Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 2 – No.8, September 2015 – www.caeaccess.org

Bin Packing Algorithms with Applications to Passenger Bus Loading and Multiprocessor Scheduling Problems

Taiwo O. Ojeyinka
Department of Computer Science,
Adekunle Ajasin University, Akungba-Akoko,
Ondo State, Nigeria

ABSTRACT
The problem of arranging items into bins in order to minimize the overall number of used bins was implemented and evaluated using the online and offline bin packing heuristics. The study implemented the *fill* function to evaluate the performance of both the online and offline variants of bin packing heuristics. The algorithms were applied to some commonly occurring NP-hard problems whose solutions require optimisation e.g. the passenger-bus scheduling and the multiprocessor job scheduling problems. The results of the evaluation show that the minimisation function varied w.r.t. the sizes of the items and also of the bins. Results further shows that a minimised resource allocation and makespan are feasible for the passenger-bus scheduling and the multiprocessor job allocation respectively.

Keywords
Bin packing, resource allocation, heuristics, NP-hard, passenger-bus, multiprocessor

1. INTRODUCTION
NP-hard problems are computationally intensive problems which use a set of algorithms known as heuristics whose solutions may not be optimal use algorithms that do not guarantee an optimal solution. A heuristic is an "intuitive" way to find a valid and often reasonably good solution for a given problem in a "reasonable" lapse of time [10]. In this study heuristics are those procedural algorithms that pack items straight into bins, there is no high-level heuristic guiding other heuristics, nor is more than one repacking of items allowed. Additionally, the solution is certain to be feasible.

For instance in a job shop scheduling problem in which *n* jobs of varying sizes are scheduled on *m* identical machines with the objective of minimizing the total processing time of all the jobs, one heuristic to solve the problem might be first-fit, or first-fit decreasing, or first-fit increasing, or best-fit or any of its variants etc. However, achieving the best solution to the problem is very difficult with heuristics hence they are not considered as general approaches. Heuristics are considered the best solution when time of execution is considered an essential factor, but they can produce very poor results especially when quality is taken as an important into consideration because they depend on the nature and the instance of the problem. In the Bin Packing Problem, a number of heuristics exist from first-fit to best-fit or worst-fit and other variants. Each of these heuristics consists of putting objects of varying sizes (or weights) into the first or best or worst bin respectively (depending on the sizes of the bins), that can take it. Each one has shown to give good results in each case though not necessary the optimal level.

For the purpose of this study, two problems which can be solved with heuristic algorithms are presented: 1) passenger-bus allocation and 2) multiprocessor job allocation problems. The goal of this study is to minimise the total number of identical bins required to pack a number of items based on the heights or the weights of the items, and use the procedure to provide efficient. 1) Allocation of passengers to buses: the objective is to allocate efficiently buses to finite set of passengers in such a way that the overall number of buses and the total number of seats occupied are minimal. 2) Improved job allocation strategy for multiprocessor scheduling problems: the objective here is to allocate processors to a set of processes in a multiprocessor such that the overall makespan is minimal.

Generally, packing problem is a daily affair which is inevitable as people have need of packing items from place to place hence the necessity of bin packing algorithm. Optimizing the solution to bin packing problem can therefore not be overlooked. This study will expose the application of bin packing heuristic algorithms and also enhance their uses in scheduling and resource allocation problems since the heuristics represent a set of approximation algorithms to some NP-hard problems. The work considers the process of allocating a finite set of items into bins using a specified number of heuristic algorithms.

2. THE BIN PACKING ALGORITHM
Bin packing problem has been studied since the early 70's and different variants of the problem continue to attract researchers' attention. The bin packing algorithm is a general assignment problem often found in optimisation methods in computer science and operations research. According to [3], the bin packing problem is defined as follows:

> *"Given a finite collection of n sizes (or weights) s_1, s_2, s_3, ... , s_n, and a collection of identical bins with capacity C (which is greater than the largest of the sizes), find is the minimum number k of bins into which the sizes can be placed without packing beyond the bin capacity C?"*

