Yokohama Publishers
ISSN 1349-8169 ONLINE JOURNAL

Pacific Journal of Optimization
Vol. 3, No. 3, pp. 511–527, September 2007

PERFORMANCE ESTIMATIONS OF FIRST FIT ALGORITHM FOR ONLINE BIN PACKING WITH VARIABLE BIN SIZES AND LIB CONSTRAINTS

J.Y. Lin, P. Manyem and R.L. Sheu

Abstract: We consider the NP Hard problem of online Bin Packing while requiring that larger (or longer) items be placed below smaller (or shorter) items — we call such a version the LIB version of problems. Bin sizes can be uniform or variable. We provide analytical upper bounds as well as experimental results on the asymptotic approximation ratio for the first fit algorithm.

Key words: *online approximation algorithm, asymptotic worst case ratio, bin packing problem, longest item, uniform sized bins, variable sized bins*

Mathematics Subject Classification: *68W25, 68Q17, 90B05, 90C27*

1 Background

In the classical one-dimensional Bin Packing problem, we are given a list $L = (i : 1 \leq i \leq n)$ of items. The *size* of item i is a_i, where each $a_i \in (0, 1]$. The problem is to pack these n items into bins of size one such that the number of bins used is minimized. A bin is said to be *used* if it contains at least one item. A feasible solution is one where the sum of the sizes of the items in each used bin is at most equal to the bin size.

VSBP (Variable Sized Bin Packing Problem) is similar to the classical problem stated above, except that the bin sizes can be different — we are given a collection $\mathcal{B}$ of distinct bin sizes s_1 through s_K, and s_K is the largest (or just *longest*, in the one-dimensional case) bin size with $s_K = 1$. Size s_1 is the smallest. The objective is to minimize the sum of the sizes of the bins used. The dual of Bin Packing is *Bin Covering*, where the item sizes in a bin should total up to *at least* the bin size.

Bin Packing can be offline or online. If the sizes of all items are known in advance, this is referred to as *offline* bin packing. In the *online* version of Bin Packing, items in L arrive one by one. When an item i of length a_i arrives, it must immediately be assigned to a bin and this assignment cannot be changed later. There are two types of online problems. In a *completely online* case, the size of item i is known ONLY AFTER item $(i-1)$ has been placed in a bin. The other type, however, may be called as *semi online* in which *all* item sizes are known in advance just like the offline version. Moreover, the instance is associated with a certain order and items arrive one by one following this given order. In this paper, we refer to an online problem as the second type and an explicit definition will be stated in Section 2.

In all versions of Bin Packing, it is assumed that there is an infinite supply of bins of each size. Hence, running out of bins to place items is never an issue.

LIB version of Bin Packing. The bin packing problem considered in this paper is an online version with variable bin sizes and imposes this additional requirement: In any bin, for any pair of items i and j, if $\text{size}(j) = a_j > \text{size}(i) = a_i$, then j should be placed in the bin *below* i. In other words, *longer* items should be placed lower in any bin than *shorter* items. We call this the LIB version, for *Longest Item at the Bottom*. Moreover, we assume that the length of each item in L cannot be arbitrarily small. Namely, there is a $0 < \gamma \leq s_1$ such that $a_i \in [\gamma, 1]$ for all $i \in L$. This assumption makes perfect sense in a practical application.

Literature Review. The exact problem studied in this paper has not been considered before, hence we review results for online Bin Packing problems that are slight variations of ours. Table 1 summarizes the results known so far in online Bin Packing. The numbers in square brackets refer to the bibliography.

For online Bin Packing with uniform bin sizes and the LIB constraint, Manyem [7] provided an algorithm based on First Fit with a guaranteed (worst case) AAR of 3. (See equation (3.1) for the definition of AAR, the asymptotic approximation ratio.) For the same problem, using adversary arguments, Finlay and Manyem [5] proved that no algorithm can guarantee an AAR less than 1.76. Again for the same problem, in [10], Manyem et al. construct counter-examples to show that the guaranteed AAR's of the well-known First Fit (FF), Best Fit (BF) and Harmonic Fit (HF) algorithms are at least two.

For online Bin Packing with variable bin sizes without the LIB constraint, Csirik [4] provided an algorithm based on Harmonic Fit that guarantees an asymptotic approximation ratio (AAR) of at most 1.7. In the same paper, with just two bin sizes, Csirik was able to cut down the ratio to 1.4. Later, the upper bound 1.7 was further reduced to 1.63597 by Seiden et al. [12] with a refined version of the harmonic algorithm. Interestingly, they also gave the first ever lower bound 1.335 for variable-sized online bin packing problem when there are only two bin sizes. The authors claimed that their techniques will be applicable to a more general case.

Variable sized bins. Our paper deals with problem instances of variable bin sizes and the LIB constraint. We derive an upper bound "B_{FF}" for Variable Sized Bin Packing with LIB in Theorem 3.11 of this paper. The bound is a function of four parameters (see Theorem 3.11). This is a generalisation of an earlier result by Manyem et al [9] where all available bin sizes are restricted to be multiples of the smallest bin size.

Recently, Xing and Chen [15] studied a new variant of uniform sized bin packing problem called the A-shaped bin packing problem. Each item is specified by a pair of numbers: height and radius. In packing, the sum of the heights of items in a bin can not exceed one and an item with a larger radius can not be placed on top of another item having a smaller radius. The uniform sized bin packing problem with the LIB constraint can be viewed as a special case of the A-shaped problem by considering the height as the same as the radius. Xing and Chen in [15] proposed a radius classifying algorithm which places two items into the same bin only if they have the same radius, forcing the LIB constraint to be satisfied. By the same idea as in [6], they prove that the radius classifying algorithm combined with First Fit (RCFF) has the following inequality

$$RCFF(L) \leq 1.7L^* + 2T \tag{1.1}$$

where $RCFF(L)$ is the total number of bins used by the algorithm, L^* is the optimal number of bins necessarily to pack the list of items, and T is the number of distinct radii in the list. This bound is not applicable to our problem since we deal with variable bin sizes and the list L may have an infinite T.

Basic Problem	LIB ?	Bin Sizes	Upper Bound	Lower Bound
Bin Packing	No	Uniform	1.59 [11]	1.53 [13]
	No	Variable	1.63597 [12]	1.33561[12]
	Yes	Uniform	3.0 [8]	1.76 [5]
	Yes	Variable	B_{FF} [Theorem 3.11]	1.76 [5]

Table 1: Bounds on Approximation Ratios in online Bin Packing and Covering with LIB

Organisation of this Paper. We provide a version of First Fit (FF) heuristic in Section 2 and then prove an upper bound on the guaranteed AAR (Asymptotic Approximation Ratio) in Section 3. The experimental results are in Section 4. The results here are more general than the ones mentioned in [9], where the bin sizes are multiples of the smallest bin size.

