ON-LINE OPEN-END BIN PACKING

Siu-chung Lau[*], *Gilbert H. Young*[+], *W. K. Kan*[*] *and Yu-Liang Wu*[*]

Abstract

The bin packing problem has been studied extensively since the 1970's, and it is known to be applicable to many different areas especially in operations research and computer science. In this paper, we present a variant of the classical bin packing problem which allows the packing to exceed its bin size but at least a fraction of the last piece is within the bin, and we call it open-end bin packing problem. This paper is focused on on-line open-end bin packing. An on-line open-end bin packing algorithm is to assign incoming pieces into the bins on-line, that is, there is no information about the sizes of the pieces in future arrivals. An on-line algorithm is optimal if it always produces a solution with the minimum number of bins used for packing. We show that no such optimal algorithm exists. We also present seven efficient on-line algorithms, Next Fit, Random Fit, Worst Fit, Best Fit, Refined Random Fit, Refined Worst Fit and Refined Best Fit, which give sub-optimal solutions. The performances of these algorithms are studied. A case study for the application of the studied problem is presented, and this is a practical problem on maximizing the savings of using stored-value tickets issued by Kowloon-Canton Railway (KCR) which is one of the major public transportation means in Hong Kong.

Keyword: Open-end Bin Packing, On-line, Algorithm.

1. Introduction

The class of one-dimensional bin packing problem has been studied extensively in the literature [1-18]. Because of its wide applications, it has drawn tremendous attention in the past two decades. Due to the NP-hardness of the problem, most of the studies have been concentrated on deriving efficient heuristics to produce sub-optimal solutions. Bin packing models a variety of problems encountered in operations research and computer science, from storing a set of records into the minimum number of fixed size units [4,6,7], to the problem of minimizing the number of processors with a given schedule length [4,8]. In this paper, we present a variant of bin packing problem which allows packing beyond its bin size provided that at least a fraction of the last piece is inside the bin. We call this family of bin packing problem the *open-end bin*

[*] Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. Email: {schlau,wkkan,ylw}@cse.cuhk.edu.hk, Tel.: 852-2609-8440, Fax: 852-2603-5032
[+] Department of Computing, The Hong Kong Polytechnic University, Hong Kong.

packing problem. See Figure 1 for the 1-dimensional open-1-end bin packing problem.

Figure 1

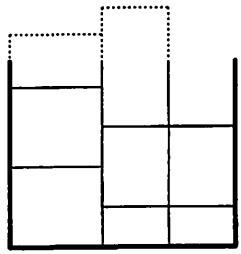

Figure 2

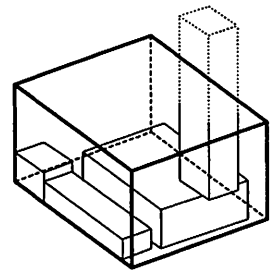

Figure 3

Note that if both ends of the bin are open (open-2-end), then it gives another variant of the bin packing problem. The formal definition of the multi-dimensional open-end bin packing problem is given in [19]. Figure 2 and Figure 3 are examples of 2-dimensional and 3-dimensional open-end bin packing problems. In this paper, we concentrate on the bin packing problem with only one end open. For off-line open-end bin packing, all pieces are available to be assigned at time zero. The complexity of the off-line open-end bin packing can be shown to be NP-hard. This problem models minimizing the number of processors needed to schedule a set of tasks by a given maximal starting time of all tasks.

For *on-line open-end bin packing*, it is computationally more difficult. An on-line open-end bin packing algorithm is to assign an incoming piece to a bin without any knowledge of future arrivals of pieces. In this paper, we concentrate on the on-line case. Results of the off-line case can be found in [20]. This problem models a real case as studied in Section 5.

In Section 2, we give the formal definition of on-line open-end bin packing problem studied in this paper. In Section 3, we present the nonexistence result of an optimal on-line open-end bin packing algorithm. In Section 4, we give seven on-line algorithms which give sub-optimal solutions and compare their performances. In Section 5, a case study of maximizing savings of using stored-value tickets issued by KCR in Hong Kong is shown, and the performance of different algorithms are compared. In the last section, we give some concluding remarks.

2. Problem Definition

In this section, we formally define the problem of open-end bin packing.

We are given a set of bins $\{B_1, B_2,..., B_k\}$, size of bin S, and a list L of m pieces $\{p_1, p_2,..., p_m\}$. We are required to assign these pieces to k bins. A bin is full if the total size of pieces assigned to it is more than or equal to S, that is, the last piece included in the bin may exceed the bin size S but at least a nonzero fraction of it is inside the bin. For on-line open-end bin packing, one needs to assign pieces to bins without knowing the sizes of the coming pieces. When a bin is full, a new bin will be added. That means we always have k bins available all the time. Again, we want to minimize the number of bins used for packing all pieces. In other words, we want to maximize the excess amount of the bins. So we define the saving percentage of a bin as the excess amount of bin i over the size of bin i.