The bin packing problem consists of *k* number of identical bins each of capacity C for which a finite collection of objects with sizes s_1, s_2, s_3... s_n are required to be stored and whose total sum is greater than the bin's capacity. In this study, the capacity *C* and the sizes *s* are positive integers and each individual size is less than the bin capacity *C*. The aim is to find the minimum number of bins that can accommodate all the items (either in terms of their weights or sizes). This case is the one-dimensional bin packing problem, but there are other variants of bin packing problem such as 2D packing, linear packing, packing by weight, packing by cost, etc. A typical application of the bin packing problem, which is closely related to the problem under study, is that of loading trucks with weight capacity constraints. Other applications are filling up containers, and creating file backups in removable media. The two dimensional bin packing (2BP) considers packing of items with both the size and the weight of each item during packing

Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 2 – No.8, September 2015 – www.caeaccess.org

in order to achieve the same objective of using a minimum number of bins as the one dimensional problem, [2]. Typical applications of 2D packing includes transportation in airlines luggage loading or the loading of varying sizes of containers on the ship. Industrial applications includes optimal cutting of ceramics, glass, wood etc. [4]. Three dimensional bin packing (3BP) problem also exists. It is much more complex than the 2BP but it is less studied.

The classical bin packing problem is given as a set of items I with a size or weight function *s/w*: $I \rightarrow (0, 1]$. The objective here is to pack the items into as few unit sized bins as possible. The packing is known as online if the order of bins and the packing are not known in advance. This has resulted into variants of online bin packing. First, the classical *online bin packing* addresses a non-deterministic arrival of items over time and are required to be packed as they arrive. The *dynamic bin packing* problem is such that allows items to be removed or depart over time. The model has applications in 1.) Pick-and-drop bus transit system, the operating system loading of programs on the memory, reading and writing files on random access disks etc. When already packed items are allowed to be rearranged, the packing is known as *relaxed online bin packing.* A dynamic bin packing with repacking is called fully *dynamic bin packing*. See Table 1 for an overview of different models.

Table1: Online bin packing models

Name	Deletion	Repacking
Online Bin Packing	✗	✗
Relaxed Online Bin Packing	✗	✓
Dynamic Bin Packing	✓	✗
Fully Dynamic Bin Packing	✓	✓

A number of heuristics are available for solving the bin packing problem.

a. The First-Fit (FF) algorithm places a new object of weight w_i in the first bin that has space to accommodate it.

b. The Last-Fit (LF) algorithm follows the first-fit except that it places a new object in the last bin.

c. The Next Fit (NF) algorithm places a new object in the next bin, starting a new bin if necessary.

d. The Best Fit (BF) algorithm places a new object in the most-filled bin that still has space to accommodate it.

e. The Worst Fit (WF) algorithm places a new object in the least-filled existing bin.

f. The Almost Worst Fit (AWF) algorithm places a new object in the second least-filled bin.

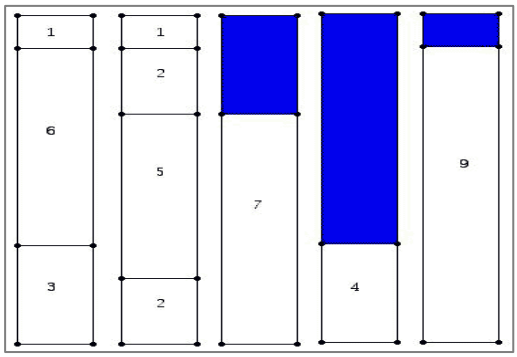

Fig. 1: A schematic diagram of 10 items packed into 5 bins

A straightforward approach to solving this type of problem is to divide the total sum of the sizes by the capacity of the bins. [3]. Assume integer numbers, the minimum number of bins is:

$$\left\lceil \frac{\varphi}{C} \right\rceil = \sum_{i=1}^{n} w_i \qquad \textit{Eq. 1}$$

Using either FF, WF, NF or BF the required amount of time to compute the minimum number of bins is $n \log n$ where n is the number of sizes [3].

