First-fit and various Heuristics of Best Fit Algorithm for Online 2-Dimensional Vector Packing with Bounded and Unbounded Space

Ullash Saha [*], Sourav [*]

[1]Department of Computer Science, University of Manitoba,
Winnipeg, Manitoba

[*]*sourav*, [*]*sahau@myumanitoba.ca*

ABSTRACT

In classical bin packing problem, a list of elements [0, 1] are given in some order and the motivation is to place those elements into a minimum number of bins with a constraint that the total size of elements in a bin cannot be more than 1. However, vector packing is another variant of the bin packing problem where items are d-dimensional vectors. The goal is to put these items into a minimum number of d-dimensional bins with constraint of the size of each dimension. Let us consider a sequence of n items $\sigma = < (a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4), \ldots\ldots, (a_n, b_n) >$ which is revealed in online manner where each item is a 2-dimensional vector $[0,100]^2$. The goal is to pack these items into 2-dimensional bins with minimizing the total number of opened bins. Each bin has a capacity of 100 in each of its d dimensions. In this project, we have done an empirical evaluation of the first-fit algorithm and various heuristics of the best-fit algorithm for online 2-dimensional vector packing with bounded and unbounded space. We compared different variants of the best-fit algorithm and found out the heuristic that works best on average. In the end, we observed the anomalous behaviour of the best-fit algorithm for online vector packing with bounded space.

1. INTRODUCTION

Let us consider a sequence of n items $\sigma = < (a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4), \ldots\ldots (a_n, b_n) >$ which is disclosed in online manner where each item has 2-dimensions [4] . The size of each dimension lies between [0,100]. The objective is to pack these items into a minimum number of 2-dimensional bins of capacity $\overrightarrow{100^2}$. An item will be fitted in a bin if the bin has enough space for both dimensions of the item. An item (a_i, b_i) can be put into a 2-dimensional bin B_{xy} if empty space of

$B_x \geq$ size of a_i and empty space of $B_y \geq$ size of b_i. Different algorithms have different strategies for placing the vectors into the bins as well as different strategies for closing a bin. However, in unbounded-space setting, there is no restriction in the number of opened bins at a time. On the other hand, in the bounded-space setting [5], there are at most k number of bins remains active or opened at a time. If an additional bin needs to be opened, one of the existing k bins should be closed (i.e., it is put aside and not referred to). Opening a new bin increases the cost of an algorithm by 1. The goal of this project is to find out the algorithm which requires minimum cost for online vector packing in average case.

Moreover, in the packing problem, for easier sequences in which some items are removed or shrinked, sometimes require a larger number of bins in certain bin packing algorithms. These algorithms are called "anomalous algorithms" and are prone to unpredictable behavior (which is undesired). Note that an algorithm is anomalous if the cost is increased by removing/shrinking items [1]. Best fit algorithm is such an “anomalous algorithm”. In bounded-space setting, it is expected that the cost of the algorithm will be decreased if we increase the bounded value k. Best fit algorithm sometimes does the opposite. Increasing the value of k causes more cost in best fit algorithm. In this project we have observed such anomalous behaviour of best fit algorithm.

Combinatorial optimization problems like multiprocessor scheduling, bin packing and knapsack problem are widely studied in computer science. The motivation of all these problems is to pack items of different sizes into the bins having some capacity constraints. However, Vector bin packing models are inspired from virtual machine placements problems as well as online resource allocation in the cloud where users can request a set of servers on demand. Vector packing problem can span across various dimensions based on the demand of servers and other resources like bandwidth, network resources, memory and disk etc. The rise of virtualization technology increases the trend of using services of the data center because it helps to reduce IT budget and other costs related to maintenance. However, the goal is not only to reduce IT budget, but also minimizing energy cost of the data center. The problem of resource allocation becomes more challenging when we have multidimensional nature of load for data centers. In this case, each host has the capability to offer its own CPU resources, memory resources and network resources. The objective is to use the minimum number of hosts without violating any request.

However, we can place various online virtual machine requests on the same host with the corresponding load across each dimension in an additive manner. Therefore, the problem of serving online virtual machines requests by the different hosts is based on an online vector bin packing problem. The vector bin packing problem was introduced in 1977 [7]. This paper has suggested two different type of algorithm for vector bin packing namely First Fit Decreasing algorithm and Best Fit Decreasing Algorithm. In these algorithms, the items are sorted in decreasing order. These algorithms work effectively when dimension d=1. Moreover, these algorithms have an offline sequence of inputs available beforehand, so it is quite easy to run the sorting algorithm. On the other hand, if the sequence of input is generated in an online manner for 2-dimensional items, it will be quite challenging to fit those items in online 2-D Vector packing problem. In our problem, we are also considering k bounded space. A bounded value of k means that throughout the operation there cannot be more than k active bins for a given time.

In this project, we have investigated the computational behaviour of the first-fit algorithm and three different variants i.e. max dimension, sum of the dimensions and min dimension of the best-fit algorithm for online 2-dimensional vector packing with bounded and unbounded space. We empirically evaluated these heuristics with online 2-dimensional input sequences and investigated which variant of best-fit algorithm works best on average. We also implemented the first-fit algorithm for vector packing and made a comparison with the best-fit algorithms in terms of their performances. In the end, we have also found the anomalous behaviour of the best-fit algorithm in the bounded-space setting.

2. RELATED WORK`

Approximation Algorithm [6]: To measure the performance of a given algorithm A for a task sequence σ, we compare it with the performance of an optimal algorithm for the same sequence. The ratio of A(σ) and OPT(σ) provides us with the quality measure of a given algorithm for a sequence σ. An approximation algorithm A has absolute performance guarantee 1 + Є, if

$$A(\sigma) \leq (1 + Є)\ OPT(\sigma)$$

The vector bin packing problem is NP-hard [8,9]. No polynomial time approximation algorithm scheme (PTAS) for bin packing problem can have absolute performance better than 3/2 based on NP hardness of partition problem. In practice, 1-dimension bin packing problem has a large running time for PTAS. However, when $d \geq 2$, the vector bin packing become APX-hard and there is no asymptotic PTAS for bin packing problem unless P=NP [8].