In this paper, we define our performance measure *average saving percentage*, of any given on-line algorithm, A, as follows:

$$AverageSavingPercentage(A) = \frac{\sum_{i=1}^{n} SavingPercentageOfBin(i)}{n}$$

where n is the total number of bins used for packing all pieces.

An optimal on-line algorithm for open-end bin packing is the one which always produces a solution with the maximum average saving percentage. We are interested in finding an optimal on-line algorithm for open-end bin packing here. In the next section, we show that no such optimal algorithm can exist.

3. Nonexistence Result

Theorem 1: No optimal on-line algorithm can exist for the open-end bin packing problem.

Proof: Assume there is an optimal online open-end bin packing algorithm *OPT*, i.e., *OPT* always produces the largest saving percentage for all kinds of inputs. Considering two input sequences L_1:{0.4, 0.4, 0.6, 0.6} and L_2:{0.4, 0.4, 0.5, 0.9} which will be packed into two bins with sizes equal to 1. Obviously, the optimal packing for L_1 will have the 0.4, 0.4 and 0.6 pieces put into one of the bins and the remaining 0.6 piece into the other one. The optimal packing for L_2 will have the 0.4, 0.5 and 0.9 pieces put into one bin and the remaining 0.4 piece into the second bin. Suppose a sequence of four pieces with the first and second ones equal to 0.4 is to be packed by *OPT*. Without loss of generality, we can assume that *OPT* put the first 0.4 piece into the first bin. When handling the second 0.4 piece, *OPT* cannot determine whether to put it into the first or second bin since either choice may lead to a non-optimal solution. It is

contradictory to the assumption that *OPT* is optimal and that completes our proof.

4. Heuristics

From the nonexistence result in Section 3, we can only go for sub-optimal on-line algorithms for the open-end bin packing problem. In this section, we introduce seven sub-optimal on-line algorithms, and the comparison of their performances are discussed. They are:

Next Fit:
When there is an incoming item, we put it in the first available bin.

Random Fit:
When there is an incoming item, we put it into a bin randomly.

Worst Fit:
(1) When there is an incoming piece, if the maximum space among the bins is less than the size of the incoming piece, then go to step (2) else we put it in the bin with maximum space.
(2) Place the incoming piece into the bin with the minimum space.

Best Fit:
(1) When there is an incoming piece, if the maximum space among the bins is less than the size of the incoming piece, then go to step (2) else we put it in the bin which has the space just bigger than the size of the piece.
(2) Place the incoming piece into the bin with the minimum space.

Refined Random Fit:
1) When there is an incoming item, if the maximum space among the bins is less than the size of the incoming piece, then go to step (3) else go to step (2)
2) If (size of the minimum space bin + size of the piece) / size of the bin – 1 > satisfactory threshold go to step (3) else we put it into a bin randomly.
3) Put the piece into the bin with minimum space.

Refined Worst Fit:
1) When there is an incoming item, if the maximum space among the bins is less than the size of the incoming piece, then go to step (3) else go to step (2)
2) If (size of the minimum space bin + size of the piece) / size of the bin – 1 > satisfactory threshold go to step (3) else we put it into the bin with maximum space.
3) Put the piece into the bin with minimum space.

Refined Best Fit:

(1) When there is an incoming piece, if the maximum space among the bins is less than the size of the incoming piece, then go to step (3) else go to step (2).

(2) If (size of the minimum space bin + size of the piece) / size of the bin – 1 > satisfaction threshold go to step (3) else we put it in the bin which has the space just bigger than the size of the piece.

(3) Place the incoming piece into the bin with the minimum space.

Satisfaction threshold is a predefined value between 0 and 1.

In order to observe how these algorithms perform, some computational experiments were done with the following parameters.

- The bin sizes were set to 1.
- The sizes of the incoming items were uniformly distributed over the range (0,1].
- Each run consists of 5000 items.
- The process was repeated 1000 times and the mean of the average saving percentage of the 1000 repetitions was calculated.

The average percentage is plotted against number of bins, ranged from 1 to 50, in Figure 4. The thresholds of the refined algorithms are set to 10%, 30%, 50%, 70% and 90% respectively.