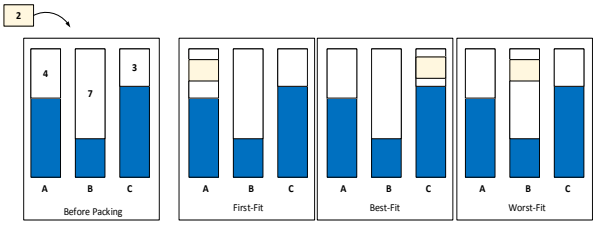

Fig. 2: Example illustrating the bin packing algorithms.

As an illustration in Fig 2, suppose that there are currently three bins which have capacity 10 and have *unused space* as follows: Bin A, 4 units, Bin B, 7 units, and Bin C with 3 units. Suppose the next arrived item in the list is of size 2. By using First Fit the item is put in Bin A, Best Fit puts it in Bin C, and Worst Fit puts it in Bin B.

The offline bin packing problem is one where all the sizes of the items are known before the packing. Several heuristics have been studied to this end for the packing of items using the offline BBP. Unlike the online algorithm which is a real time problem, the offline variant is not real time.

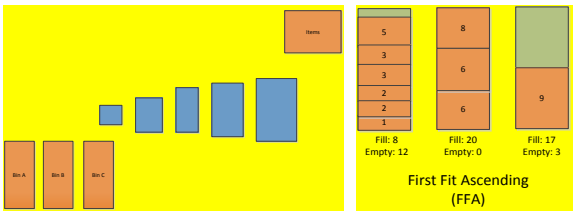

Fig. 3: An Offline First Fit Ascending heuristic

Various variants of heuristics have been applied which includes first fit, next fit, best fit, first fit decreasing etc. to mention a few. Items are packed into the bin in such a way that the item are packed with minimum number of bins.

Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 2 – No.8, September 2015 – www.caeaccess.org

Table 2: Types of Bin Packing Problems

Types	Static	Dynamic
Online	1D	1D
	2D	2D
	Strip Packing	Strip Packing
Offline	1D	1D
	2D	2D
	Strip Packing	Strip Packing

2.1 Bin Packing Problem as an Optimisation Problem

NP-hard problems are usually solved using heuristics algorithms which try to reach an optimal solution, or at least a solution as close as possible to one optimal in a reasonable time. The bin packing problem determines if the weight w_i in a list L of items can be packed into an integer D, referring to the number of bins of capacity C. This problem is NP-complete if it is proposed as a decision problem. However, as an optimisation problem the problem reduces to NP-hard. A viable way of solving the problem is to find an approximate optimal solutions for the bin packing problem in polynomial time. The bin packing theorem states thus [11]:

Given: a finite number of n items to be placed in bins of capacity C each.

Item: Each item i has x units of size.

The objective functions are: 1) minimize the number of bins required to pack all the *n* objects and 2) maximize the number of empty spaces left after all *n* objects have been accommodated.

3. MODEL DEVELOPMENT

For this research work items which have different weights are packed at point of arrival into the bin. Using the bin packing algorithms item are allocated to resources using both the online and the offline bin packing heuristics. Results obtained are evaluated using the fill function for both the online and offline algorithms. VBA and Excel were used to implement the algorithms.

3.1 The "Fill" Function

The most obvious objective function for this problem is the number of items used by the solution, but it does not create smooth search space for optimisation [13]. To make search space smooth, function that takes the fill of bins in the solution into account is used and it looks like this:

$$f_p = \frac{\sum_{i=1}^{n}\left(\frac{f_i}{C}\right)^k}{n} \qquad \text{Eq. 2}$$

$\boldsymbol{f}_p$: Fill function, $\boldsymbol{n}$: number of bins, $\boldsymbol{f_i}$: fill of the $\boldsymbol{i}$th bin, $\boldsymbol{c}$: capacity of the bin, $\boldsymbol{k}$: constant greater than 1.

3.2 Model Development for the Passenger Bus Loading Problem

In this project, a bin packing algorithm is used to develop efficient solution for passenger bus loading system to reduce wastes of available bus spaces across the fleet.

For instance, Table 3 shows 188 passengers loaded into certain number of 25-seater buses. The number 25 (0) denotes 25 passengers are loaded into a 25-seater bus leaving 0 empty space. From the table, Loading 1 and Loading 3 leave 12 spaces each. Loading 1 is more efficient than Loading 3 whose empty spaces have been fragmented. However, the two loadings 1 and 3 are more efficient than Loading 2 which used additional bus making a total of 9 buses instead of 8.