Kou et. al [7] suggested two different types of algorithm for vector bin packing namely First Fit Decreasing algorithm and Best Fit Decreasing Algorithm. However, in these algorithms, the items are sorted in decreasing order. Maruyama et. al [10] had provided the generalization of 1-dimensional heuristics of bin packing to the multidimensional and compared performance of various heuristic packing algorithms and provide guidelines for bin packing. Spieksma[11] used heuristics from FFD for 2-dimensional vector packing and investigated lower bound for an optimal solution. Steven S. Seiden [12] provided Harmonic++ algorithm for online bin packing problem and corrected the performance ratio of Harmonic+1 which is at least 1.59217. Azar et al. [13] provided tight bound for online vector packing. They provided strong lower bound $\Omega(d^{\frac{1}{B}-\epsilon}1)$.

Best fit algorithm for bin packing has some anomalous behaviour. Graham [2] has shown such anomalous behaviour in multiprocessing scheduling algorithm. He found that a decrease of the tasks timing or increase of the requirement of processors in the favourable fashion does not improve the execution time. On the other hand, it will increase the required time to complete the list of tasks. Monotonic behaviour is also studied in dynamic memory allocation. In double paging

system [3], if we keep reference string and real memory with fixed size but increasing the size of virtual memory can cause the increment of the page fault because of anomalies.

Considering above work, we investigated the computational behaviour of the first-fit and proposed heuristics of the best-fit algorithm with bounded and unbounded space for online vector packing.

3. First fit and Various Heuristics of Best Fit Algorithm

In best-fit algorithm, an incoming input is placed in the bin with the largest level which has enough space for the item. Basically, in best-fit algorithm, we need to find the fullest bin (less empty space) which has enough space for the current item and place the item into that bin. If no bin has enough space for the current item, then we open a new bin. In unbounded-space setting, there is no restriction in the total number of the opened bins at a time. Nevertheless, in bounded-space setting, a bounded value k implies that there can not be more than k number of active bins at a time. If an additional bin needs to be opened, one of the existing k bins should be closed. The best-fit algorithm chooses the fullest bin whenever it requires to close a bin. It is very simple to define the fullest bin (highest level) in the classical bin packing problem (1-dimensional bin packing). The total size of the items in a bin is called the level of that bin in 1-dimensional bin packing.

However, in vector bin packing problem, it is critical to define the level of a bin. Therefore, it is difficult to find the fullest bin to place an item or to close a bin. In this project, we have considered three variants to define the level of a bin. In 2-dimensional vector packing, each item has two fields or dimensions. We can consider the first dimension as x-coordinate of the item and the second dimension as y-coordinate of the item. Moreover, the bins have also 2 dimensions. First dimension contains x-coordinates of the items and second dimension contains y-coordinates of the items. So, each bin has an x-coordinate and a y-coordinate as well. We can define the level of a bin in terms of maximum coordinate, sum of the coordinates and the minimum coordinate of the bin.

Each bin has 2 coordinates. Firstly, we have defined the level of a bin in terms of their maximum coordinate size. Let us consider we have three opened bins (B1, B2, B3) in a certain time having a size of (40,70), (80,5) and (55,42) respectively. If we consider the level of the bins in terms of their maximum coordinate, the levels of B1, B2 and B3 will be 70, 80 and 55 respectively. The best-fit algorithm will select B2 as the fullest bin because it has the highest maximum coordinate (highest level). If two bins have the same size of maximum coordinate, then the algorithm will define the level of the bins in the term of minimum coordinate and select the one which has the highest minimum coordinate.

Secondly, we have defined the level of a bin in terms of their minimum coordinate size. Again, we are considering three opened bins (B1, B2, B3) in a certain time having a size of (40,70), (80,5) and (55,42) respectively. If we consider the level of the bins in terms of their minimum coordinate, the levels of B1, B2 and B3 will be 40, 5 and 42 respectively. The best-fit algorithm will select B3 as the fullest bin because it has the highest size of minimum coordinate. If two bins have the

same size of minimum coordinate, then the algorithm will choose the one whose size of the maximum coordinate is higher.

Finally, we have designated the level of a bin in terms of the total size of their coordinates. If we consider the same example of three opened bins (B1, B2, B3) in a certain time having a size of (40,70), (80,5) and (55,42) respectively, the levels of B1, B2 and B3 will be 110, 85 and 107 respectively. The best-fit algorithm will select B1 as the fullest bin because it has the highest total. If two bins have the same sum of the coordinates, then the algorithm will choose the bin whose size of the maximum coordinate is higher.

A best-fit algorithm performs 2 functionalities in bounded-space setting (i.e. place an item into a bin and close a bin). The algorithm can choose the fullest bin in three ways (max of the sum, max of maximum Coordinate, max of minimum coordinate) to place an item as well as for closing a bin. So, we have implemented nine algorithms in our project for bounded-space setting as shown in figure 1. In unbounded-space setting, no bins are required to be closed. Three variants of algorithms are enough for placing the items in the bins for the bounded-space setting.

On the other hand, the approach of the first-fit algorithm for vector packing is simpler than the best-fit algorithm. We know that the sequence of input discloses in an online manner. When a new item is appeared, the first fit algorithm simply places the item into the first available active bin which has enough space for both dimensions of the vector item. In bounded-space setting, Whenever the algorithm requires to close a bin, it chooses the first active bin.