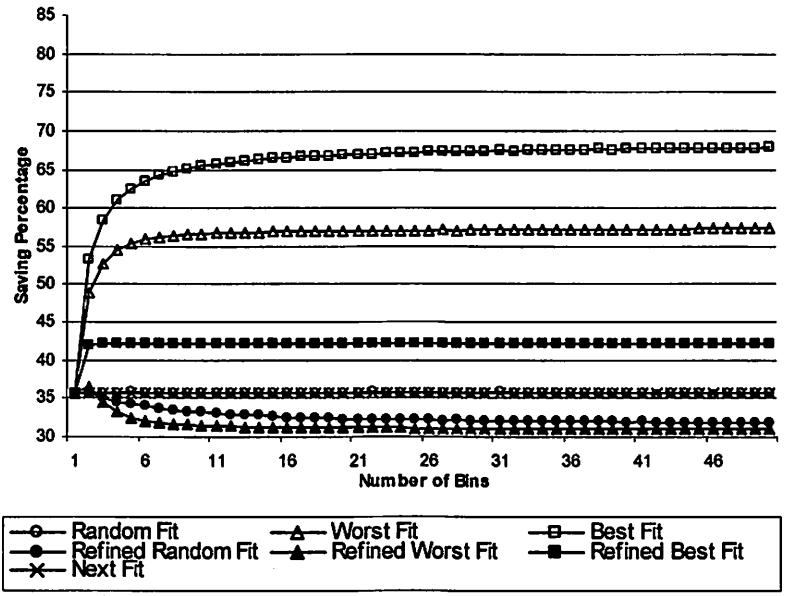

Figure 4(a): 10% threshold

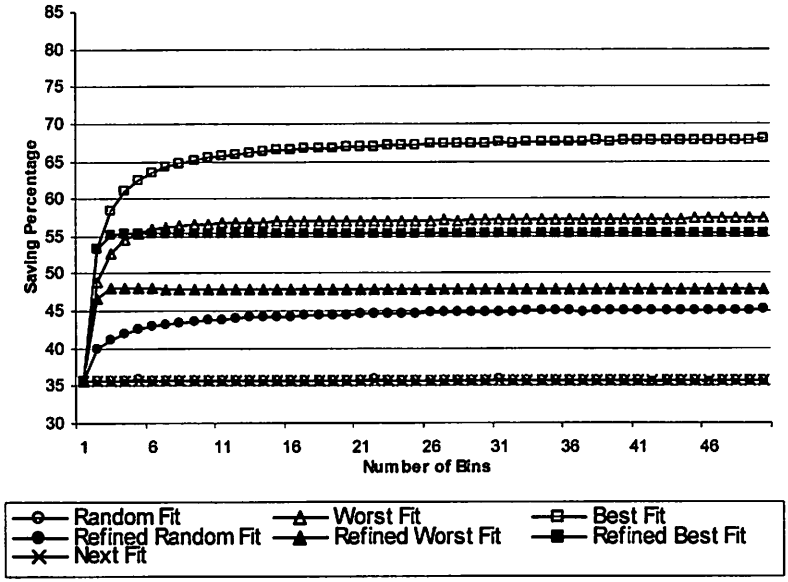

Figure 4(b): 30% threshold

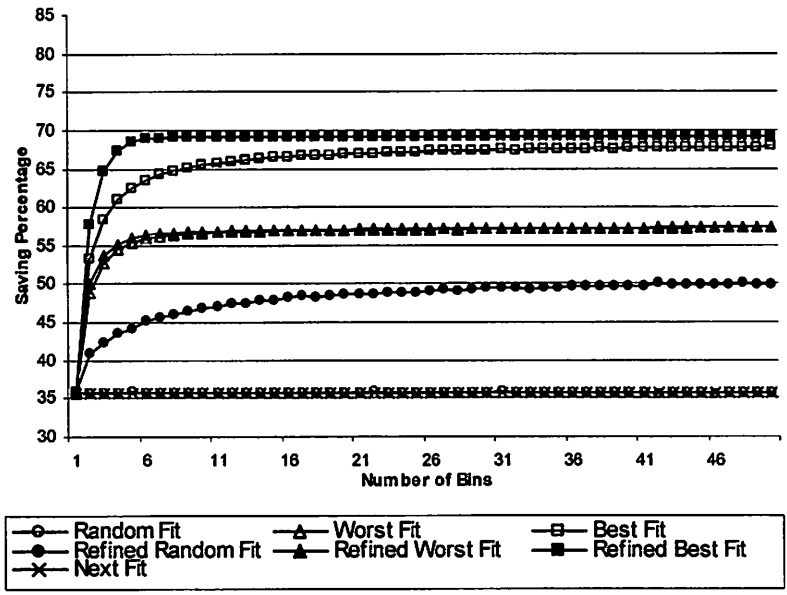

Figure 4(c): 50% threshold

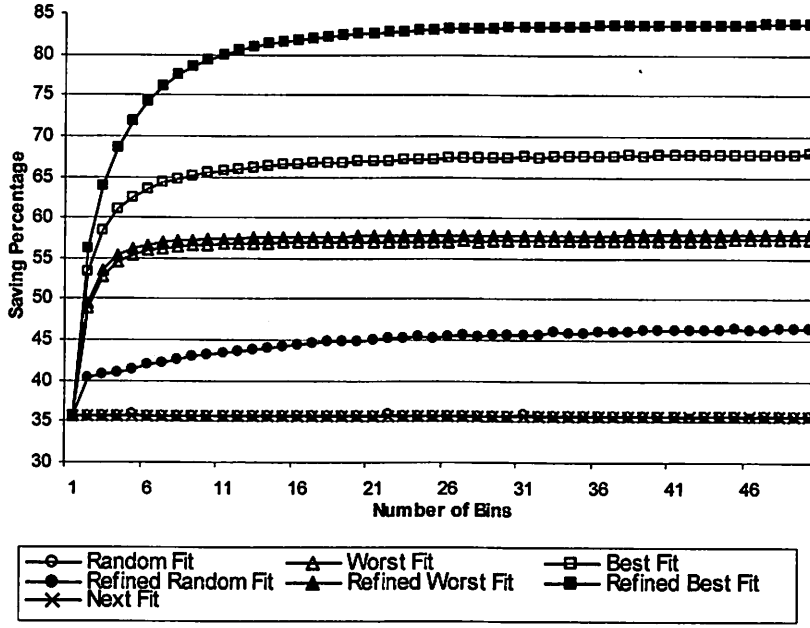

Figure 4(d): 70% threshold

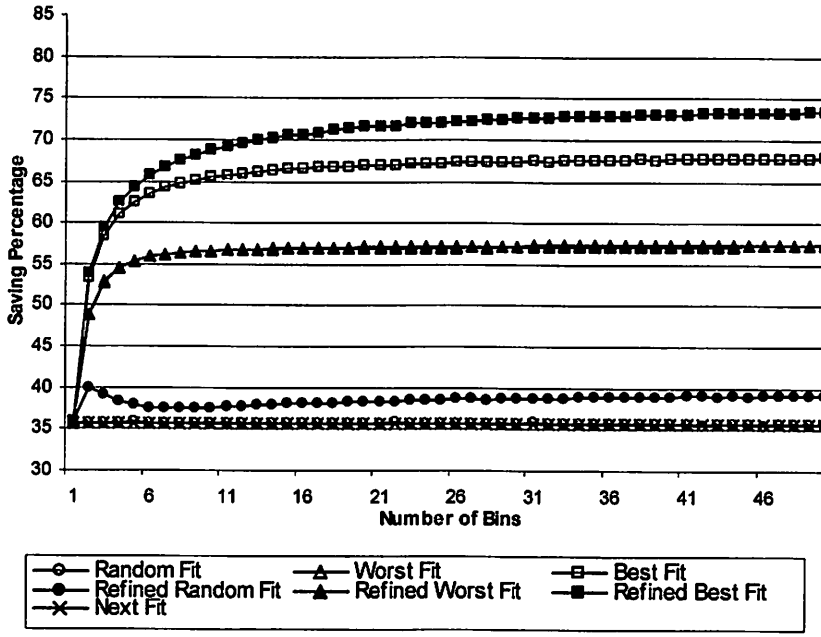

Figure 4(e): 90% threshold

After studying the performances of these seven on-line algorithms from the graphs, we can draw the following conclusions.

- The Next Fit lines and Random Fit lines overlap in all five graphs. In fact, the result of Next Fit will be the same as that of Random Fit if the incoming pieces are redistributed. Therefore, if the distribution of incoming pieces is random enough and the run is long, the performances of the two algorithms will be similar as reflected in our experiments.
- Generally, Best Fit outperforms Worst Fit and the performance of Worst Fit is higher than Next/Random Fit.
- The refined algorithms perform differently with different threshold. Refined Worst Fit has a lower performance than the original Worst Fit algorithm most of the time. Refined Best Fit and Refined Random Fit perform poorer than the original Best Fit and Random Fit algorithms when the threshold is small and they outperform the original algorithms when the threshold is increased.
- Refined Best Fit with 70% threshold does the best among all algorithms. It can consistently produce a high saving percentage (over 75%) using only a small number of bins.

As we can observe from the above results, the quality of refined algorithms depends highly on the value of satisfaction threshold. In order to have a deeper understanding of the relationship between them, the experiments were repeated with the number of bins fixed to 10, 30 and 50 respectively. The saving percentage is plotted against the thresholds of the three refined algorithms, which were allowed to vary from 1% to 99%. The performances of the original algorithms (Random Fit, Worst Fit and Best Fit) are also added for reference. The graphs are shown in Figure 5.

