0.2	1.46	1.5

The above table shows that Best Fit Online Algorithm has better competitive ratio than any other online algorithms. The competitive ratio of Best Fit is 1.29 for input items in the interval (0, 1] and number of input items is 10.

REFERENCES

[1] Johnson, D.S.: Near-optimal bin-packing algorithms. Doctoral Thesis. MIT Press, Cambridge (1973)

[2] Baker, B.S.: A new proof for the first-fit decreasing bin-packing algorithm. J. Algorithms, 49-70(1985)

[3] A. Borodin and R. El-Yaniv. Online Computation and Competitive Analysis. Cambridge University Press, 1998.

[4] Richard M. Karp. Online Algorithms Versus Offline Algorithms: How much is it worth to know the Future? International Computer Science Institute

[5] Joseph malkevitch. Bin Packing, American Mathematical Society

[6] L.Becchetti. Sequential algorithms for partitioning problems: Minimum Bin Packing

[7] https://www2.informatik.hu-berlin.de/alcox/lehre/lvws1011/coalg/bin_packing.pdf

[8] Doina Bein, Wolfgang Bein, and Swathi Venigella. Cloud Storage and Online Bin Packing. http://www.egr.unlv.edu/~bein/pubs/bein_cloud_19.pdf

[9] D. S. Johnson, A.Demers, J. D. Ullman, M. R. Garey, and R. L. Graham(1974). Worst-case performance bounds for simple one-dimensional packing algorithms.SIAM J. Computing 3, 299-325.

[10] Prof. RNDr. Ji˘r´ı Sgall, DrSc. Online algorithms for variants of bin packing, Charles University in Prague, 2014

[11] Darapuneni, Yoga Jaideep, "A Survey of Classical and Recent Results in Bin Packing Problem" (2012). UNLV Theses/Dissertations/Professional Papers/Capstones. Paper 1663.

[12] C. Lee and D. Lee. A simple on-line bin-packing algorithm. *Journal of ACM*, 32:562572,1985

[13] RAMANAN, P., BROWN, D., LEE, C., AND LEE, D. 1989. On-line bin packing in linear time. J. Algor. 10, 305–326.

APPENDIX 1

```
#include<iostream>
#include<ctime>
#include<math.h>
#include<cstdlib>
using namespace std;
int main()
{
 const int granularity = 100;
 int size;
 int lower = 1;
 int upper = granularity;
 cout << "Enter the size of the array: " << endl;
 cin >> size;
 cout << "-------------------------------------" << endl;
 // Generating random numbers
 int array[size];
 srand(time(0));
 cout << "random array is:" << endl;
 for (int i=0; i<size; i++)
  {
   array[i] = (lower + rand() % (upper - lower + 1)) ;
   cout << array[i] << endl;
  }
 cout << "-------------------------------------" << endl;
 // Packing random numbers into bins
 int tally=0;
 int numberofbins = 1;
 for (int j=0; j<size;j++)
  {
   if (tally + array[j] < granularity)
            {
              tally = tally + array[j];
              cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins <<  endl;
            }
```

```
   else
            {
              array[j] = granularity - tally;
              tally = 0;
              cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins <<  endl;
              if (j<(size-1))
                {
                  numberofbins++;
                }
            }
  }
// Rearranging the packed items
int cs = size;
int b[size];
int k = 0;
cout << "----------------------------------------" << endl;
while (cs > 0)
  {
    int  index = rand() % cs;
    b[k] = array[index];
    k++;
    array[index] = array[cs - 1];
    cs--;
  }
int bin = 1;
for ( int l=0; l<size; l++)
  {
    cout <<  b[l]/float(granularity) << endl;
  }
cout << "----------------------------------------" << endl;
```