1.1 Applications

Bin Packing and Covering theory does help to solve practical industry based problems such as assigning semiconductor wafer lots to customer orders [2]. Another interesting application arises during assigning tasks to computer processors based on a task priority. Each bin is analogous to a processor. The size of a bin corresponds to the processor's capabilities (such as speed), and the position of a task in a bin corresponds to its priority.

The LIB version of Bin Packing has applications in the Transportation industry*, especially with loading of pallets in a truck. If long items are placed at the bottom of a pallet inside a truck, transportation is easier. In terms of weight, if heavier items are placed at the bottom, better stability of the truck can be achieved, and smaller items will not get crushed by larger items.

The dual of Bin Packing is *Bin Covering*, where the item sizes in a bin should total up to *at least* the bin size. Bin Covering has been applied in the industry, from packing peaches into cans in an "online" manner (so that the weight of each can is at least equal to its advertised weight) to breaking up a large company into smaller companies such that each new company is viable [14].

2 Problem and Algorithm

Problem Statement: Online LIB Variable-Sized Bin Packing (OLIBP). *Given an infinite supply of variable sized bins, and n items ordered from 1 up to n, each item with size in $[\gamma, 1]$ and $0 < \gamma < 1$. Each item arrives following the given order and, upon the arrival, should be placed in a bin assigned to it (on top of items previously placed in that bin). This placement cannot be changed later. In placing the items, the following LIB condition (2.1) and online condition (2.2) given below should be obeyed. For any used bin:*

$$[i \text{ is below } j \text{ in a used bin}] \implies [a_i \geq a_j], \tag{2.1}$$

and in a used bin, if item i is below item j, then i should have arrived prior to j in the input list L, that is,

$$[i \text{ is below } j \text{ in a used bin}] \implies [i < j]. \tag{2.2}$$

*Of course, to apply this research in a practical context, one needs to obtain algorithms for three-dimensional bin packing. But before embarking on an analysis of the three-dimensional case, we should analyse the simpler (one-dimensional) problem first.

a_i	Size of item i
b_j	Bin number j
B	Set of used bins
$\mathcal{B}$	Set of available bin sizes, s_1 through s_K
i	Index for an item (usually)
K	Number of available bin sizes (cardinality of $\mathcal{B}$)
L	Input list of items, in a given sequence
N (or n)	Cardinality of L (usually)
p	percentage of ones (used in experimental studies)
R_{ALG}	Worst case asymptotic approximation ratio for algorithm ALG
$s_1(s_K)$	Size of the smallest (largest) available bin size
$topSize(b_j)$	Size of the item at the top of bin b_j
$totalSize(b_j)$	Sum of the sizes of the items in bin b_j

Table 2: Notation (in alphabetical order)

LIB	Largest (Longest, in the one-dimensional case) Item at the Bottom
FF	First Fit heuristic
AAR	asymptotic approximation ratio
SU	Space Utilization factor (in a bin, or set of bins)
VSBP	Variable Sized Bin Packing
OLIBP	Online and LIB version of VSBP

Table 3: Acronyms

A feasible solution is one where the sum of the item sizes in each used bin is at most equal to the bin size. The available bin sizes consist of a finite set $\mathcal{B} = \{s_j : 1 \leq j \leq K\}$, $s_j < s_{j+1}$, $1 \leq j \leq K-1$. *The bin sizes are normalized, that is,* s_K *(the largest bin size) is equal to one. The smallest bin size* s_1 *is greater than zero. The goal is to find a feasible solution that minimizes the sum of the size of used bins.*

2.1 NP-hardness

Once we replace the online condition with the online constraint (2.2), it can be shown that the online (optimization) version is NP-hard. We shall do this by showing that the decision version is NP-complete. If a decision problem is in NP, then the corresponding optimization version is said to be an *NPO* problem [1], where NPO stands for *NP Optimization*.

Proof that the decision version is in NP. It is simple to show this fact. Given a solution S (the one guessed by a non-deterministic Turing machine), it can be checked within polynomial time in n, whether S is feasible for the given instance — this is just an exercise in checking the LIB (2.1) and online (2.2) constraints for all items in the used bins. There are n used bins at the most, each of which contains at most n items. In addition, we should ensure that each item has been placed in a bin. Feasibility also involves checking whether the number of used bins is at most a given value K. Thus the problem is in NP (and hence the optimization problem, OLIBP, is in NPO).

Reduction. The offline Bin Packing problem (also Problem P_1 below), known to be NP-complete [3], can be reduced to the online version described above. Let us say that (in short)

P_1: *offline version without the LIB constraint* (uniform bin sizes),

P_2: *online version whose item sizes are non-increasing* (uniform bin sizes),
P_3: *online version with LIB* (uniform bin sizes), and
P_4: *online version with LIB* (variable bin sizes).

We shall defer the specific definition for each problem (especially P_2) to the reduction proof.

As in NP-completeness reductions, all problems considered are decision versions. Moreover, in reductions from P_i to P_{i+1}, $1 \leq i \leq 2$, we will use the same value of k for the source problem P_i and the target problem P_{i+1}, where k appears in the decision query, "Is there a feasible solution that uses at most k bins?". In the P_3 to P_4 reduction, we will use the same k in P_3 and P_4, however, the query in P_4 changes to, "Is there a feasible solution where the sum of the sizes of the used bins is at most k ?".

We can do a step-by-step reduction from P_1 to P_4 as follows.

Reduction from P_1 to P_2: Let I_2 be an instance of P_2. Then the item sizes in I_2 are non-increasing, in the sense that $a_i \geq a_j$ for $i < j$ where $i < j$ is defined by the given online order. In other words, for an instance of P_2, heavier (longer) items always arrive earlier. Consequently, a given instance I_1 of P_1 can be transformed to an instance I_2 of P_2, simply by sorting the items in I_1 in the order of non-increasing sizes.

We should show that a feasible solution for I_1 can be converted to a feasible solution for I_2 and vice-versa. Clearly, a feasible solution S_2 to I_2 using k_2 bins (where $k_2 \leq k$) is also a feasible solution to I_1. In the other direction, a feasible solution S_1 to I_1 using k_1 bins ($k_1 \leq k$) can be transformed to a feasible solution for I_2, by rearranging items in each used bin of S_1, so that the Lib constraint (2.1), and therefore the online condition (2.2) is satisfied.