In bin packing, there are a number of bins with certain capacity, where the task required is to arrange objects of various sizes into the bins. In this case, the idea is not only to use the least possible number of buses but also to maximize the number of empty seats across the fleet. In a more complicated instance, there can be a group of passengers (e.g. 2, 3, or 5 etc.) who want to be in the same bus for certain reasons, that is, there may be a number of tour groups of various sizes, which needs to be accommodated all with the fewest number of empty seats across the fleet. In this case, the first step here may be to develop an exhaustive search tree by using every possible combinations of passenger groups and increasing the number of buses until there remains no more passengers to load.

Table 3: Example of 188 passengers loaded into 25-seater buses

BUS	Loading 1	Loading 2	Loading 3
1	25 (0)	25 (0)	25 (0)
2	25 (0)	23 (2)	24 (1)
3	25 (0)	19 (6)	23 (2)
4	25 (0)	19 (6)	23 (2)
5	25 (0)	17 (8)	23 (2)
6	25 (0)	24 (1)	23 (2)
7	24 (1)	23 (2)	24 (1)
8	14 (11)	22 (3)	23 (2)
9		16 (9)	
	188 (12)	**188 (37)**	**188 (12)**

For the online algorithm, a group of passengers or at least an individual arrives at a station at random. Passengers or group of passengers are loaded on the bus in the FCFS order of arrival. Each algorithm i.e. FF, NF, BF, WF is implemented to load item groups into the buses such that the first arrived individual or group is loaded into the first bus and so on. The following assumptions are given for the implementation:

- Each bus has a maximum capacity C to load passengers. No further loading is allowed when the bus is filled up.
- The number of available buses is limitless.
- Passengers can exist as an individual or as a group of people. The number of passengers within a group is represented by the integer value of passenger loaded on the bus.
- The number of individual passenger within a group cannot be greater than C.
- The total number of all the passengers is greater than the capacity C of the bus.
- Passengers cannot be relocated once they have been allocated to a bus (repacking is restricted).
- There is no specific order of selecting a bus. All buses have the same capacity and thus any bus can be selected at random.

The four online algorithms are implemented with increasing bus capacities. The objective (*fill*) function is calculated for each online algorithm to know which bus has the highest unused space evenly distributed across the buses. The algorithm with the highest unused space is considered the best. Similar implementations were adopted for the offline versions: FFA, FFD, NFA, NFD, BFA, BFD, WFA, and WFD as shown in Table 2.

3.3 Model Development for the Multiprocessor Scheduling Problem

Multiprocessor scheduling (or minimum makespan scheduling) problem is a scheduling problem in computer science and operations research in which several tasks having different length of processing are scheduled on a multiprocessor such that the finishing time of the last task (also called makespan) is minimized. In this problem, m identical processors $P_1 \ldots, P_m$ and n tasks T_1, J_2, T_n are given. Task T_i has a processing time $t_i \geq 0$ and the goal is to assign tasks to the processors so as to minimize the total makespan. The problem of study here considers the non-pre-emptive scheduling for FCFS, SJF and LJF using a single central queue.

Algorithm 1: Order (list) the tasks arbitrarily
For i = 1 to n do
Assign task Ti to the machine with least current load
Update load of the machine that receives task Ti

Listing 1: A greedy algorithm for the online multiprocessor scheduling problem [1]:

We give the following assumptions for the implementation:

- Each processor has an unlimited processing capacity C to execute the tasks. Continuous loading on the processor is possible as long as there are availability of tasks to be scheduled.
- The number of available processors is limitless.
- Tasks are independent of one another.
- There is no processor idle time or wait time.
- The total number of all the tasks is greater than the number of processors ($T > P$).
- Tasks cannot be relocated once they have been allocated to any processor (i.e. no migration).
- The order of scheduling of tasks follows the operating systems scheduling disciplines.
- The order of scheduling of processors follows the bin packing heuristics. All processors are identical and homogeneous and thus any one can be selected with a desired algorithm.