1. maxTotal_maxTotal:
   Place an item in terms of maximum sum of the co-ordinates
   Close a bin in terms of maximum sum of the co-ordinates
2. maxTotal_maxMax:
   Place an item in terms of maximum sum of the co-ordinate
   Close a bin in terms of max of maximum co-ordinate
3. maxTotal_maxMin:
   Place an item in terms of maximum sum of the co-ordinates
   Close a bin in terms of max of minimum co-ordinate
4. maxMax_maxTotal:
   Place an item in terms of max of maximum co-ordinate
   Close a bin in terms of maximum sum of the co-ordinate
5. maxMax_maxMax:
   Place an item in terms of max of maximum co-ordinate
   Close a bin in terms of max of maximum co-ordinate
6. maxMax_maxMin:
   place an item in terms of max of maximum co-ordinate
   Close a bin in in terms of max of minimum co-ordinate
7. maxMin_maxTotal:
   Place an item in terms of max of minimum co-ordinate
   Close a bin in terms of maximum sum of the co-ordinat
8. maxMIn_maxMax:
   Place an item in terms of max of minimum co-ordinate
   Close a bin in terms of max of maximum co-ordinate
9. maxMin_maxMin:
   place an item in terms of max of minimum co-ordinate
   Close a bin in terms of max of minimum co-ordinate

Fig1. Nine variants of the best-fit algorithm.

At the end of the paper, we have attached the pseudocode (Supplementary Tables) of the algorithms. The source code can be found in [14].

4. EXPERIMENTS

To evaluate the algorithms for online vector packing, the experiments were conducted with identical bins. Each bin had 2-dimensions with the capacity of 100 for each dimension. Our algorithms were run on a synthetic instance of online input sequence which was generated using a pseudorandom number generator. The generated values using a pseudorandom generator were uniformly distributed and it was not possible to predict future values from the previous inputs. The values were only integer numbers for both dimensions. Figure 2 shows different input fields used to run all the algorithms for vector bin packing and their corresponding output for each algorithm.

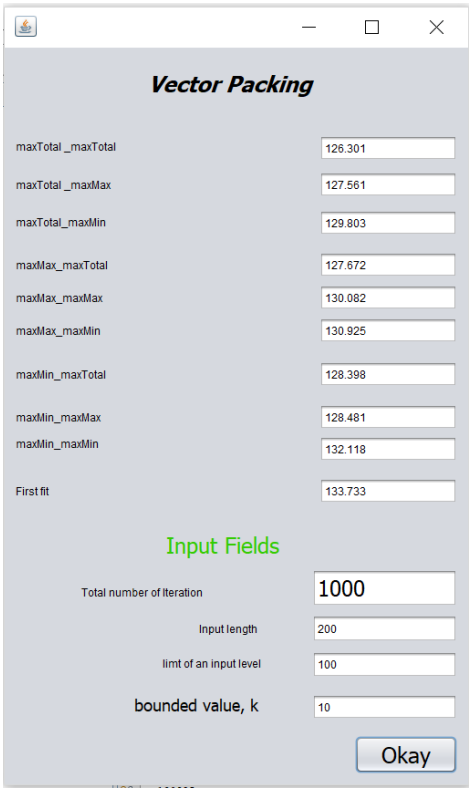


Fig2. An Interface of online vector bin packing with input field and output of each algorithm.

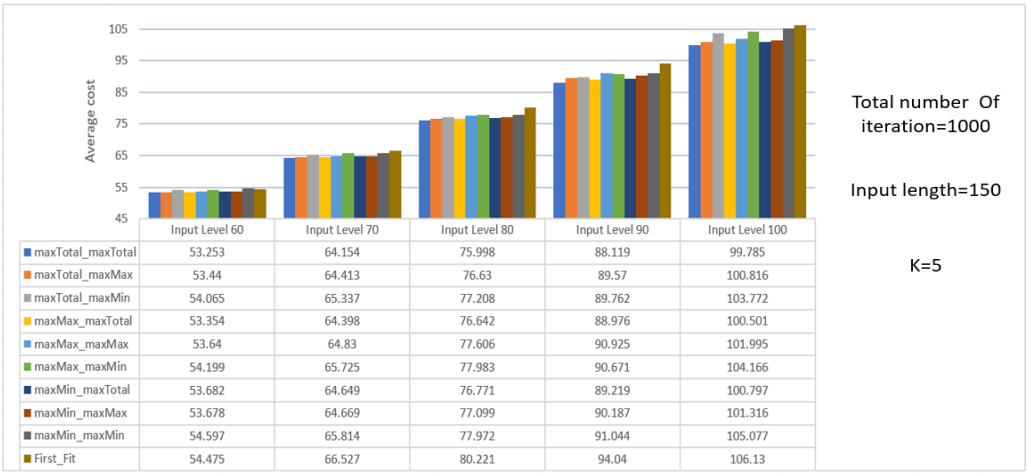
From Figure 2, we can see that there are 4 input fields which are used to test the performances of the algorithms. The length of the input sequence indicates the total number of items in the sequence. The size of each dimension of an input remains within a specific limit. Let us say, limit of input level is 60. Then the values of both dimensions will be integers and remain between [0,60]. A bounded value of k implies that, at any given time, at most k bins are active. If the total number of iteration = n, each algorithm will be tested by n different random input sequences for the fixed length, input limit and bounded value. Finally, the algorithms will calculate their average cost and display it based on the value of the input fields. For analyzing the algorithms in unbounded-space setting, we kept the value of k as same as the length of the input sequence. Any algorithm will open at most n number of bins where n is the input length. If k=length of the input sequence, all

the bins will be active at any time. In unbounded-space setting, no bins are required to close. That is why maxTotal_maxTotal, maxTotal_maxMax and maxTotal_maxMin gave the same output as all the algorithms used the same strategy (sum of the coordinates) to place the items. Similarly, maxMax_maxTotal, maxMax_maxMax, maxMax_maxMin gave the same output and maxMin_maxTotal, maxMin_maxMax, maxMin_maxMin gave the same result.