In both proof directions, the number of bins used is unchanged when we transform the solution for one problem to a solution for the other.

Hence the reduction from P_1 to P_2, which proves that P_2 is NP-complete.

Reduction from P_2 to P_3: Consider an instance I_2 of P_2. Since both problems are online version, I_2 is also an instance for P_3 (but call it I_3). In addition, the item sizes in I_2 are non-increasing, in which case the online and LIB constraints play exactly the same role, and have exactly the same effect. Therefore, a solution S_3 to I_3 satisfying both the online and the LIB constraint is surely a solution to I_2. Conversely, a solution S_2 to I_2 satisfying only the online constraint will, simultaneously, meet the LIB constraint. Finally, S_2 and S_3 has the same objective value (counting the number of used bins).

Hence the reduction from P_2 to P_3, and thus P_3 is NP-complete.

Reduction from P_3 to P_4: This is straightforward. P_3 is just a special case of P_4. Given an instance I_3 of P_3, use only unit-sized bins in the instance I_4 of P_4 — the input list for both instances is exactly the same. The objective function in I_3, which counts the number of unit-sized bins, is clearly equal to the objective function in I_4, which computes the sum of the sizes of (unit-sized) used bins. It follows that P_4 is NP-complete (and hence the optimization version of P_4, which is OLIBP, is NP-hard).

The above argument also applies to optimal solutions for I_1 and I_2. One might argue that since I_1 has more "freedom" in the placement of items, and that the solution space of I_2 is a subset of the solution space of I_1, an optimal solution S_1^* for I_1 should use fewer bins (say s_1) than an optimal solution S_2^* for I_2 (let us say, s_2 number of bins) — that is, it can be argued that $s_1 < s_2$. However, this is untrue, because items in each bin of S_1^* can be sorted in the order of item sizes to provide a feasible solution $\bar{S}_2^*$ for I_2. Now, $\bar{S}_2^*$ uses fewer bins than S_2^* (s_1 instead of s_2), contradicting our assumption that S_2^* is an optimal solution to I_2.

2.2 First Fit

The First Fit algorithm can be modified to accommodate OLIBP. (See [8] or [7] or [3] for descriptions of First Fit.) The behaviour of FF is summarized as follows: When an item i arrives, assume that bins b_1 through b_m have already been *used*, in that order. Each such bin b_j, $1 \leq j \leq m$, has two parameters, $topSize(b_j)$ and $totalSize(b_j)$, representing the size of the topmost item in b_j and the sum of the item sizes in b_j respectively. FF scans b_1 through b_m in that order. For each such bin b_j, it checks if (1) $a_i \leq topSize(b_j)$, and (2) $a_i \leq size(b_j) - totalSize(b_j)$. FF places item i in the first such bin b_j that satisfies both these conditions and updates $topSize(b_j)$ as well as $totalSize(b_j)$. If no such bin among b_1 through b_m satisfies these conditions, FF opens a new bin b_{m+1} of size

$$\lceil a_i \rceil_{\mathcal{B}} = min\{s_j | s_j \geq a_j, s_j \in \mathcal{B}\} \tag{2.3}$$

to place i. For instance, if $\mathcal{B} = \{0.2, 0.4, 0.6, 0.8, 1.0\}$, an arriving item of size 0.64 will be placed in a bin of size $\lceil .64 \rceil_{\mathcal{B}} = 0.8$, not in a bin with a size of one.

Algorithm (ALG). First Fit (online variable-sized LIB Bin Packing).
Given: Items $1 \cdots N$ with sizes $a_1 \cdots a_N$, $\gamma \leq a_i \leq 1$ for $1 \leq i \leq N$,
bin sizes $s_1 \cdots s_K$, $0 < s_1 < s_2 < \ \cdots \ < s_{K-1} < s_K = 1$.
Running Time: $O(N(K+N))$.

```
1 nBin (number of bins used) = 0;
2 for (item = 1 to N) do
3     placed[item] = NO;
4     bin = 1;
5     While (bin ≤ nBin AND placed[item] == NO) do
6           X = (topSize[bin] ≥ size[item]);
7           Y = (size[bin]-totalSize[bin] ≥ size[item]);
8           if (X == true AND Y == true) then
9               place item in bin;
10              update topSize[bin] and totalSize[bin];
11              placed[item] = YES;
12          end if
13          bin = bin + 1;
14    end While
15    if (placed[item] == NO) then (item not placed in any previous bin)
16          nBin = nBin + 1; (new, fresh, unused bin)
17          size[nBin] = s1;
18          While (size[nBin] < size[item])
19              increase size[nBin] to next higher bin size available;
20          place item in nBin;
21          topSize[nBin] = size[item];
22          totalSize[nBin] = size[item];
23          placed[item] = YES;
24    end if
25 end for
```

The key difference between original FF algorithm (where only bins of unit-size are used) and ALG is that, in case an unused bin is needed, ALG searches (in lines 18-19) the set of bin sizes $\{s_1, s_2, \cdots, s_K\}$ for the *best fitting bin* for the item to be placed.

3 Proof of A Bound of AAR

In this subsection, we give estimations of the asymptotic approximation ratio (AAR) for ALG. Let $ALG(L)$ denote the sum of the sizes of bins generated online by ALG to pack L. Let $OPT(L)$ be the optimal value of bin sizes necessary for packing items in L. The AAR is defined by

$$R_{ALG} = \lim_{s\to\infty} \sup_{L}\{\frac{ALG(L)}{OPT(L)}|OPT(L) > s\}. \tag{3.1}$$

Define the $SU(B)$ (*Space Utilization factor for a set of used bins* $B = \{b_1, b_2, \cdots b_m\}$) by the First Fit algorithm as follows:

$$SU(B) = \frac{\sum_{b_j\in B}\sum_{i\in b_j} a_i}{\sum_{b_j\in B} \text{ size}(b_j)} = \frac{\sum_{b_j\in B} totalSize(b_j)}{\sum_{b_j\in B} \text{ size}(b_j).} \tag{3.2}$$

In other words, $SU(B)$ is the ratio of the space occupied by items in the bins B, to the sum of the sizes of the bins in B. If B consists of just one bin b_j, we will simply write $SU(b_j)$ as a shorthand for $SU(\{b_j\})$. The following observation follows immediately from the definition of $SU(B)$.