3.3.1 The Objective Function

The scheduling assignments for the multiprocessor scheduling problem are in two stages: Task scheduling (allocation of tasks to processor and Processor scheduling (allocation of processor for scheduled tasks).

3.3.2. Task scheduling

Task schedule is a process that decides which task is next to be scheduled on the processor. A schedule of this nature naturally follows the disciplines in the operating system schedule of task of processors viz: the First-Come-First-Serve (FCFS), Shortest Job First (SJF), Shortest Time Remaining First (STRF), Round Robing (RR), Priority (P) etc. [14]. For the purpose of this study, three of these policies considered: FCFS, SJF and LPT.

3.3.2.1 First-Come-First-Serve (FCFS): The FCFS schedules tasks in the order of arrival into the ready queue of operating system. This is likened to the online algorithm such that the order of arrangement of tasks is not known a priori. The FCFS task is defined by:

$$T_i = \text{first}_j\{T_j\} \qquad \textit{Eq. 3}$$

3.3.2.2 Shortest Job First (SJF): SJF is a non-pre-emptive scheduling in which tasks with the shortest execution tasks are scheduled before tasks with longer executing tasks. The SJF task is defined by:

$$T_{min} = \min_j\{T_j\} \qquad \textit{Eq. 4}$$

3.3.2.3 Longest Processing Time (LPT): LPT sorts *n* tasks in descending order of processing time. It is a non-pre-emptive scheduling algorithm. The LPT task is defined by:

$$T_{max} = \max_j\{T_j\} \qquad \textit{Eq. 5}$$

3.3.3 Processor scheduling

Processor schedule is a process that decides on which processor a scheduled task should run. The objective is to simply minimize the latest finishing time. The minimal processing time solution will have the cost:

$$t_p = \min_k \max_i \left(\sum_{j=1}^{m} T_j + \sum_{k=1}^{z} O_{jh} \right) \qquad Eq. 6$$

where k = processor scheduling policy, j = task, O = overhead of task migration (if any), and h = the number of migration per task.

According to our methodology from *Eq. 2*, the total makespan for this scheduling assignment is given in *Eq. 7*.

$$t_p = \sum_{t=1}^{n} \left(\frac{f_i}{C}\right)^k \qquad \textit{Eq. 7}$$

For the evaluation criteria, it is more desirable to find the minimum or maximum values of turnaround time, throughput, and efficiency rather than their averages.

3.3.3.1 Turnaround Time: is the average time elapsed from when the first task is scheduled to the time it completed. This is the same as the makespan.

3.3.3.2 Throughput: In this context, throughput is defined as the number of tasks completed in a time unit. This is the same as the number of tasks over the total

$$\gamma_p = \frac{n}{t_p}. \qquad \textit{Eq. 8}$$

3.3.3.3 CPU Efficiency: the efficiency of scheduling in this study relates to the proportion of unused space to the total maximum allocated space. Ideally, the efficiency is calculated thus:

$$E = \frac{1-f}{OPT} \qquad \textit{Eq. 9}$$

Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 2 – No.8, September 2015 – www.caeaccess.org

4. RESULTS

Results of implementation show an overlap between FF and BF algorithms as the bin capacity increases. In Figure 5 each online algorithm was compared with its offline ascending and descending versions. From the results both FF and BF show better performance over NF and WF at low bin capacities <= 50 but the performances of NF and WF got better as the bin capacity increases.

For large bus capacities, the performance of NF and WF get better than for FF and BF. The FFA and FFD had their performances improved as the bin size increases. However, WFA got better than WFD for large bus capacities. Similar results were obtained for ascending and descending offline algorithms for FF, BF, NF are shown in Figure 1.