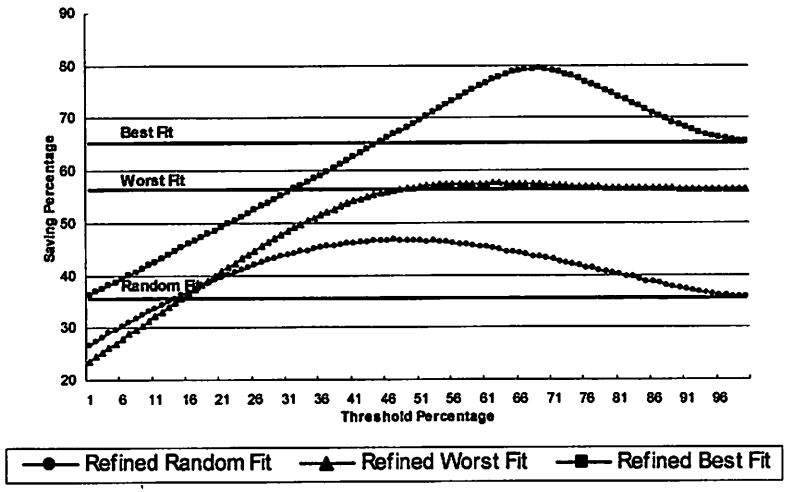

Figure 5(a): 10 bins

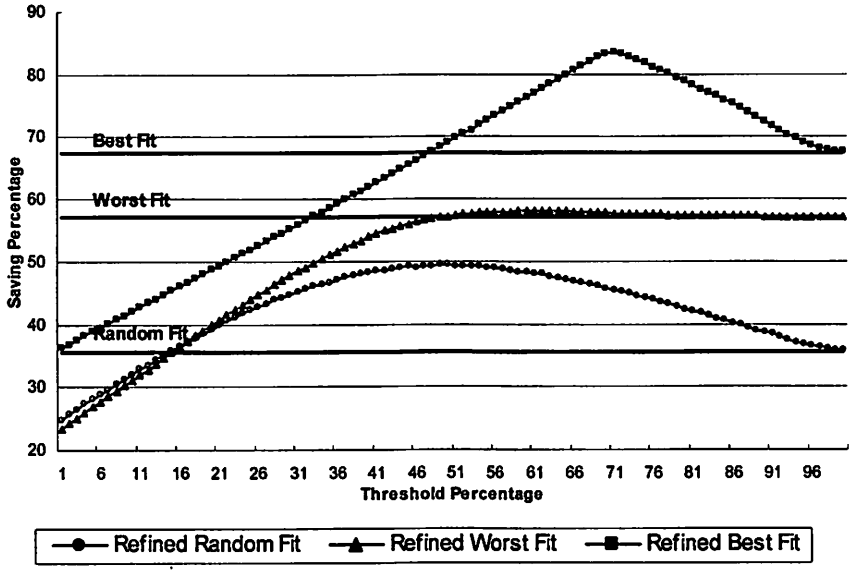

Figure 5(b): 30 bins

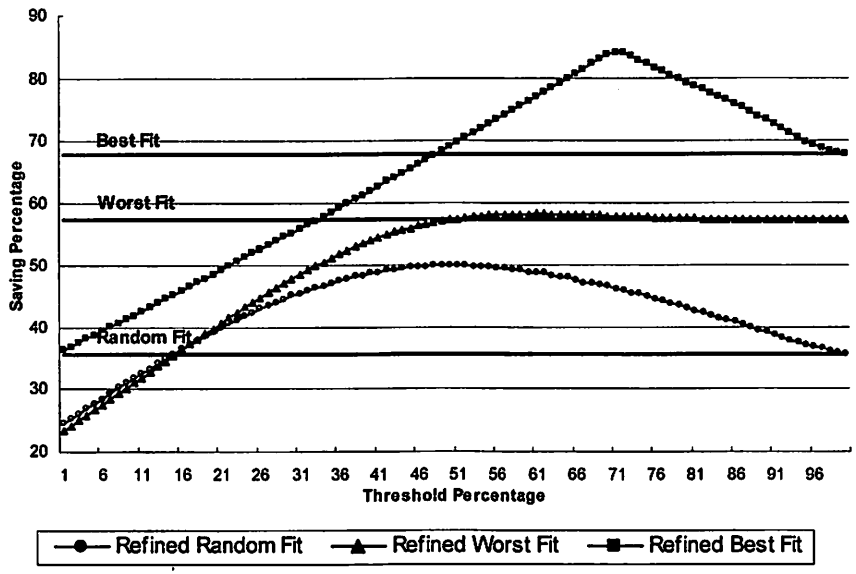

Figure 5(c): 50 bins

From the results, we can summarize that:

- With a probably set threshold, Refined Best Fit outperforms all other algorithms. It produces a 79% saving with 10 bins and the threshold set to 69%; a 83% saving with 30 bins and 71% threshold; a 84% saving with 50 bins and 72% threshold.
- All three refined algorithms generally show inverted U-shape lines with varying thresholds. They all have poor performance when the threshold is small. With a small threshold (below 20%), the algorithms will probably close the bins too early and usually a small saving percentage will be resulted. After that, the performance of each algorithm consistently rises when the threshold increases, reaches its own maximum and falls back to the value of the original algorithms when the threshold approaches 100%.
- When comparing the graphs in Figures 5(a), 5(b) and 5(c), we can observe that the optimum thresholds and saving percentages increase when the number of bins increases. A larger number of bins means more buffers for the algorithms to hold the pieces before it is forced to close a bin. It increases the chance to wait for a large piece and pack it as the last piece of a bin to earn a high saving percentage.