Lemma 3.1. *If $SU(b_i)$ of each used bin b_i, $i = 1, 2, \cdots, m$ is greater than or equal to δ, then R_{ALG} has an upper bound of $\frac{1}{\delta}$.*

Proof. Since $SU(b_j) \geq \delta$, the total item size in b_j should be larger than $\delta \times size(b_j)$. Hence

$$\sum_{b_j\in B} totalSize(b_j) \geq \delta \sum_{b_j\in B} size(b_j). \tag{3.3}$$

Any feasible packing, including the optimal one, must use bins whose total size is at least $\sum_{b_j\in B} totalSize(b_j)$. Therefore,

$$\lim_{s\to\infty} \sup\{\frac{R_{ALG}(L)}{OPT(L)}|OPT(L) > s\} \leq \lim_{s\to\infty} \sup\{\frac{\sum_{b_j\in B} size(b_j)}{\delta\sum_{b_j\in B} size(b_j)}|OPT(L) > s\} = \frac{1}{\delta}.$$

□

In the rest of this paper, we will choose δ to be

$$0 < \delta \leq \min_{1\leq i\leq K-1} \frac{s_i}{s_{i+1}}$$

and consider instances where there are some bins with the space utilization less than δ (so that Lemma 3.1 does not apply). Let

$$J = \max_{1\leq j\leq m} \{j|SU(b_j) < \delta\}.$$

In other words, b_J is the last bin in B for which $SU(b_j) < \delta$ is true. Let I be the bottom item of bin b_J and let $totalSize^I(b_j)$ be the sum of the sizes of items in bin b_j *when item I arrived*. Similarly, $topSize^I(b_j)$ is the size of the top item in bin b_j *when item I arrived*.

Lemma 3.2. *Let δ, I, J be defined as above. Then we have (1) $a_I \in (\gamma, s_1]$; (2) $size(b_J) = s_1$; and (3) $a_I < \delta s_1$.*

Proof. Clearly, a new bin b_J is opened when I arrives, since I has been placed at the bottom of b_J. (1) By ALG, if $a_I \in (s_i, s_{i+1}]$, $1 \leq i \leq K-1$, then $size(b_J) = s_{i+1}$ and $totalSize(b_J) \geq a_I > s_i$. Hence

$$SU(b_J) = \frac{totalSize(b_J)}{size(b_J)} > \frac{s_i}{s_{i+1}} \geq \delta,$$

which contradicts $SU(b_J) < \delta$. Therefore, $a_I \in (\gamma, s_1]$.

(2) Since $a_I \in (\gamma, s_1]$, $size(b_J) = s_1$.

(3) Since

$$\frac{a_I}{size(b_J)} \leq SU(b_J) < \delta,$$

we have $a_I < \delta s_1$. □

Lemma 3.3. *If $\gamma \geq \delta s_1$, $R_{ALG} \leq 1/\delta$.*

Proof. If $\gamma \geq \delta s_1$, then $a_I \geq \delta s_1$. By Lemma 3.2, the $SU(B)$ factor of every used bin is at least δ. By Lemma 3.1, R_{ALG} is bounded above by $1/\delta$. □

Let us continue with the notations defined above Lemma 3.2. Upon I's arrival, there are two reasons why I was placed in a new bin b_J, and not in any of the bins b_j $(1 \leq j \leq J-1)$ used earlier: either (i) $totalSize^I(b_j) + a_I > size(b_j)$, or (ii) $topSize^I(b_j) < a_I$. Since there could be some items that arrived after I and were placed in b_j, the following inequalities hold:

$$totalSize^I(b_j) \leq totalSize(b_j); \tag{3.4}$$

$$topSize^I(b_j) \geq topSize(b_j). \tag{3.5}$$

Now, partition $\{b_j | 1 \leq j \leq J-1\}$ into two disjoint sets $\mathcal{C}$ and $\mathcal{D}$ with the following definition:

Type-c bins: First, consider the set $\mathcal{C}$ of bins, with $|\mathcal{C}| = c$, and

$$\mathcal{C} = \{b_j | totalSize^I(b_j) + a_I > size(b_j), 1 \leq j \leq J-1\}.$$

Refer to $\mathcal{C}$ as *type-c* bins. Since $a_I < \delta s_1$, it follows that

$$totalSize^I(b_j) > size(b_j) - \delta s_1, \quad \forall b_j \in \mathcal{C}.$$

By inequality (3.4),

$$totalSize(b_j) > size(b_j) - \delta s_1, \quad \forall b_j \in \mathcal{C}. \tag{3.6}$$

Define $p = \sum_{b_j \in \mathcal{C}} size(b_j)$. Then,

$$OPT(L) \;\geq\; sum\ of\ item\ sizes \;\geq \sum_{b_j \in \mathcal{C}} totalSize(b_j) \;\geq\; \sum_{b_j \in \mathcal{C}} [size(b_j) - \delta s_1] \;=\; p - \delta c s_1. \tag{3.7}$$

On the other hand, for type-c bins, the solution returned by ALG has a value of $\sum_{b_j \in \mathcal{C}} size(b_j) = p$. The upper and lower bounds for p are:

$$cs_1 \leq p \leq c. \tag{3.8}$$

Lemma 3.4. *For* $\delta < \frac{1}{2}$, $\mathcal{C} \subseteq \{b_j | SU(b_j) \geq \delta, 1 \leq j \leq J-1\}$.

Proof. By inequality (3.6),

$$\frac{totalSize(b_j)}{size(b_j)} > 1 - \frac{\delta s_1}{size(b_j)}.$$

Since $size(b_j) \geq s_1$,

$$SU(b_j) = \frac{totalSize(b_j)}{size(b_j)} \geq 1 - \frac{\delta s_1}{s_1} = 1 - \delta > \delta.$$

□

Type-d bins: Secondly, let $\mathcal{D}$ be the subsets of bins, with $|\mathcal{D}| = d$ and

$$\mathcal{D} = \{b_j | topSize^I(b_j) < a_I, 1 \leq j \leq J-1\} \cap \bar{\mathcal{C}},$$

where $\bar{\mathcal{C}} = \{b_j | totalSize^I(b_j) + a_I \leq size(b_j), 1 \leq j \leq J-1\}$. Name these bins as *type-d* bins.

Among type-d bins, consider any two, say b_j and b_k with $j < k$, meaning that bin b_j was opened before bin b_k. Let $e^I(j)$ $[e^I(k)]$ be the topmost item of b_j $[b_k]$ when I arrived.

Lemma 3.5. *If* $b_j, b_k \in \mathcal{D}$ *with* $j < k$, *then (1)* $totalSize^I(b_j) + a_{e^I(k)} \leq size(b_j)$; *(2)* $a_{e^I(j)} < a_{e^I(k)} < a_I$.