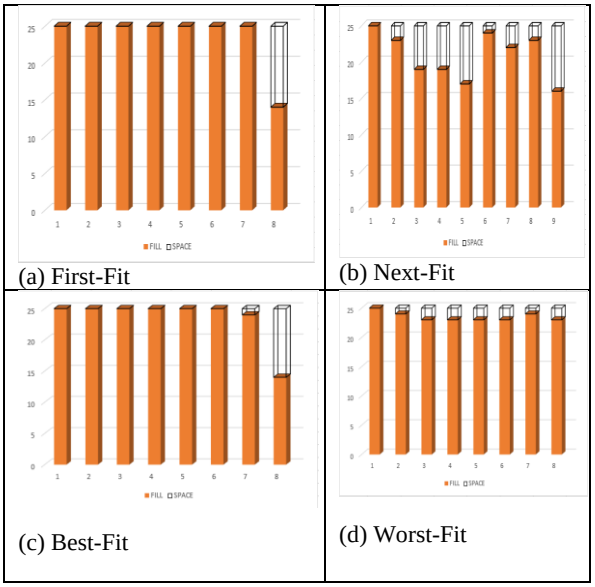

Fig. 4: Fill function of online bin-packing algorithms for Bus Capacity = 30. (Lower is better)

Table 4: *1-f(x)* function values of passenger bus loading problem for Bus Capacity = 30. (Higher is better)

Algorithm	Online	Offline	
		Ascending	Descending
First Fit	0.139365	0.1943	0.1327
Next Fit	0.193	0.194	0.186
Best Fit	0.139365	0.194286	0.132698
Worst Fit	0.190794	0.194286	0.175873

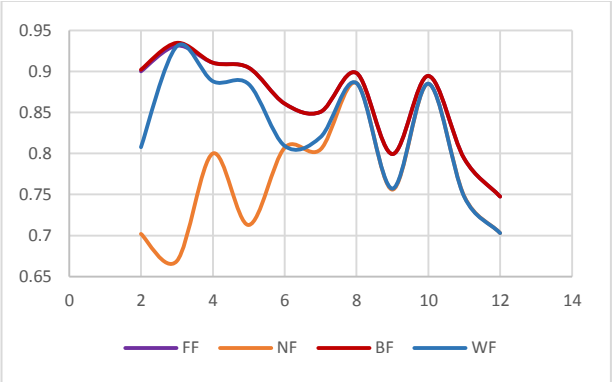

Fig. 5: Fill function of online heuristics with increasing bus capacities

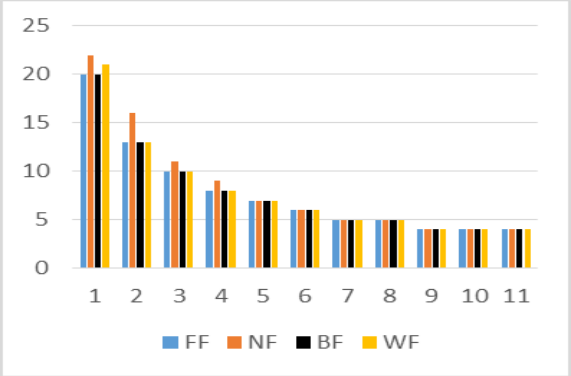

Fig. 6: *Fill* function of online heuristics with increasing number of buses

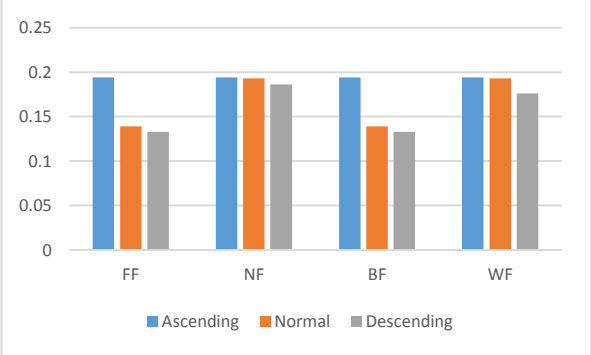

Fig. 7: Fill function of online and offline heuristics for Bus Capacity = 30. (Higher is better)

Table 5: Turnaround time, throughput and relative efficiencies for FCFS, SJF and LPT of a multiprocessor scheduling problem

Scheduling Algorithm	Allocation Heuristic	t_p	γ_p	E_p
SRT	FFA	6.02	33.33	0.10
	NFA	5.64	33.33	0.10
	BFA	6.02	33.33	0.10
	WFA	5.64	33.33	0.10
MIN		**5.64**	**33.33**	
FCFS	FF	5.64	33.33	0.04
	NF	5.64	31.23	0.10
	BF	5.64	33.33	0.04
	WF	5.64	31.23	0.10