In terms of complexity, Next Fit and Random Fit run in $O(m)$. Worst Fit, Best Fit, Refined Random Fit, Refined Worst Fit and Refined Best Fit all run in $O(m + k \log k)$ where m is the number of pieces and k is the number of available bins. In the next section, we will study how these seven on-line sub-optimal algorithms perform in a real case.

5. Case Study - Stored-Value Tickets for KCR (Hong Kong)

We will employ our algorithms in a real case which tries to optimize the usage of stored-value tickets of KCR in Hong Kong. There are 3 categories of travelers: Adult, Senior Citizen & Child, and Student. For our study, we use the travel fare for adults only. For adults below 65, there are three kinds of tickets with value $50, $100 ($3 bonus) and $200 ($12 bonus). Domestic Travel Fare table for Adult is as follows:

KN	1.5	1.5	2.5	2.5	2.5	3	3	3	3.5	3.5
2.8	MK	1.5	2.5	2.5	2.5	3	3	3	3.5	3.5
2.8	2.8	KT	2.5	2.5	2.5	3	3	3	3.5	3.5
4.4	4.4	4.4	TW	1.5	1.5	1.5	2.5	2.5	2.5	2.5
4.4	4.4	4.4	2.8	ST	1.5	1.5	2.5	2.5	2.5	2.5
5	5	5	2.8	2.8	FT	1.5	2	2	2.5	2.5
5.6	5.6	5.6	3	3	2.8	Univ	1.5	1.5	2.5	2.5
6.3	6.3	6.3	4.6	4.6	4.1	3	TP	1.5	1.5	1.5
6.3	6.3	6.3	4.6	4.6	4.1	3	2.8	TW	1.5	1.5
7.3	7.3	7.3	5.1	5.1	4.9	4.6	3	3	FN	1.5
7.3	7.3	7.3	5.1	5.1	4.9	4.6	3	3	2.8	SS

The average percentage is plotted against the number of tickets and the thresholds of the refined algorithms are set to 10% and 30%. The 50%, 70% and 90% graphs are omitted as they are similar to the 30% graph. From the graphs, we can find that 10% Refined Best Fit gains a saving percentage of more than 12%. However, when 30% threshold is used, the refined algorithms revert back to the original ones. It implies that 30% threshold is too high and the optimum thresholds for the refined algorithms should most likely fall between 10% and 30%.

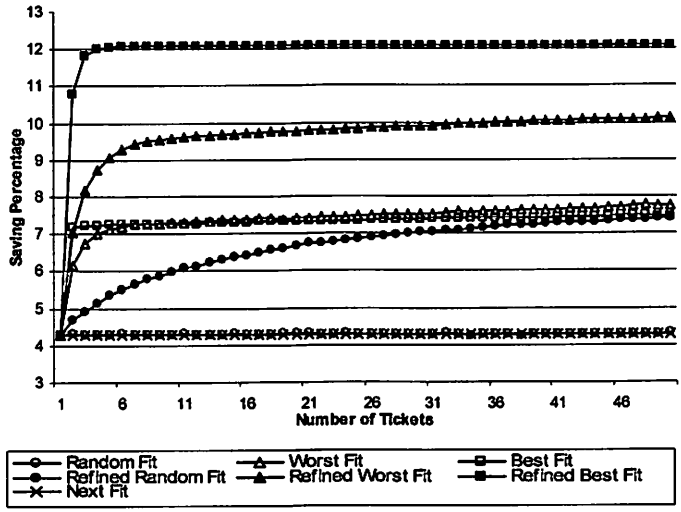

Figure 6(a): 10% threshold

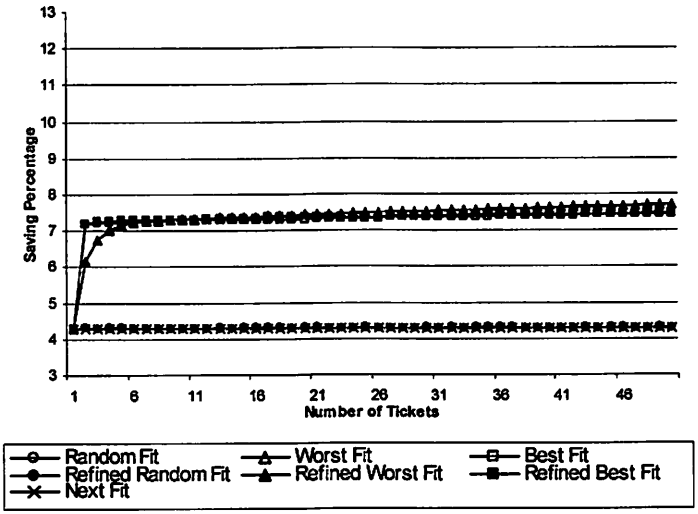

Figure 6(b): 30% threshold

Next, the saving percentage is plotted against the threshold of the three refined algorithms using 5, 10, 30 and 50 tickets (Figure 7). The maximum saving occurs when using Refined Best Fit with 11% threshold and 50 tickets. However, as it is quite inconvenient to carry such a large number of tickets in daily life and the difference between using 5 tickets and 50 tickets is not so large, we suggest the KCR users should apply the Refined Best Fit algorithm with 11% threshold and 5 tickets as a convenient and economical strategy.