APPENDIX 2

```
#include<iostream>
#include<ctime>
#include<math.h>
#include<cstdlib>
using namespace std;
int main()
{
 const int granularity = 100;
 int size;
 int lower = 1;
 int upper = granularity;
 cout << "Enter the size of the array: " << endl;
 cin >> size;
 cout << "---------------------------------------" << endl;
 // Generating random numbers
 int array[size];
 srand(time(0));
 cout << "random array is:" << endl;
 for (int i=0; i<size; i++)
  {
   array[i] = (lower + rand() % (upper - lower + 1)) ;
   cout << array[i] << endl;
  }
 cout << "---------------------------------------" << endl;
 // Packing random numbers into bins
 int tally=0;
 int numberofbins = 1;
 for (int j=0; j<size;j++)
  {
   if (tally + array[j] < granularity)
            {
              tally = tally + array[j];
              cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins <<  endl;
            }
```

```
   else
            {
              array[j] = granularity - tally;
              tally = 0;
              cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins <<  endl;
              if (j<(size-1))
                {
                  numberofbins++;
                }
            }
  }
// Rearranging the packed items
int cs = size;
int b[size];
int k = 0;
cout << "----------------------------------------" << endl;
while (cs > 0)
  {
    int  index = rand() % cs;
    b[k] = array[index];
    k++;
    array[index] = array[cs - 1];
    cs--;
  }
int bin = 1;
for ( int l=0; l<size; l++)
  {
    cout <<  b[l]/float(granularity) << endl;
  }
cout << "----------------------------------------" << endl;
// First Fit Bin Packing
int maxbin=0;
int fbin[size];
for(int i=0; i<size; i++)
  {
```

```
    fbin[i] = 0;
   }
  for(int g=0; g<size; g++)
   {
   for(int i=0;i<size;i++)
    {
            if(fbin[i] + b[g] <= granularity)
             {
               fbin[i] = fbin[i] + b[g];
               cout << " item:" << b[g]/float(granularity) << "    " <<  "put item in bin:" << i+1 <<  "    "  << "size of bin:" <<
fbin[i]/float(granularity) <<  endl;
               if (i+1 > maxbin)
                 {
                        maxbin = i+1;
                 }
               break;
             }
    }
   }
  cout << "----------------------------------------" << endl;
  cout << "number of bins for first fit:" << maxbin << endl;
  cout << "optimal number of bins:" << numberofbins << endl;
  cout << "ratio between first fit and optimal:" << maxbin/(float)numberofbins << endl;
}
```

APPENDIX 3

```
#include<iostream>
#include<ctime>
#include<math.h>
#include<cstdlib>
using namespace std;
int main()
{
 const int granularity = 100;
 int size;
 int lower = 1;
 int upper = granularity;
 cout << "Enter the size of the array: " << endl;
 cin >> size;
 cout << "---------------------------------------" << endl;
 // Generating random numbers
 int array[size];
 srand(time(0));
 cout << "random array is:" << endl;
 for (int i=0; i<size; i++)
  {
   array[i] = (lower + rand() % (upper - lower + 1)) ;
   cout << array[i] << endl;
  }
 cout << "---------------------------------------" << endl;
 // Packing random numbers into bins
 int tally=0;
 int numberofbins = 0;
 for (int j=0; j<size;j++)
  {
   if (tally + array[j] < granularity)
            {
              tally = tally + array[j];
              cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins+1 <<  endl;
            }
```

```
    else
                {
                  array[j] = granularity - tally;
                  tally = 0;
                  cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins+1 <<  endl;
                  if(j<(size-1))
                    {
                      numberofbins++;
                    }
                }
   }
  // Rearranging the packed items
  int cs = size;
  int b[size];
  int k = 0;
  cout << "----------------------------------------" << endl;
  while (cs > 0)
    {
      int  index = rand() % cs;
      b[k] = array[index];
      k++;
      array[index] = array[cs - 1];
      cs--;
    }
  int bin = 1;
  for ( int l=0; l<size; l++)
    {
      cout <<  b[l]/float(granularity) << endl;
    }
 cout << "----------------------------------------" << endl;
  // Next Fit Bin Packing
  int maxbin = 1;
  int nbin[size];
  for(int i=0;i<size;i++)
    {
```

```
    nbin[i] = 0;
   }
  for(int g=0;g<size;g++)
   {
    for(int i=(maxbin-1);i<size;i++)
            {
             if(nbin[i] + b[g] <= granularity)
               {
                nbin[i] = nbin[i] + b[g];
                cout << "item:" << b[g]/float(granularity) << "    " <<  "put item in bin:" << i+1 <<  "    "  << "size of bin:" <<
nbin[i]/float(granularity) <<  endl;
                if (i+1 > maxbin)
                        {
                         maxbin = i+1;
                        }
                break;
               }
            }
   }
  cout << "----------------------------------------" << endl;
  cout << "number of bins for next fit:" << maxbin << endl;
  cout << "optimal number of bins:" << numberofbins+1 << endl;
  cout << "ratio between next fit and optimal:" << maxbin/(float)(numberofbins+1) << endl;
}
```