Proof. By the definition of $\mathcal{D}$, $a_{e^I(k)} = topSize^I(b_k) < a_I$ for all $b_k \in \mathcal{D}$, and

$$totalSize^I(b_j) + a_I \leq size(b_j).$$

It follows that

$$totalSize^I(b_j) + a_{e^I(k)} < size(b_j).$$

In other words, there was enough space in b_j for item $e^I(k)$. If $e^I(k)$ had arrived after $e^I(j)$, then a placement of $e^I(k)$ over $e^I(j)$ would have been attempted and failed due to $a_{e^I(j)} < a_{e^I(k)}$. On the other hand, if $e^I(j)$ had arrived after $e^I(k)$, then, a placement of $e^I(k)$ over an earlier item $x < e^I(j)$ in b_j would have been attempted and failed, implying that $a_x < a_{e^I(k)}$. Since x is below $e^I(j)$ in b_j, $a_{e^I(j)} \leq a_x$. It follows that $a_{e^I(j)} \leq a_x < a_{e^I(k)}$. □

Lemma 3.6. *Let* $\mathcal{D} = \{b_{t_1}, b_{t_2}, \cdots, b_{t_d}\}$, *with* $t_1 < t_2 < \cdots < t_d$ — *thus among the* $\mathcal{D}$ *bins,* b_{t_1} *was opened the earliest and* b_{t_d} *the last. For any item* $\psi \in b_{t_k}$, $k = 2, 3, \cdots, d$, *if* $\psi < I$ *and* $a_\psi < a_I$, *then there is another item* $\tau \in b_{t_{k-1}}$ *such that* $\tau < \psi$ *and* $a_\tau < a_\psi$.

Proof. Since $b_{t_{k-1}} \in \mathcal{D}$, we have the following inequality:

$$totalSize^I(b_{t_{k-1}}) + a_I \leq size(b_{t_{k-1}}).$$

By assumption, ψ arrived before I. Therefore,

$$totalSize^\psi(b_{t_{k-1}}) \leq totalSize^I(b_{t_{k-1}}).$$

Moreover, $a_\psi < a_I$ implies that

$$totalSize^I(b_{t_{k-1}}) + a_\psi < size(b_{t_{k-1}}).$$

Then

$$totalSize^{\psi}(b_{t_{k-1}}) + a_{\psi} \leq size(b_{t_{k-1}}). \tag{3.9}$$

This implies that, when ψ arrives, there is at least one item τ already in $b_{t_{k-1}}$ so that $a_{\tau} < a_{\psi}$. Otherwise, by (3.9), ψ would have been placed in $b_{t_{k-1}}$. □

Let the topmost item of b_{t_d}, as I arrived, be α_d. Then $\alpha_d < I$ and $a_{\alpha_d} < a_I$. Apply Lemma 3.6 backward repeatedly, we obtain a sub-list of L such that $\alpha_1 < \alpha_2 < \cdots < \alpha_d < I$, $a_{\alpha_1} < a_{\alpha_1} < \cdots < a_{\alpha_d} < a_I$ with $\alpha_k \in b_{t_k}$, $k = 1, 2, \cdots, d$. Let $\Lambda = \{\alpha_1, \alpha_2, \cdots, \alpha_d\}$. According to the online and LIB constraints, every item in Λ must be placed in distinct bins and each item has a length at least γ. As a result, we have

Lemma 3.7. *The sum of the bin sizes optimal algorithm to pack type-d bins should be at least γd, whereas the value of the solution returned by ALG is $q = \sum_{b_j \in \mathcal{D}} size(b_j) \leq d$.*

Type-f bins: Beyond bin b_J, the last bin with space utility $SU(b_J) < \delta$, there could be several used bins all of which have $SU \geq \delta$. Name these bins as *type-f* and denote them by $\mathcal{F} = \{b_j \in B | j \geq J + 1\}$. Let the sum of their sizes be

$$f = \sum_{b_j \in \mathcal{F}} size(b_j), \tag{3.10}$$

which is the value returned by ALG. Again, by equation (3.3) in Lemma 3.1, the sum of item sizes in type-f bins is at least

$$\sum_{b_j \in \mathcal{F}} totalSize(b_j) \geq f\delta, \tag{3.11}$$

which will be used as a lower bound for packing items in type-f bins the optimal way.

Thus the entire set of bins used by ALG is made up of, in this order: (i) a mixture of type-c bins and type-d bins, (ii) bin b_J containing item I, and (iii) type-f bins. The lower bound estimations for the sum of item sizes in each category are: (i) $p - \delta c s_1$ for type-c bins (by (3.7)); (ii) γd for type-d bins (by Lemma 3.7); (iii) γ for bin b_J; (iv) $f\delta$ for type-f bins (by (3.11)). Therefore, the lower bound for the optimal bin sizes is $p - \delta c s_1 + d\gamma + \gamma + f\delta$, whereas the solution returned by ALG is $p + q + s_1 + f$ (q is defined in the statement of this lemma.) The asymptotic ratio AAR requires us to consider the ratio

$$\frac{ALG(L)}{OPT(L)} \leq \frac{p + q + s_1 + f}{p - \delta c s_1 + d\gamma + \gamma + f\delta} \tag{3.12}$$

for all large inputs L for which $OPT(L) > s$ and $s \to \infty$. Equivalently, one of the numbers p, q, f must tend to infinity in the limit. Observe that, by (3.8), $p \to \infty$ implies $c \to \infty$ and also by Lemma 3.7, $q \to \infty$ implies $d \to \infty$. In what follows, we shall write $(\cdot, \cdot, \cdots, \cdot) \to \infty$ to indicate at least one of the components tend to infinity. Taking the limit on both sides of (3.12) gives

$$R_{ALG} \leq \lim_{(p,c,q,d,f) \to \infty} \frac{p + q + s_1 + f}{p - \delta c s_1 + d\gamma + \gamma + f\delta}$$

$$= \lim_{(p,c,q,d,f) \to \infty} \frac{p + q + f}{p - \delta c s_1 + d\gamma + f\delta}$$

where s_1 in the numerator and γ in the denominator do not affect the limit in any case.