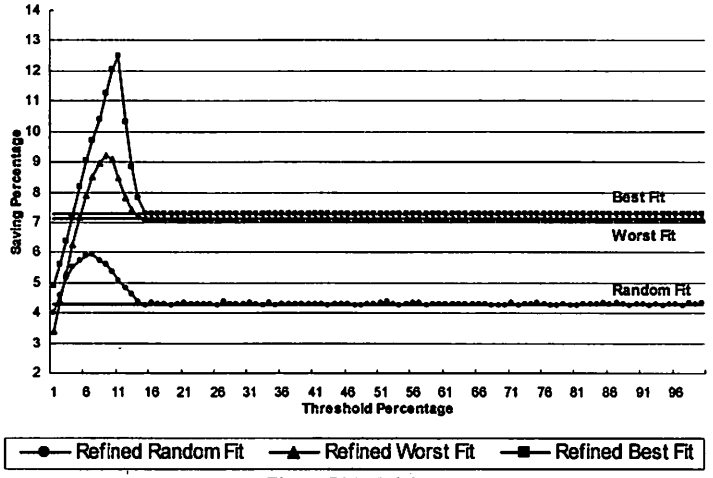

Figure 7(a): 5 tickets

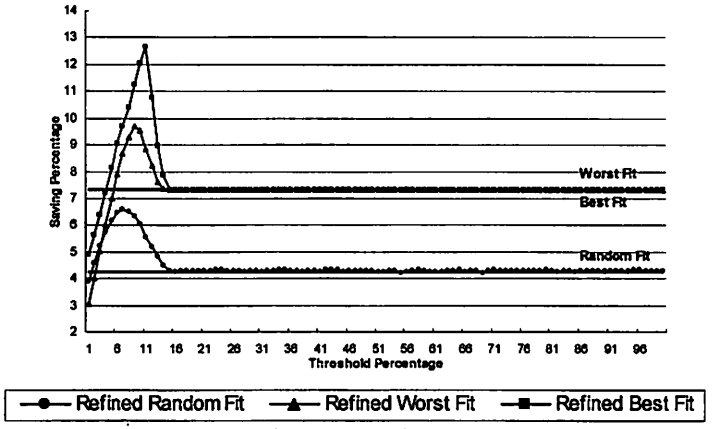

Figure 7(b): 10 tickets

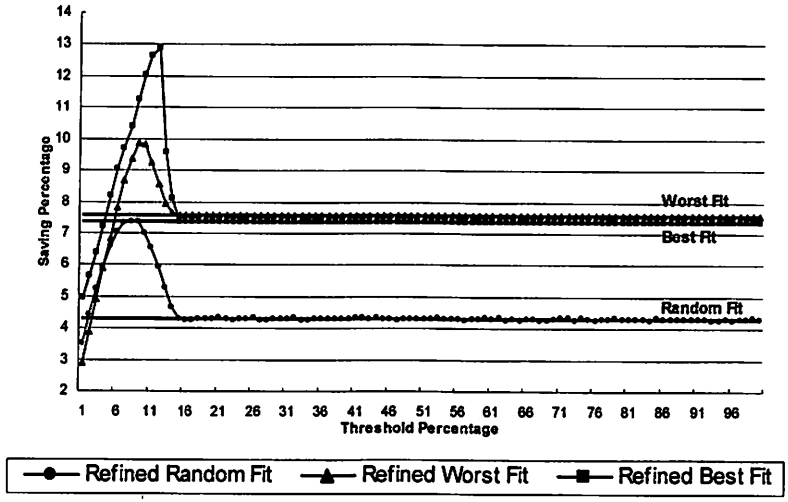

Figure 7(c): 30 tickets

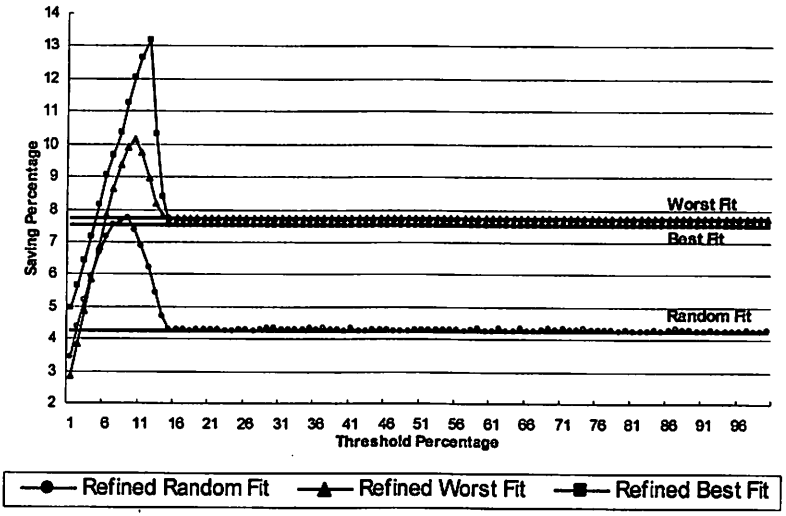

Figure 7 (c) 30 tickets

6. Conclusions

In this paper we have proposed a variant of the classical bin packing problem called open-end bin packing problem. It opens up a challenge on the computability of a number of interesting optimization problems. We showed that optimal on-line algorithm for this problem does not exist. Seven

sub-optimal on-line algorithms, Next Fit, Random Fit, Worst Fit, Best Fit, Refined Random Fit, Refined Worst Fit and Refined Best Fit were proposed, and their performances were compared in Section 4.

It was found that Refined Best Fit has the best overall performance provided that it has the appropriate satisfaction threshold. An application of the scope of study in this paper was discussed in a case study presented in Section 5.

Under the roof of the concept of having open end(s) for a bin, there are numerous interesting optimization problems on open-end bin packing for intellectual challenge, but those problems with practical applications should be on the top of the list. For future research, two possible directions are investigation on multi-dimensional open-end bin packing and variants of the proposed open-end bin packing problems such as requiring a minimal fraction of the last piece in each bin.

For problems directly related to the studied on-line open-end bin packing problem in this paper, it is interesting to derive better sub-optimal on-line algorithms and investigate how much better they perform.

Bibliography

[1] Assmann, S. J., Johnson, D. S., Kleitman, D. J., and Leung, J. Y-T.: On a Dual Version of the One-Dimensional Bin Packing Problem. J. of Algorithms **5** (1984) 502-525.

[2] Brown, D. J.: A Lower Bound for On-line One-dimensional Bin Packing Algorithm. Tech. Rep. No. R-864, Coordinated Sci. Lab., Univ. of Illinois, Ill. (1979).

[3] Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S.: An Application of Bin Packing to Multiprocessor Scheduling. SIAM J. Comput. **7** (1978) 1-17.