APPENDIX 4

```
#include<iostream>
#include<ctime>
#include<math.h>
#include<cstdlib>
using namespace std;
int main()
{
 const int granularity = 100;
 int size;
 int lower = 1;
 int upper = granularity;
 cout << "Enter the size of the array: " << endl;
 cin >> size;
 cout << "-------------------------------------" << endl;
 // Generating random numbers
 int array[size];
 srand(time(0));
 cout << "random array is:" << endl;
 for (int i=0; i<size; i++)
  {
   array[i] = (lower + rand() % (upper - lower + 1)) ;
   cout << array[i] << endl;
  }
 cout << "-------------------------------------" << endl;
 // Packing random numbers into bins
 int tally=0;
 int numberofbins = 1;
 for (int j=0; j<size;j++)
  {
   if (tally + array[j] < granularity)
            {
              tally = tally + array[j];
              cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins <<  endl;
            }
```

```
   else
            {
              array[j] = granularity - tally;
              tally = 0;
              cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins <<  endl;
              if (j<(size-1))
                {
                  numberofbins++;
                }
            }
  }
// Rearranging the packed items
int cs = size;
int b[size];
int k = 0;
cout << "---------------------------------------" << endl;
while (cs > 0)
  {
    int  index = rand() % cs;
    b[k] = array[index];
    k++;
    array[index] = array[cs - 1];
    cs--;
  }
int bin = 1;
for ( int l=0; l<size; l++)
  {
    cout <<  b[l]/float(granularity) << endl;
  }


cout << "---------------------------------------" << endl;
//Best Fit Bin Packing
int maxbin = 0;
int bbin[size];
int bfindex = 0;
```

```
 for(int i=0; i<size; i++)
  {
   bbin[i] = 0;
  }
 for(int g=0; g<size; g++)
  {
   int diff = granularity;
   for(int i=0;i<size;i++)
            {
             if(bbin[i] + b[g] <= granularity)
               {
                 if (diff > (granularity - (bbin[i] + b[g])))
                        {
                         diff = granularity - (bbin[i] + b[g]);
                         bfindex = i;
                        }
               }
             if (i >= maxbin)
               break;
            }
   bbin[bfindex] = bbin[bfindex] + b[g];
   if (bfindex+1 > maxbin)
            {
             maxbin = bfindex+1;
            }
   cout << " item:" << b[g]/float(granularity) << "     " << "difference:" << diff/float(granularity) <<  "     " << "put item in bin:" << bfindex+1
<<  "     "  << "size of bin:" << bbin[bfindex]/float(granularity) <<  endl;
  }
 cout << "----------------------------------------" << endl;
 cout << "number of bins for best fit:" << maxbin << endl;
 cout << "optimal number of bins:" << numberofbins << endl;
 cout << "ratio between best fit and optimal:" << maxbin/float(numberofbins) << endl;
}
```