Lemma 3.8. *If* $c > 0$, $\dfrac{s_1(1-\delta)}{\delta} < 1$ *and* $\gamma < \delta$, *then*

$$\frac{p+q+f}{p-\delta c s_1 + d\gamma + f\delta} \le \frac{1+\frac{d}{c}(1-\frac{\gamma}{\delta})}{s_1(1-\delta)}.$$

Proof. Since $cs_1 \le p \le c$ and $q \le d$,

$$\begin{aligned}&\frac{p+q+f}{p-\delta c s_1 + d\gamma + f\delta}\\ &\le \frac{c+d+f}{cs_1 - \delta c s_1 + d\gamma + f\delta}\\ &= \frac{1}{\delta}\frac{c\delta + d(\delta-\gamma) + d\gamma + f\delta}{cs_1 - \delta c s_1 + d\gamma + f\delta}.\end{aligned}$$

Since $\delta > \gamma$ and $cs_1 > \delta cs_1$,

$$\frac{c\delta + d(\delta-\gamma) + d\gamma + f\delta}{cs_1 - \delta c s_1 + d\gamma + f\delta} \le \max\{1, \frac{c\delta + d(\delta-\gamma)}{cs_1 - \delta c s_1}\}.$$

Moreover, by assumption,

$$\frac{c\delta + d(\delta-\gamma)}{cs_1 - \delta c s_1} = \frac{c + d(1-\frac{\gamma}{\delta})}{c}\frac{\delta}{s_1(1-\delta)} > 1.$$

This implies that

$$\frac{p+q+f}{p-\delta c s_1 + d\gamma + f\delta} \le \frac{1}{\delta}\frac{c\delta + d\delta(1-\frac{\gamma}{\delta})}{cs_1 - \delta c s_1} = \frac{1+\frac{d}{c}(1-\frac{\gamma}{\delta})}{s_1(1-\delta)}.$$

□

Remark. We may choose $\delta = \min\limits_{1\le i\le K-1}\{\dfrac{s_i}{s_{i+1}}\}$, then the condition $s_1\dfrac{1-\delta}{\delta} < 1$ holds automatically. To see this, let $k \in [1, K-1]$ be the integer such that $\delta = \dfrac{s_k}{s_{k+1}}$. Then

$$s_1\frac{1-\delta}{\delta} = s_1\frac{1-(s_k/s_{k+1})}{(s_k/s_{k+1})} = s_1\frac{s_{k+1}-s_k}{s_k} = \frac{s_1}{s_k}(s_{k+1}-s_k) < 1.$$

Define $M = \lim\limits_{(c,d)\to\infty}\dfrac{d}{c}$, which might be infinite. Then, under the assumption of Lemma 3.8,

$$R_{ALG} \le \frac{1+M(1-\frac{\gamma}{\delta})}{s_1(1-\delta)}. \tag{3.13}$$

Lemma 3.9. *If* $c = 0, d > 0$ *and* $\gamma < \delta$, *then*

$$\frac{p+q+f}{p-\delta c s_1 + d\gamma + f\delta} \le \frac{1}{\gamma}.$$

Proof. Since $c = 0$ implies $p = 0$ and also $q \le d$ and $\gamma < \delta$,

$$\frac{p+q+f}{p-\delta c s_1 + d\gamma + f\delta} = \frac{q+f}{d\gamma + f\delta} \le \frac{1}{\delta}\frac{d\delta + f\delta}{d\gamma + f\delta} \le \frac{1}{\delta}\frac{d\delta}{d\gamma} = \frac{1}{\gamma}.$$

□

Lemma 3.10. *If $c > 0$, $d = 0$ and $\delta \leq \min\{\frac{1}{2}, \min_{1 \leq i \leq K-1}\{\frac{s_i}{s_{i+1}}\}\}$, then $R_{ALG} \leq \frac{1}{\delta}$.*

Proof. Since $d = 0$, there are no type-d bins and $\mathcal{C} = \{b_j \mid 1 \leq j \leq J-1\}$. By Lemma 3.4, $SU(\mathcal{C}) \geq \delta$. This makes type-c and type-f indistinguishable and the lower bound estimations (of the optimal solution value) for the item sizes should follow the same rationale: (i) $p\delta$ for type-c bins and (ii) $f\delta$ for type-f bins. Therefore,

$$R_{ALG} \leq \lim_{(p,f)\to\infty} \frac{p + s_1 + f}{p\delta + \gamma + f\delta} = \lim_{(p,f)\to\infty} \frac{p+f}{p\delta + f\delta} = \frac{1}{\delta}.$$

□

Remark. When $d = 0$, the upper bound in (3.13) reduces to $\frac{1}{s_1(1-\delta)}$. By the assumption in Lemma 3.8, $s_1(1-\delta) < \delta$. Hence, $\frac{1}{\delta}$ is a better bound than $\frac{1}{s_1(1-\delta)}$ for the case $d = 0$.

Remark. If $c = d = 0$, by Lemma 3.1, $R_{ALG} \leq \frac{1}{\delta}$.

Theorem 3.11. *The asymptotic approximation ratio obtained by ALG (the FF heuristic) for the online VSBP Problem with the LIB constraint is guaranteed to be at most*

$$\max\{\frac{1 + M(1 - \frac{\gamma}{\delta})}{s_1(1-\delta)}, \frac{1}{\delta}, \frac{1}{\gamma}\}. \tag{3.14}$$

If there are only type-c and type-f bins, then R_{ALG} is at most $\frac{1}{\delta}$. If there are only type-d and type-f bins, then R_{ALG} is at most $\frac{1}{\gamma}$. If there are only type-f bins, then R_{ALG} is at most $\frac{1}{\delta}$.

Theorem 3.11 is a theoretical upper bound for the FF heuristic for solving OLIBP. Although this bound could be infinite whenever M is, we show in the next section that, by over 80,000 instances, this is rarely the case in practice.

4 Experimental Studies in OLIBP

The purpose of this section is two-folded: (1) we carried out simulations to study the practical performance of ALG and (ii) to get some idea about how tight the bounds we derived in Theorem 3.11 could be. To this end, we need to know the optimal bin sizes which will be computed by a branch and bound (B&B) based algorithm (described below). In the numerical experiments, we first determine N (the number of items) and K (the number of bin sizes). Then, the sizes of both the items and the bins were created randomly by uniform distribution. For each pair of (N, K), 5000 sets of data were generated. There are totally 80,000 instances tested in this experiment. The most extensive example we computed, is to pack 1000 items (N=1000) into a collection of bins with seven distinct bin sizes ($K = 7$). However, the branch & bound algorithm is very time consuming, so we were able to run it only for few values of (N, K). See Table 4.