MIN		**5.64**	**31.23**	
LPT	FFD	6.07	30.97	0.031
	NFD	5.70	33.01	0.091
	BFD	6.07	30.97	0.031
	WFD	5.77	32.59	0.079
MIN		**5.70**	**30.97**	

Results in Table 5 show the turnaround time, the throughput and the efficiency of multiprocessor scheduling policies for FCFS, SJF, and LPT under the assumptions considered above. Experimental results show that the Next Fit algorithm and its variants are better than other algorithms: NF algorithm performs better in FCFS; NFA is better for SJF while NFD leads in the case of LPT.

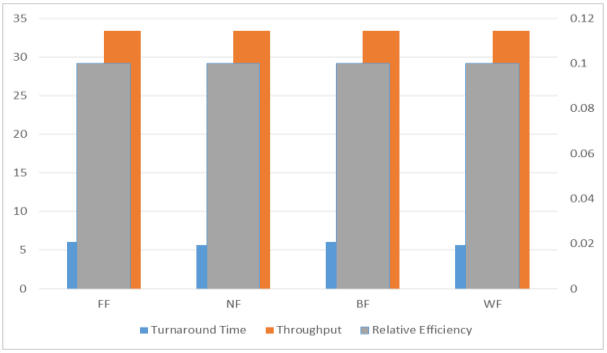

Fig. 8: Turnaround time, throughput and relative efficiencies for SJF

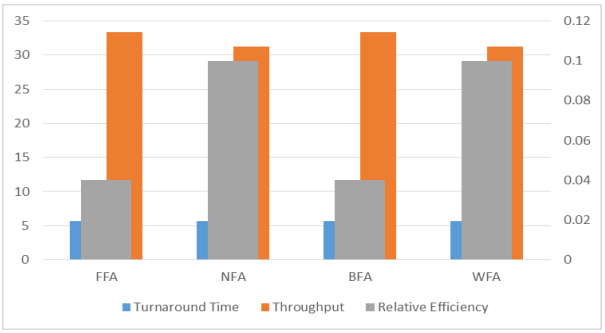

Fig. 9: Turnaround time, throughput and relative efficiencies for FCFS

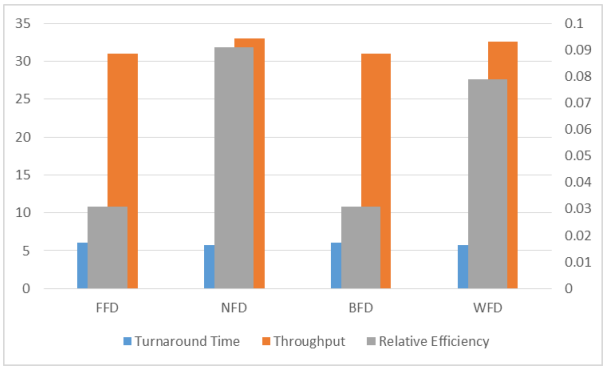

Fig. 10: Turnaround time, throughput and relative efficiencies for LPT

5. RELATED WORK

Quite a number of research has been carried into the application of bin packing problem. Reviews of related work was made and reported in this research work.

Corcoran A. L, Wainwright R. L. [8] developed a genetic algorithms LibGA to solve combinatorial optimization problems. The authors used bin packing Next Fit heuristic to obtain evaluation function for the genetic algorithm solution. A multiprocessor scheduler was discussed and implemented using Job Shop scheduling algorithm but not with bin packing method. The work proposed the possibility of achieving efficient scheduling of jobs on multiprocessor but it was neither treated nor implemented.

The work of Zapata, O. U. P., & Alvarez, P. M. [15] implemented a real-time scheduling algorithms to assign pre-emptible tasks to multiple processors. The bin packing algorithms were combined with real time multiprocessor scheduling policies, Rate Monotonic (RM) and Earliest Deadline First (EDF) scheduling policies which are different from the policies considered in this study.