APPENDIX 5

```
#include<iostream>
#include<ctime>
#include<math.h>
#include<cstdlib>
using namespace std;
int main()
{
 const int granularity = 100;
 int size;
 int lower = 1;
 int upper = granularity;
 cout << "Enter the size of the array: " << endl;
 cin >> size;
 cout << "---------------------------------------" << endl;
 // Generating random numbers
 int array[size];
 srand(time(0));
 cout << "random array is:" << endl;
 for (int i=0; i<size; i++)
  {
   array[i] = (lower + rand() % (upper - lower + 1)) ;
   cout << array[i] << endl;
  }
 cout << "---------------------------------------" << endl;
// Packing random numbers into bins
 int tally=0;
 int numberofbins = 1;
 for (int j=0; j<size;j++)
  {
   if (tally + array[j] < granularity)
              {
               tally = tally + array[j];
               cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins <<  endl;
              }
```

```
   else
            {
              array[j] = granularity - tally;
              tally = 0;
              cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins <<  endl;
              if (j<(size-1))
                {
                  numberofbins++;
                }
            }
  }
// Rearranging the packed items
int cs = size;
int b[size];
int k = 0;
cout << "---------------------------------------" << endl;
while (cs > 0)
  {
    int  index = rand() % cs;
    b[k] = array[index];
    k++;
    array[index] = array[cs - 1];
    cs--;
  }
int bin = 1;
for ( int l=0; l<size; l++)
  {
    cout <<  b[l]/float(granularity) << endl;
  }
cout << "---------------------------------------" << endl;
//Worst Fit Bin Packing
int maxbin = 0;
int bbin[size];
int bfindex = 0;
int diff = 0;
```

```
  for(int i=0; i<size; i++)
    {
     bbin[i] = 0;
    }
  for(int g=0; g<size; g++)
    {
     diff = 0;
     for(int i=0;i<size;i++)
                {
                 if(bbin[i] + b[g] <= granularity)
                   {
                    if (diff <= (granularity - (bbin[i] + b[g])))
                              {
                               if (diff != 0 && bbin[i]==0)
                                 break;
                               diff = granularity - (bbin[i] + b[g]);
                               bfindex = i;
                              }
                   }
                 if (i >= maxbin)
                   break;
                }
     bbin[bfindex] = bbin[bfindex] + b[g];
     if (bfindex+1 > maxbin)
                {
                 maxbin = bfindex+1;
                }
     cout << "item:" << b[g]/float(granularity) << "     " << "difference:" << diff/float(granularity) <<  "     " << "put item in bin:" << bfindex+1
<<  "     "  << "size of bin" << bfindex+1 << ": " << bbin[bfindex]/float(granularity) <<  endl;
    }
  cout << "----------------------------------------" << endl;
  cout << "number of bins for worst fit:" << maxbin << endl;
  cout << "optimal number of bins:" << numberofbins << endl;
  cout << "ratio between worst fit and optimal:" << maxbin/float(numberofbins) << endl;
}
```