5. RESULTS

Average case performance result

For empirical evaluation of First fit and various heuristics of Best fit algorithms, all the algorithms were coded using Java language and ran them with different random sequences. There are four input parameters (i.e. input length, input level limit, k, number of iterations) for testing the outputs of the algorithms. Initially, the experiment was done with bounded value k=5, length of input sequence=150 and number of iteration =1000 as shown in figure 3. We changed the limit of the input level gradually (input level limit =60, 70, 80, 90, 100) under the same input setting. The cost of each algorithm was measured using the average number of bins opened by each of the algorithms.

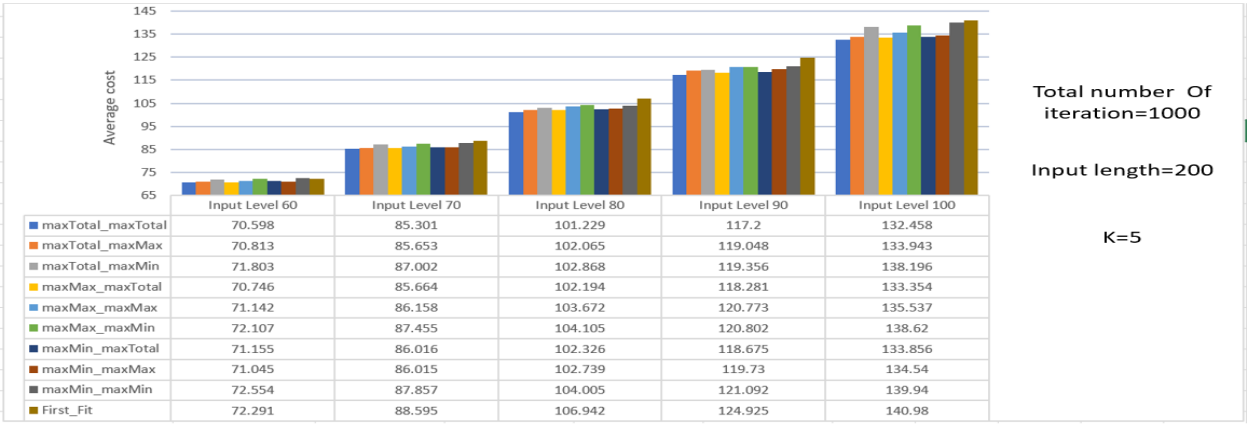


	Input Level 60	Input Level 70	Input Level 80	Input Level 90	Input Level 100
maxTotal_maxTotal	53.253	64.154	75.998	88.119	99.785
maxTotal_maxMax	53.44	64.413	76.63	89.57	100.816
maxTotal_maxMin	54.065	65.337	77.208	89.762	103.772
maxMax_maxTotal	53.354	64.398	76.642	88.976	100.501
maxMax_maxMax	53.64	64.83	77.606	90.925	101.995
maxMax_maxMin	54.199	65.725	77.983	90.671	104.166
maxMin_maxTotal	53.682	64.649	76.771	89.219	100.797
maxMin_maxMax	53.678	64.669	77.099	90.187	101.316
maxMin_maxMin	54.597	65.814	77.972	91.044	105.077
First_Fit	54.475	66.527	80.221	94.04	106.13

Fig3. Average cost of the different heuristics when k=5, the sequence having length=150 and run for 1000 times.

Figure 3 shows that maxTotal_maxTotal performed the best in average case. maxMax_maxTotal and maxTotal_maxMax also gave a promising result close to maxTotal_maxTotal. However First fit algorithm is worst as compared to any heuristic of the best-fit algorithm.

In the next experiment, the length of the input sequence was set to 200 items to observe the behaviour of each algorithm. Figure 4 shows the performance of the different heuristics when the length of the input sequence was only changed to 200 while other parameters remained the same as the first experiment.



	Input Level 60	Input Level 70	Input Level 80	Input Level 90	Input Level 100
maxTotal_maxTotal	70.598	85.301	101.229	117.2	132.458
maxTotal_maxMax	70.813	85.653	102.065	119.048	133.943
maxTotal_maxMin	71.803	87.002	102.868	119.356	138.196
maxMax_maxTotal	70.746	85.664	102.194	118.281	133.354
maxMax_maxMax	71.142	86.158	103.672	120.773	135.537
maxMax_maxMin	72.107	87.455	104.105	120.802	138.62
maxMin_maxTotal	71.155	86.016	102.326	118.675	133.856
maxMin_maxMax	71.045	86.015	102.739	119.73	134.54
maxMin_maxMin	72.554	87.857	104.005	121.092	139.94
First_Fit	72.291	88.595	106.942	124.925	140.98

Fig4. Average cost of the different heuristics when k=5, the sequence having length=200 and run for 1000 times.

However, change in length of the input sequence did not affect the comparative performances of the algorithms. maxTotal_maxTotal was the best one (opened lesser bin than any other algorithms). The first-fit gave better performance than maxMin_maxMin only for once when the input level was 60.

We checked the costs of the algorithms under different parameters. Figure 5, 6, 7, 8, and 9 show some experiments we did for bounded-space setting (k=5, 10, 20).

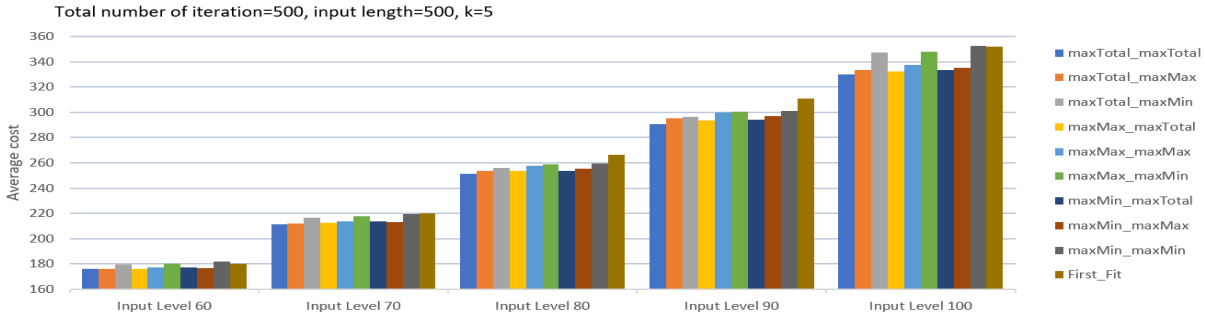


Fig5. Average cost of the different heuristics when k=5, the sequence having length=500 and run for 500 times.