In the B&B approach here, a node t of the B&B tree represents a partial solution that packs items, in this order, from 1 to some $i \in L$. The children of t represent different ways to pack item $i+1$, either in used bins at node t or in a new bin. The lower bound was computed at each node and tested against the current upper bound. At the beginning of the B&B algorithm, we set the upper bound to be the value of the ALG solution. As soon as B&B found a complete solution whose value is smaller than the ALG value, the upper bound is changed from the ALG value, to the value of this complete solution. If the lower-bound was larger than the current upper bound at any node, all its sub-trees were pruned.

List Size (N)	Number of Bin Sizes (K)	Number of Runs	Ave. Ratio (vs. B&B)	Max. Ratio (vs. B&B)	Percentage of ones (vs. B&B)	Min. UB	Ave. UB	Max. UB
10	5	5000	1.045	1.491	0.309	1.095	4.540	100.000
10	7	5000	1.036	1.319	0.267	1.148	4.121	100.000
15	5	5000	1.052	1.389	0.144	1.113	4.778	100.000
15	7	5000	1.046	1.266	0.080	1.106	4.108	100.000
20	5	5000	NA	NA	NA	1.095	4.841	100.000
20	7	5000	NA	NA	NA	1.099	4.047	100.000
25	5	5000	NA	NA	NA	1.077	4.688	100.000
25	7	5000	NA	NA	NA	1.162	4.127	100.000
100	5	5000	NA	NA	NA	1.118	4.399	75.000
100	7	5000	NA	NA	NA	1.102	3.866	72.000
200	5	5000	NA	NA	NA	1.095	4.738	84.000
200	7	5000	NA	NA	NA	1.104	3.903	52.000
500	5	5000	NA	NA	NA	1.086	4.185	71.000
500	7	5000	NA	NA	NA	1.131	3.951	58.000
1000	5	5000	NA	NA	NA	1.093	4.456	75.000
1000	7	5000	NA	NA	NA	1.106	3.930	69.000

Table 4: Simulations for the exact ratios and the upper bounds of R_{ALG} in Theorem 3.11.

(c, d)	M	Number of Occurences	Average Upper Bound	Maximum Upper Bound
(62,3)	0.048	3	9.056	10 ($s_1 = 0.60000$)
(63,2)	0.032	8	11.273	19.64286 ($s_1 = 0.0800$)
(64,1)	0.016	33	15.685	66.66667 ($s_1 = 0.02500$)
(65,0)	0	4956	3.836	69.000 ($\delta = 0.01449$)

Table 5: The upper bounds of R_{ALG} in 5000 runs for $N = 1000$, $K = 7$.

At any partial solution, let $L' = \{i + 1, i + 2, \ldots, N\}$ be the set of items not yet placed. V is a subset of L' so that, if $j \in V$, at least one of these three conditions are satisfied:

1. $a_j > 0.5$ (When an attempt was made to place j on top of k with $a_k \geq a_j$, we would have found that $a_j + a_k > size(l)$ for all used bins l.);
2. a_j is larger than the *topSize* of any of the used bins;
3. a_j is larger than the empty space in any of the used bins.

In other words, items in V cannot be placed in any used bins so far, while items in $L' - V$ are allowed to be placed in a used bin provided there is sufficient space. Let (i) u be the sum of the sizes of bins used thus far; (ii) x be the space leftover in used bins; (iii) v be the sum of sizes of items in V; and (iv) w be the sum of the sizes of items in $L' - V$. Then we obtain a lower bound to be used in the branch & bound algorithm as follows: $u + v + \max\{0, w - x\}$.

With the B&B method above, we were able to obtain the optimal values $OPT(L)$ for instances $(N, K) = (10, 5), (10, 7), (15, 5), (15, 7)$. Then, the FF based ALG was applied to solve the same sets of data and its performance is measured by the $\frac{ALG(L)}{OPT(L)}$ ratios. In Table 4, since there are 5000 runs for each set of (N, K), the fourth column reports the average value of the ratios, the fifth column the worst value of the ratios, and the sixth column lists the percentage of ones (when $ALG(L) = OPT(L)$) within the 5000 runs. Then, we can arrive a few conclusions from the data: (i) on average, ALG returned a value roughly 5% higher than the optimal value; (ii) the worst ratio 1.491 (see the row (N, K)=(10, 5)) indicates that the ALG could use 49% more space than what is necessary; (iii) as the problem size (N, K) increases, the percentage of ones drops (the sixth column). That is, $ALG(L)$ deviates from the $OPT(L)$ value more and more often as the problem size gets larger.

In the remainder of this section, we shall discuss how we simulate the theoretical upper bound in Theorem 3.11, and its implications. First, we need to collect a few parameters: s_1, γ, δ, M. Since s_1 is the smallest bin size, γ is the smallest size of the items and $\delta = \min_{1 \leq i \leq K-1}(s_i/s_{i+1})$, they can be obtained immediately once the item sizes and the bin sizes are generated by the computer. As for M, its original definition is the limit of d/c as in (3.13). What we shall do is to run ALG to collect the actual value c (number of *type-c* bins) and d (number of *type-d* bins) and use $\frac{d}{c}$ to replace the limit value M. By this way, whenever an instance L is given and ALG is executed, we obtain not only a solution to pack the online items, but also an estimated upper bound for the ratio $ALG(L)/OPT(L)$. From Table 4, the numerical upper bounds vary from values less than 2 (column 6), to at most 100 (the last column). Their average falls around four.

If the simulated upper bound is 100, it is probably not very useful. On the other hand, take $(N, K) = (1000, 7)$ as an example where the best upper bounds over 5000 runs was 1.106. In this particular case, since the problem size is large, it is very difficult to find the

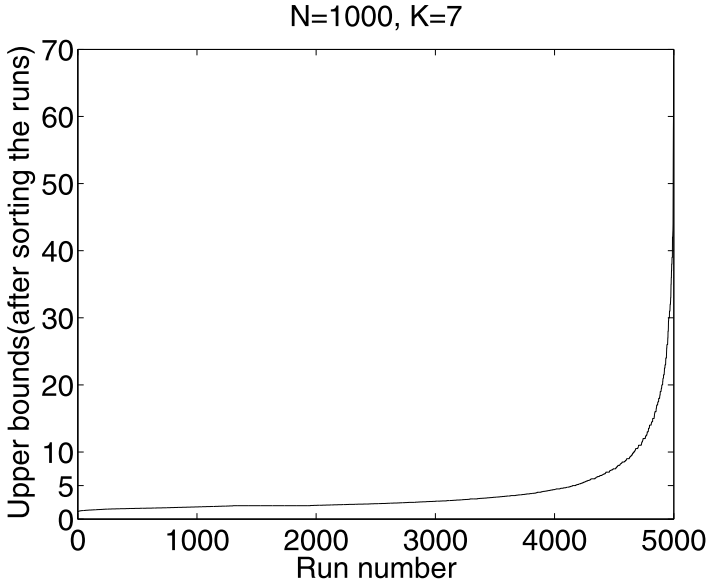

Figure 1: N=1000, K=7

optimal solution for packing the items subject to the online and LIB constraints. By the estimation

$$\frac{ALG(L)}{OPT(L)} \leq 1.106,$$

we can conclude that $OPT(L) \geq 0.904(ALG(L))$ which is valuable information regarding the unknown optimal value. From column 8, we found that the average value of the numerical upper bounds for all rows is less than 5. It indicates that the bound might not be tight enough in average, but useless upper bounds such as 100 are rare.