[4] Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S.: Approximation Algorithms for Bin Packing - An Updated Survey. Algorithm Design for Computer System Design. G. Ausiello, M. Lucertini and P. Serafini (Eds.), Springer-Verlag, New York, (1984) 49-106.

[5] Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S.: Bin Packing with Divisible Item Sizes. J. of Complexity **3** (1987) 406-428.

[6] Coffman, Jr., E. G., and Leung, J. Y-T.: Combinational Analysis of an Efficient Algorithm for Processor and Storage Allocation. SIAM J. Comput. **8** (1979) 202-217.

[7] Coffman, Jr., E. G., and Leung, J. Y-T., and Ting, D. W.: Bin Packing: Maximizing the Number of Pieces Packed. Acta Informatica **9** (1978) 263-271.

[8] Deuermeyer, B. L., Friesen, D. K., and Langston, M. A.: Maximizing the Minimum Processor Finish Time in a Multiprocessor System. SIAM J. Alg. Meth. (1982) 190-196.

[9] Garey, M. R. & Johnson, D. S.: Computer and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA, (1979).

[10] Hochbaum, D. S., and Shmoys, D. B.: Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results. J. ACM **34** (1987) 144-162.

[11] Hu, T. C.: Combinatorial Algorithms, Addison-Wesley Publication Co. (1982).

[12] Johnson, D. S.: Near-Optimal Bin Packing Algorithm, Doctoral Thesis, MIT, Cambridge, Mass. (1973).

[13] Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., and Graham, R. L.: Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. SIAM J. Comput. **3** (1974) 299-325.

[14] Karlin, Samuel & Talyor, Howard M.: A First Course in Stocastic Processes. 2nd Ed., Academic Press, Inc. (1975).

[15] Langston, M. A.: Processor Scheduling with Improved Heuristic Algorithms, Ph.D. Dissertation, Texas A&M Univ., College Station, TX, (1981).

[16] Lee, C. C., and Lee, D. T.: A Simple On-Line Bin Packing Algorithm. Journal of ACM, **32(3)** (1985) 562-572.

[17] Liang, F. M.: A Lower Bound for On-line Bin Packing Algorithm. Submitted for publication.

[18] Yao, A. C.: New Algorithms for Bin Packing, Journal of ACM, **27(2)** (1980) 207-227.

[19] Young, G. H., and Lau, S.: Foundation of Multi-dimensional Open-end Bin Packing, submitted for publication.

[20] Leung J. Y., Dror, M., and Young, G. H.: A Note on an Open-end Bin Packing problem, Journal of Scheduling, to appear.