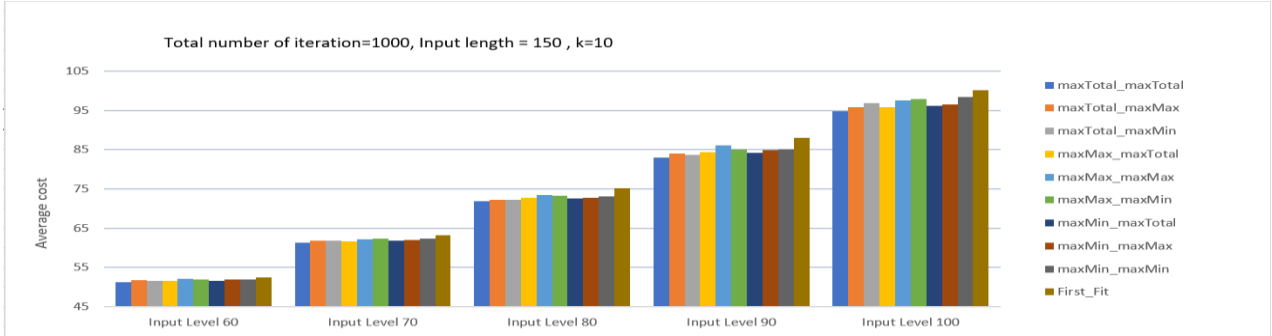


Fig6. Average cost of the different heuristics when k=10, the sequence having length=150 and run for 1000 times.

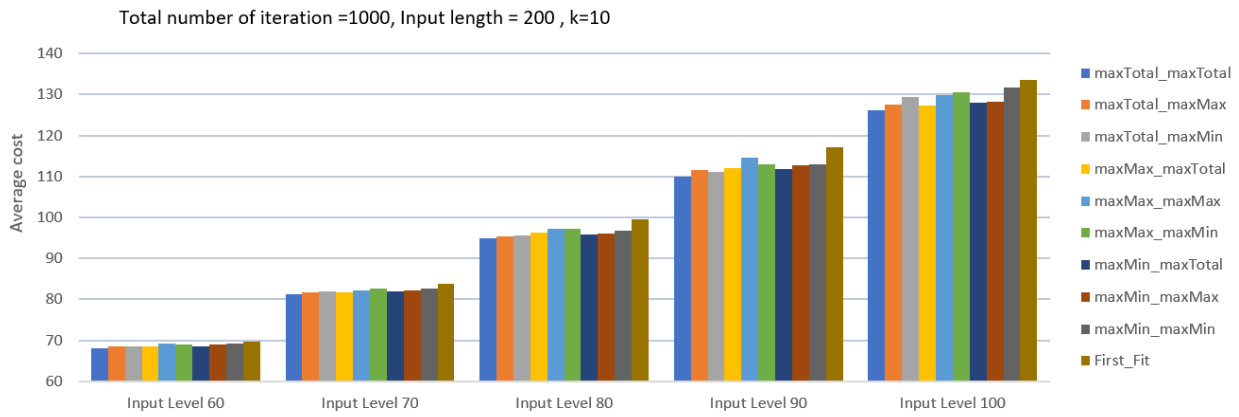


Fig7. Average cost of the different heuristics when k=10, the sequence having length=200 and run for 1000 times.

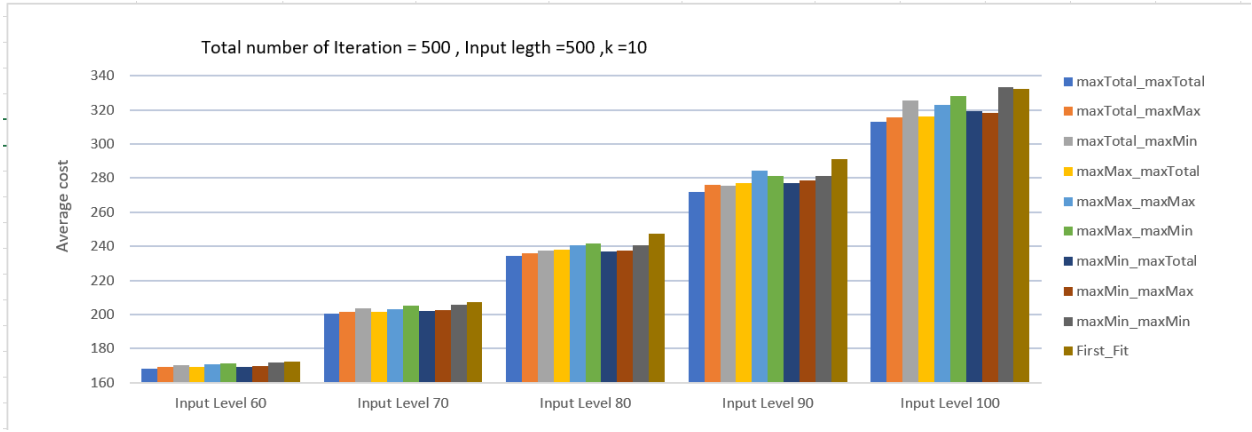


Fig8. Average cost of the different heuristics when k=10, the sequence having length=500 and run for 500 times.