6. CONCLUSION

The project work considered methods of evaluating the offline and online variants of bin packing problem using heuristics for deploying minimum objective function. Efficient packing of items is viewed as a tool for maximizing the utilization of the bins used. We considered bin packing problem and its applications to passenger bus loading and multiprocessor scheduling problems using fill function. Minimization of available resources and makespan for passenger bus loading problem and multiprocessor scheduler were implemented and then evaluated. The results obtained for passenger bus loading problem can be adapted to container or transportation loading system where minimum resources are required. Maximization of throughput and efficiency (as the case may be) for packing items into bin using this heuristics was considered.

In our proposed studies, we shall adopt this methodology for multi-objective minimization of complex problems for urban transit which addresses vehicles with different capacities and different amounts of fuel such that consideration for costs and profits are necessary. Similar approach to the one implemented for multiprocessor scheduling can be extended for thread scheduling in multicore CPUs. The proposed study will implement efficient scheduler for improved performance and lower energy consumption for multicore processors.

7. REFERENCES

[1] Albers, S., Charikar, M., & Mitzenmacher, M. 1998. Delayed information and action in on-line algorithms. In Foundations of Computer Science, November 1998. Proceedings. 39th Annual Symposium on (pp. 71-80). IEEE.

[2] Alvim, A. C. F., Ribeiro, C. C., Glover F. and Aloise, D. J. 2004. A Hybrid Improvement Heuristic for the One-Dimensional Bin Packing Problem, Journal of Heuristics, Vol. 10, No. 2, 205-229.

[3] American Mathematical Society 2010. Feature Column from the AMS http://paginas.fe.up.pt/~mac/ensino/docs/DS20102011/Bin_Packing.pdf.

[4] Ayachi, I., Kammarti, R., Ksouri, M., & Borne, P. 2013. A Genetic algorithm to solve the container storage space allocation problem. arXiv preprint arXiv:1303.1051.

[5] Bansal, N., Caprara, A., & Sviridenko, M. 2009. A new approximation method for set covering problems, with

applications to multidimensional bin packing. SIAM Journal on Computing, 39(4), 1256-1278.

[6] Coffman E. G, Galambos G, Martello S. & Vigo D. 1998. Bin packing approximation algorithms: Combinatorial analysis, pp. 151–208 in Du D-Z & Pardalos PM (Eds), Handbook of combinatorial optimization, Kluwer Academic Publishers, Boston MA.

[7] Coffman E. G, Garey. M. R. & Johnson, D. S. 1978. An Application of Bin-Packing to Multiprocessor Scheduling.. SIAM J. Comput., 7, 1-17.

[8] Corcoran A. L, Wainwright R. L. 1995. Using LibGA to Develop Genetic Algorithms for Solving Combinatorial Optimization Problems. Published in The Application Handbook of Genetic Algorithms, Volume I, Lance Chambers, editor, pages 143-172, CRC Press, 1995.

[9] Fleszar, K., and Hindi K. S., 2002. New heuristics for one-dimensional bin packing, Computers and Operations Research, Volume 29, Issue 7, pp. 821-839.glewood. Cliffs, N.J.

[10] Francq P. 2012. Optimization Problems. January, 2012 http://www.otlet-institute.org/wikics/Optimization_Problems.html

[11] Garey M. R. and Johnson D. S. 1979. Computer and Intractability: A guide to the Theory of NP-Completeness by Publisher: W. H. Freeman 1979.

[12] Gupta, J. N. D., and J. C. Ho. 1999. A new heuristic algorithm for the one-dimensional bin-packing problem, Production Planning and Control, Volume 10, Issue 6, pp. 598-603.

[13] Janković M., 2013. Genetic Algorithm for Bin Packing Problem. http://www.codeproject.com/Articles/633133/g

[14] Silberschatz A., Gagne G., and Galvin P. B. 2008, "Operating System Concepts, Eighth Edition ". John Wiley & Sons. July 2008. ISBN: 9780470128725.

[15] Zapata, O. U. P., & Alvarez, P. M. 2005. EDF and RM multiprocessor scheduling algorithms: Survey and performance evaluation. Seccion de Computacion Av. IPN, 2508.