APPENDIX 6

```
#include<iostream>
#include<ctime>
#include<math.h>
#include<cstdlib>
using namespace std;
int main()
{
 const int granularity = 100;
 int size;
 int lower = 1;
 int upper = granularity;
 cout << "Enter the size of the array: " << endl;
 cin >> size;
 cout << "---------------------------------------" << endl;
 // Generating random numbers
 int array[size];
 srand(time(0));
 cout << "random array is:" << endl;
 for (int i=0; i<size; i++)
  {
   if (i%2 == 0)
            {
             lower = 0, upper = 100;
             array[i] = (lower + rand() % (upper - lower +1));
             cout << array[i] << endl;
            }
   else
            {
             lower = 0, upper = 100;
             array[i] = (lower + rand() % (upper - lower +1));
             cout << array[i] << endl;
            }
  }
 /*  for (int i=0; i<size; i++)
```

```
  {
    array[i] = (lower + rand() % (upper - lower + 1)) ;
    cout << array[i] << endl;
    } */
cout << "-------------------------------------" << endl;
// Packing random numbers into bins
int tally=0;
int numberofbins = 0;
for (int j=0; j<size;j++)
  {
    if (tally + array[j] < granularity)
                {
                  tally = tally + array[j];
                  cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins+1 <<  endl;
                }
    else
                {
                  array[j] = granularity - tally;
                  tally = 0;
                  cout << array[j]/float(granularity) << " " << "bin:" <<  numberofbins+1 <<  endl;
                  if(j<(size-1))
                    {
                      numberofbins++;
                    }
                }
  }
// Rearranging the packed items
int cs = size;
int b[size];
int k = 0;
cout << "---------------------------------------" << endl;
while (cs > 0)
  {
    int  index = rand() % cs;
    b[k] = array[index];
```

```
    k++;
    array[index] = array[cs - 1];
    cs--;
  }
int bin = 1;
for ( int l=0; l<size; l++)
  {
    cout <<  b[l]/float(granularity) << endl;
  }
cout << "-------------------------------------" << endl;
// Harmonic Algorithm
int maxHbin=0;
int maxLbin=0;
int maxMbin=0;
int maxSbin=0;
//initialize arrays
int hbin[size];
for(int i=0;i<size;i++)
  {
    hbin[i] = 0;
  }
int lbin[size];
for(int i=0;i<size;i++)
  {
    lbin[i] = 0;
  }
int mbin[size];
for(int i=0;i<size;i++)
  {
    mbin[i] = 0;
  }
int sbin[size];
for(int i=0;i<size;i++)
  {
    sbin[i] = 0;
```

```
   }
  for(int g=0;g<size;g++)
   {
     if(b[g]>(granularity/2))
       {
         if(hbin[maxHbin] + b[g] > granularity)
                  {
            maxHbin=maxHbin+1;
                  }
         hbin[maxHbin] = hbin[maxHbin] + b[g];
         cout << "item:" << b[g]/float(granularity) << "    " <<  "put item in HUGE bin:" << maxHbin+1 <<  "    "  << "size of bin:" <<
hbin[maxHbin]/float(granularity) <<  endl;
       }
     else if(b[g]>(granularity/3))
       {
         if(lbin[maxLbin] + b[g] > granularity)
                  {
            maxLbin=maxLbin+1;
                  }
         lbin[maxLbin] = lbin[maxLbin] + b[g];
         cout << "item:" << b[g]/float(granularity) << "    " <<  "put item in LARGE bin:" << maxLbin+1 <<  "    "  << "size of bin:" <<
lbin[maxLbin]/float(granularity) <<  endl;
               }
     else if (b[g]>(granularity/4))
               {
         if(mbin[maxMbin] + b[g] > granularity)
                  {
            maxMbin=maxMbin+1;
                  }
         mbin[maxMbin] = mbin[maxMbin] + b[g];
         cout << "item:" << b[g]/float(granularity) << "    " <<  "put item in MEDIUM bin:" << maxMbin+1 <<  "    "  << "size of bin:" <<
mbin[maxMbin]/float(granularity) <<  endl;
               }
     else
               {
```

```
                int binindex = 0;
                for(int i=0; i<=maxSbin; i++)
                  {
                    if(sbin[i] + b[g] <=  granularity)
                            {
                             binindex = i;
                             break;
                            }
                    if(i==maxSbin)
                            {
                             binindex=i+1;
                             maxSbin=maxSbin+1;
                            }
                  }
                sbin[binindex] = sbin[binindex] + b[g];
                cout << "item:" << b[g]/float(granularity) << "    " <<  "put item in SMALL bin:" << binindex+1 <<  "    "  << "size of bin:" <<
sbin[binindex]/float(granularity) <<  endl;
               }
     if(g+1==size)
               {
                if(hbin[0]>0)
                  maxHbin+=1;
                if(lbin[0]>0)
                  maxLbin+=1;
                if(mbin[0]>0)
                  maxMbin+=1;
                if(sbin[0]>0)
                  maxSbin+=1;
               }
   }
  int totalHAbins=maxHbin+maxLbin+maxMbin+maxSbin;
  cout << "----------------------------------------" << endl;
  cout << "number of bins for ha fit:" << totalHAbins << endl;
  cout << "optimal number of bins:" << numberofbins+1 << endl;
  cout << "ratio between ha fit and optimal:" <<  totalHAbins/(float)(numberofbins+1) << endl;
 }
```

CURRICULUM VITAE

Graduate College
University of Nevada, Las Vegas

Kalpana Rajagopal

Home Address:
71 Sevilla Heights Dr
Henderson, NV 89074.

Degrees:
Bachelor of Technology, Computer Science,
Jawaharlal Nehru Technological University, India

Thesis Title: Simulation of Online Bin Packing in Practice.

Thesis Examination Committee:
Chairperson, Dr. Wolfgang Bein, Ph. D.
Committee Member, Dr. Ajoy K Datta, Ph. D.
Committee Member, Dr. Laxmi Gewali, Ph. D.
Graduate faculty Representative, Dr. Muthukumar Venkatesan, Ph. D.