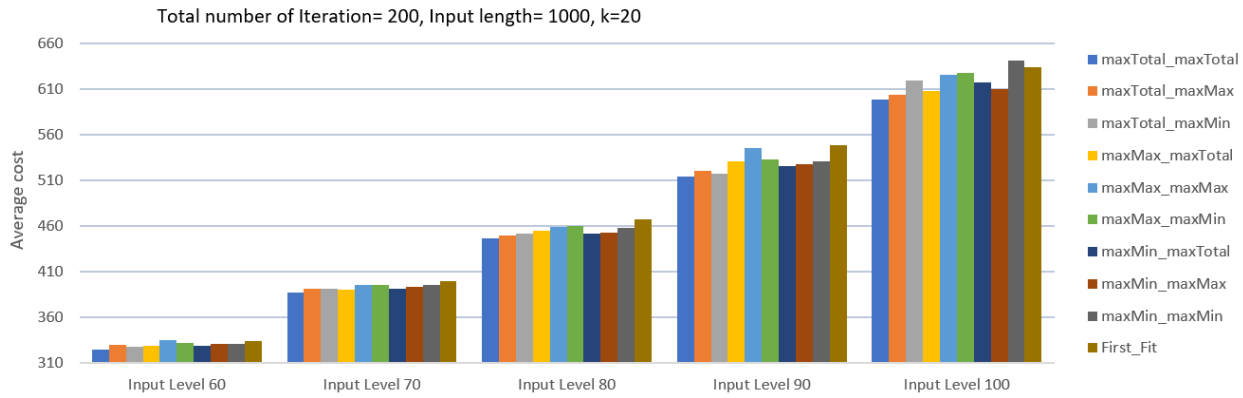
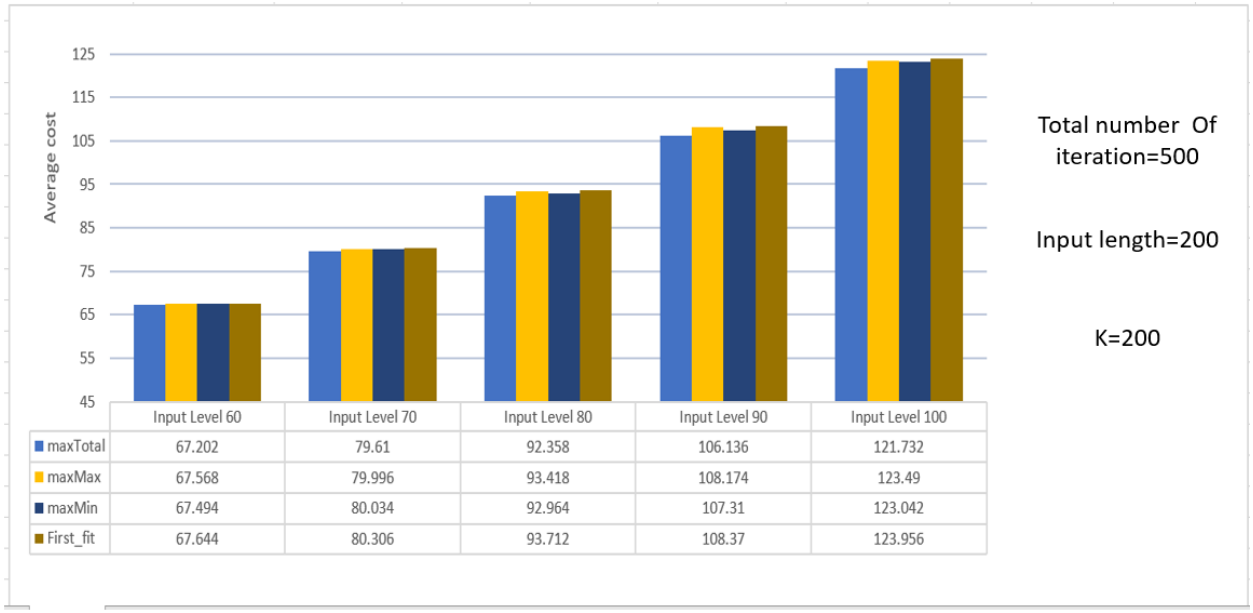


Fig9. Average cost of the different heuristics when k=20, the sequence having length=1000 and run for 200 times.

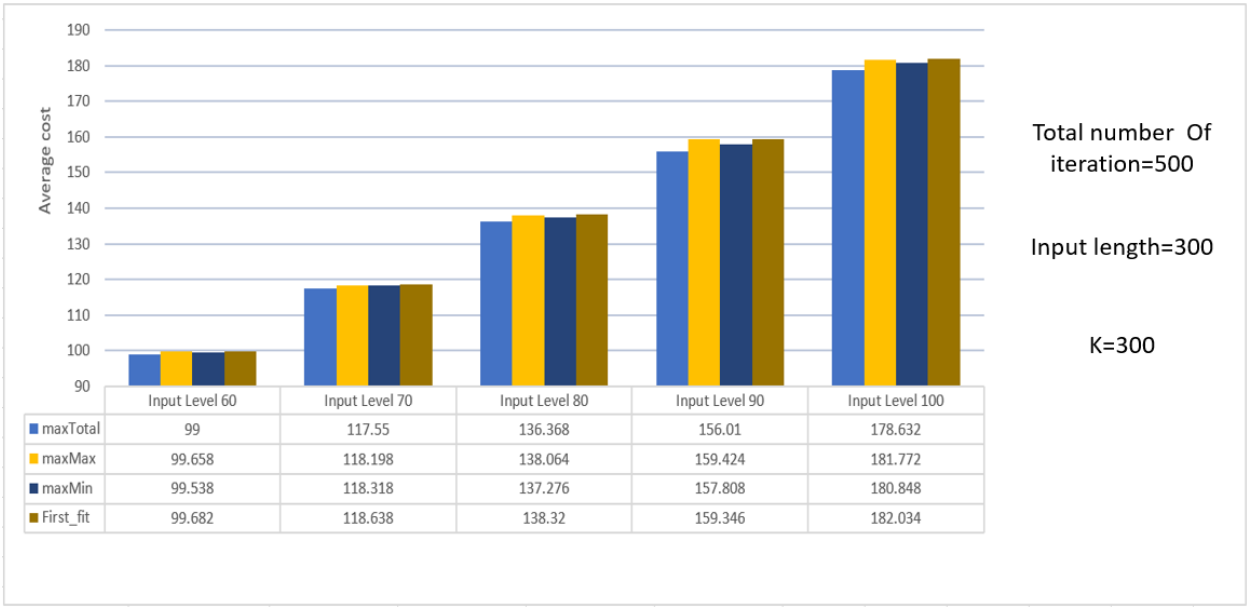
By testing the algorithms using different parameters for bounded-space setting, we found that maxTotal_maxTotal requires the least number of bins on average for packing the items. On the other hand, the first-fit algorithm requires more number of bins as compared to any heuristic of the best-fit algorithm (Rarely First fit is better than maxMin_maxMin algorithm).