Figure 1 gives a complete distribution of the upper bounds in the case of 5000 runs for $N = 1000, K = 7$. Each run provides a numerical upper bound and there are 5000 such numbers — these numbers are sorted in increasing order, and presented in Figure 1. As we can see, roughly 80% of the upper bounds are less than 5.

Finally, we take a close look at the 5000 runs for $N = 1000$, $K = 7$. Table 5 illustrates the number of occurrences (out of 5000 runs) that a particular (c, d) combination was encountered. As can be seen, more than 99% of the runs (4956 out of 5000) did not produce any *type-d* bins. As a result of Lemma 3.10, the upper bound is $\frac{1}{\delta}$. The bound is loose when δ is small. See the last row in Table 5 where the worst bound happened at $\delta = 0.01449$. For the remaining 1% of the instances (44 out of 5000), the d value was still very small (≤ 3), so that $M = d/c$ is nearly 0, and the upper bound in equation (3.13) can be approximated by $\frac{1}{s_1(1-\delta)}$. In this case, the large bounds occur when there is a large δ or a small s_1. See, for example, the second row in Table 5 where $s_1 = 0.08$ and the third row with $s_1 = 0.025$. Since a loose bound means that the ALG(L) value could miss the OPT(L) by a lot, the implication is that the FF algorithm could perform badly when some of the available bins are very small (a small s_1), or there are no bins of moderate size (if we interpret a small δ to be some i such that $s_i << s_{i+1}$).

5 Scope for Further Research

In this paper, we give a theoretical upper bound of the FF method for packing items online with the LIB constraint. This result is interesting not only because it is the first bound of this kind given in literature, but also because the numerical evidence supports it.

However, in Theorem 3.11, M could be very large or even infinity. (This is because we were unable to estimate the ratio d/c.) If this happens, the bound is surely very loose, but FF is not necessarily as bad. In fact, our numerical simulations showed that, practically, M is often very small (the largest M was 0.048 in our simulation). Moreover, FF is fairly reliable (on average, FF returned a value roughly 5% higher than the optimal value whereas in the worst example, FF used 49% more space than optimal).

Several future research directions as a follow-up to this paper are proposed below. All problems referred to here are online LIB versions.

- Try to estimate the d/c value and improve the bound by replacing the parameter M with a more definite number.
- Bin Covering: The discussion in this paper can be extended to its Bin Covering counterpart.
- Testing of HF: A Harmonic Fit (HF) heuristic could be developed and experimentally tested for the online and LIB version of VSBP.
- Higher dimensions: All problems considered here can be extended to their two and three dimensional counterparts.

Acknowledgements

All three authors would like to thank the two anonymous referees for their very useful comments. The second author (Manyem) would like to thank the Australian Academy of Science for their travel support in March-April 2005.

References

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela and M. Protasi, *Combinatorial Optimization Problems and Their Approximability Properties*, Springer, Berlin, 1999.

[2] M. Carlyle, K. Knutson, and J. Fowler, Bin covering algorithms in the second stage of the lot to order matching problem, *J. Oper. Res. Soc.* 52 (2001) 1232–1243.

[3] E.G. Coffman, M.R. Garey and D.S. Johnson, Bin packing approximation algorithms: A survey, in *Approximation Algorithms for NP-Hard Problems*, D. Hochbaum (ed.), PWS Publishing Company, Boston, MA, 1997, pp 46–93.

[4] J. Csirik, An on-line algorithms for fariable sized bin packing, *Acta Informatica* 26 (1989) 697–709.

[5] L. Finlay and P. Manyem, Online LIB problems: Heuristics for bin covering and lower bounds for bin packing, *RAIRO — Operations Research* 39 (2005) 163-183.

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

[7] P. Manyem, Bin packing and covering with longest items at the bottom: online version, *The ANZIAM Journal (formerly Journal of the Austral. Math. Soc., Series B)*, 43 (2002) E186–E231.

[8] P. Manyem, Uniform sized bin packing and covering: online version, in *Topics in Industrial Mathematics*, J.C. Misra (ed.), Narosa Publishing House, New Delhi, 2003, pp. 447–485.

[9] P. Manyem, R.L. Salt, and M.S. Visser, Lower bounds and heuristics for online LIB bin packing and covering, in *Proceedings of the 13th Australasian Workshop on Combinatorial Algorithms (Fraser Island, Queensland, Australia)*, 2002, pp. 11–42.

[10] P. Manyem, R.L. Salt, and M.S. Visser, Approximation lower bounds in online LIB bin packing and covering, *J. Automata, Languages and Combinatorics* 8 (2003) 663–674.

[11] S.S. Seiden, On the online bin packing problem, *J. ACM* 49 (2002) 640–671.

[12] S.S. Seiden, R. Stee and L. Epstein, New bounds for variable-sized online bin packing, *SIAM J. Comput.* 32 (2003) 455–469.

[13] A. van Vliet, An improved lower bound for on-line bin packing algorithms, *Information Processing Letters*, 43 (1992) 277–284.

[14] G.J. Woeginger and G. Zhang, Optimal on-line algorithms for variable-sized bin covering, *Operations Research Letters* 25 (1999) 47–50.

[15] W. Xing and F. Chen, A-shaped bin packing: Worst case analysis via simulation, *J. Ind. Manag. Optim.* 1 (2005) 323–335.

Manuscript received 23 January 2006
revised 5 October 2006, 14 February 2007
accepted for publication 26 February 2007

J.Y. Lin
National Chiayi University, Chiayi, Taiwan
E-mail address: jylin@mail.ncyu.edu.tw

P. Manyem
Centre for Informatics and Applied Optimization, University of Ballarat, Australia
E-mail address: p.manyem@ballarat.edu.au

R.L. Sheu
National Cheng-Kung University, Tainan, Taiwan
E-mail address: rsheu@mail.ncku.edu.tw