We have conducted the experiments to test the performance of algorithms for the unbounded-space setting also. Figure 10 and 11 show the outcomes of two experiments of unbounded-space setting. From Figure 10 and 11, we can see that max_Total has the least cost in terms of the number of opened bins in the unbounded-space setting. However, max_Min performs better than max_Max in average case. First fit is the worst algorithm on an average for the unbounded-space setting.



	Input Level 60	Input Level 70	Input Level 80	Input Level 90	Input Level 100
maxTotal	67.202	79.61	92.358	106.136	121.732
maxMax	67.568	79.996	93.418	108.174	123.49
maxMin	67.494	80.034	92.964	107.31	123.042
First_fit	67.644	80.306	93.712	108.37	123.956

Fig10. Average cost of the different heuristics when k=200, the sequence having length=200 and run for 1000 times.



	Input Level 60	Input Level 70	Input Level 80	Input Level 90	Input Level 100
maxTotal	99	117.55	136.368	156.01	178.632
maxMax	99.658	118.198	138.064	159.424	181.772
maxMin	99.538	118.318	137.276	157.808	180.848
First_fit	99.682	118.638	138.32	159.346	182.034

Fig11. Average cost of the different heuristics when k=200, the sequence having length=200 and run for 1000 times.

Anomaly Detection Result

Finally, we performed the anomaly test for maxTotal_maxTotal algorithm in bounded-space setting. It is expected that if we increase the bounded value k for the same input sequence, the algorithm should require less bins to pack the items. Sometimes some algorithms do not behave like that. On the contrary, the cost of the algorithm increases when k increases. maxTotal_maxTotal is such an anomalous algorithm. We kept the same input sequence and changed the bounded value k gradually to find the anomaly behaviour of maxTotal_maxTotal algorithm.

For an input sequence σ of 50 lengths, we found that maxTotal_maxTotal requires 25 bins to pack those items when k=3. We increased the value of k and detected that maxTotal_maxTotal requires 26 bins when k=4. It is an anomaly behavior of maxTotal_maxTotal algorithm for following sequence.

σ = (7,49) (5,48) (67,11) (47,33) (46,69) (57,54) (37,47) (4,4) (18,11) (28,60) (73,40) (46,3) (23,62) (21,11) (54,71) (27,54) (57,42) (5,32) (22,46) (40,20) (49,14) (80,61) (77,67) (67,22) (67,1) (15,27) (42,24) (13,69) (15,31)(3, 11) (23,44) (76,61) (56,0) (74,69) (4,54) (22,23) (22,74) (70,22) (11,57) (50,56) (50,7) (28,22) (29,52) (18,53) (18,49) (69,49) (15,45) (62,52) (14,49) (41,80)

6. Conclusion

After performing the empirical evaluation, it was found that any change either in bounded space k or in length of input sequence or in the number of iterations had no impact on comparative performances of the algorithms on average. However, maxTotal_maxTotal performs the best in the average case. Max_maxTotal and maxTotal_maxMax algorithm also provide promising result in average case as compared to all other algorithms for vector bin packing. Furthermore, anomalous behaviour was found for one of the algorithms of vector bin packing instance when bounded value k was changed from 3 to 4. Our work was based on a uniform random sequence. In future, the same experimental study will be extended to the input sequence having a different distribution such as Markov chain, stationary distribution etc. In current work, the online vector bin packing was performed with the bins having 2-dimensions. However, it will also be extended to multidimensions and we will compare the result of vector bin packing in that case. We had found anomalous behaviour for only one algorithm of vector bin packing for the input sequence of length 50. To check the anomalous behaviour of the best-fit algorithm, other heuristics of the best-fit algorithm will be tested in future. We will also try to find the anomalous behaviour of the heuristics by minimizing the length of the input sequence.

7. REFERENCES

1. Coffman, Edward G., Gabor Galambos, Silvano Martello, and Daniele Vigo. "Bin packing approximation algorithms: Combinatorial analysis." In Handbook of combinatorial optimization, pp. 151-207. Springer, Boston, MA, 1999.

2. Graham, Ronald L. "Bounds on multiprocessing timing anomalies." SIAM journal on Applied Mathematics 17, no. 2 (1969): 416-429.

3. Goldberg, Robert P., and Robert Hassinger. "The double paging anomaly." In Proceedings of the May 6-10, 1974, national computer conference and exposition, pp. 195-199. ACM, 1974.

4. Chekuri, Chandra, and Sanjeev Khanna. "On multidimensional packing problems." SIAM journal on computing 33, no. 4 (2004): 837-851.

5. Galambos, Gabor, and Gerhard J. Woeginger. "Repacking helps in bounded space on-line bind-packing." Computing 49, no. 4 (1993): 329-338.

6. Coffman, Edward G. "Approximation algorithms for bin packing: a survey." Approximation algorithms for NP-hard problems (1997).

7. Kou, Lawrence T., and George Markowsky. "Multidimensional bin packing algorithms." IBM Journal of Research and development 21.5 (1977): 443-448.

8. Woeginger, Gerhard J. "There is no asymptotic PTAS for two-dimensional vector packing." Information Processing Letters 64.6 (1997): 293-297.

9. Vazirani, Vijay V. Approximation algorithms. Springer Science & Business Media, 2013.

10. Maruyama, K., S. K. Chang, and D. T. Tang. "A general packing algorithm for multidimensional resource requirements." International Journal of Computer & Information Sciences 6.2 (1977): 131-149.

11. Spieksma, Frits CR. "A branch-and-bound algorithm for the two-dimensional vector packing problem." Computers & operations research 21.1 (1994): 19-25.

12. Seiden, Steven S. "On the online bin packing problem." Journal of the ACM (JACM) 49.5 (2002): 640-671.

13. Azar, Yossi, et al. "Tight bounds for online vector bin packing." Proceedings of the forty-fifth annual ACM symposium on Theory of computing. ACM, 2013.

14. https://drive.google.com/open?id=1SjSW8vZR_G6fTi0ENnfaNrMMZQ7SKjJc

TABLE I: Pseudocode of best fit algorithm

```
BestFitAlgorithm()
1.K ← Maximum number of open bins at any time
2.Initialize the size of a bin                                   //100 for each dimension
3. for t=0,1,2.....tn-1,tn
   • Z_t(x_t, y_t) ← current request                             // x_t=[0,100] , y_t=[0,100]
   • fullestBin1 ← FindTheFullestBinForPlacingItem (Z_t)         // for placing an item
   • if fullestBin1 ==null                                       // no open bin has enough space for placing Z_t
       if |total number of open bin at t-1|==k
          fullestBin2 ← FindTheFullestBinForClosingAbin()        // for closing an bin
          close fullestBin2
       Open a new bin and place Z_t
   • else place Z_t at fullestBin1
```

TABLE II: Pseudocode of placing an item in best fit

```
FindTheFullestBinForPlacingItem (Z_t)
//Considering Maximum of Sum of x and y co-ordinate of bins
1. fullestBin ←null
2. i=1 to |total open bins at t-1|
   if bin_i.X+ x_t ≤ 100 && bin_i.Y+ y_t ≤ 100
     if fullestBin==null
       fullestBin← bin_i
     else
       if Total(bin_i) ==Total(fullestBin)
          fullestBin ←max(MaxCoOrdinate(bin_i),
                          MaxCoOrdinate(fullestBin))
       else if Total(bin_i)>Total(fullestBin)
          fullestBin ← bin_i
3. return fullestBin
```

```
FindTheFullestBinForPlacingItem (Z_t)
//Considering Maximum of Maximum co-ordinate of bins
1.fullestBin ← null
2.i=1 to |total open bins at t-1|
   if bin_i.X+ x_t ≤ 100 && bin_i.Y+ y_t ≤ 100
     if fullestBin==null
       fullestBin← bin_i
     else
       if MaxCoOrdinate(bin_i) == MaxCoOrdinate(fullestBin)
          fullestBin ←max(MinCoOrdinate(bin_i),
                          MinCoOrdinate(fullestBin)),
       else if MaxCoOrdinate(bin_i) > MaxCoOrdinate(fullestBin
          fullestBin ← bin_i
3. return fullestBin
```

TABLE III: Pseudocode of placing an item (left) and closing a bin (right) in best fit

```
FindTheFullestBinForPlacingItem (Z_t)
//Considering the Maximum of Minimum co-ordinate of bins
1.fullestBin ← null
2.i=1 to |total open bins at t-1|
    if bin_i.X+ x_t ≤ 100 && bin_i.Y+ y_t ≤ 100
      if fullestBin==null
        fullestBin← bin_i
      else
        if MinCoOrdinate(bin_i) == MinCoOrdinate(fullestBin)
          fullestBin← max(MaxCoOrdinate(bin_i),
                          MaxCoOrdinate(fullestBin))
        else if MinCoOrdinate(bin_i) > MinCoOrdinate(fullestBin)
          fullestBin ← bin_i
3. return fullestBin
```

```
FindTheFullestBinForClosingAbin()
//Considering Maximum of Sum of x and y co-ordinate of bins
1.fullestBin ← bin_1
2.i=2 to |total open bins at t-1|
        if Total(bin_i) ==Total(fullestBin)
          fullestBin ← max(MaxCoOrdinate(bin_i),
                           MaxCoOrdinate(fullestBin))
        else if Total(bin_i)>Total(fullestBin)
          fullestBin ← bin_i
3. return fullestBin
```

TABLE IV: Pseudocode of closing a bin in best fit algorithm

```
FindTheFullestBinForClosingAbin ()
//Considering Maximum of Maximum co-ordinate of bins
1. fullestBin← bin_1
2. i=2 to |total open bins at t-1|
        if MaxCoOrdinate(bin_i) == MaxCoOrdinate(fullestBin)
          fullestBin ←max(MinCoOrdinate(bin_i),
                          MinCoOrdinate(fullestBin)),
        else if MaxCoOrdinate(bin_i) > MaxCoOrdinate(fullestBin)
          fullestBin ← bin_i
3. return fullestBin
```

```
FindTheFullestBinForClosingAbin ()
//Considering Maximum of Minimum co-ordinate of bins
1. fullestBin ← bin_1
2. i=2 to |total open bins at t-1|
        if MinCoOrdinate(bin_i) == MinCoOrdinate(fullestBin)
          fullestBin ← max(MaxCoOrdinate(bin_i),
                           MinCoOrdinate(fullestBin)),
        else if MinCoOrdinate(bin_i) > MinCoOrdinate(fullestBin
          fullestBin← bin_i
3. return fullestBin
```