Bin packing and covering with longest items at the bottom: online version

P. Manyem*

Received November 8, 2000; revised March 21, 2002

Abstract

We consider the NP hard problems of online bin packing and online bin covering while requiring that larger (or longer, in the one-dimensional case) items be placed at the bottom of the bins, below smaller (or shorter) items. Bin sizes can be uniform or variable. If variable, the bin sizes are drawn from a finite collection. In uniform sized

*CIAM, School of Mathematics, University of South Australia, Mawson Lakes, SA 5095, AUSTRALIA. mailto:Prabhu.Manyem@unisa.edu.au

[0]See http://anziamj.austms.org.au/V43/E044 for this article and ancillary services,
 Published June 17, 2002; amended August 9, 2002.

online bin packing, we prove an upper bound of two on the approximation ratio for special cases of the problem and provide computational results for the general case using a variation of the first fit heuristic. In uniform sized online bin covering, we prove a non-approximability result and present a modified first fit heuristic. In online variable-sized bin covering, we show that the approximation ratio guaranteed by our heuristic is a function of bin lengths.

Contents

1 Background

In the classical one-dimensional bin packing problem, we consider 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 unit sized bins such that the number of bins used is minimized. A feasible solution is one where the sum of the sizes of the items in each bin is at most equal to the bin size. The 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, where s_1 is the largest (or just *longest*, in the one-dimensional case) bin size with $s_1 = 1$. Size s_k is the smallest.

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), and the next item $i + 1$ of length a_{i+1} becomes known only after item i has been assigned to its bin. A bin is said to be *used* if it contains at least one item (of non-zero length).

In all versions of bin packing (Section 2) and bin covering (Section 3), it is assumed that there is an infinite supply of bins of any size. Hence running out of bins to place items is never an issue. Bin packing is NP-complete in all its four variations considered here — bin size is either uniform or variable, bin filling is either online or offline. Coffman et al. [3] provide a comprehensive review of approximation solutions to bin packing. A more recent review appears in [4].

Our version of bin packing: The online bin packing version we consider here imposes an additional requirement: In any bin, for any pair of items i and j, if the size of j is greater than that of 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*.

This notion is generalized. For a given list of items L, if a *sequence* is imposed on the placement of items in any bin, how much harder does the problem become? Alternatively, suppose each item i has a value v_i . Then we could impose the constraint that an item with higher value should be placed below an item with lower value in any bin.

Our version of bin packing has applications in the transportation industry. If long items are placed at the bottom 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 sizes of items in a bin should total up to *at least* the bin size. The online LIB variation of bin covering, which requires placement of longer items below shorter items, is treated in Section 3.

Organization of this paper: In Sections 2.2 and 2.3, we present the behaviour of the well-known Next Fit and First Fit algorithms respectively for the online LIB version of bin packing with uniform size bins. Sections 3 and beyond consider the online LIB bin covering problem. The organization of Section 3 is described at the beginning of that section.

2 Bin Packing

Problem statement: online LIB uniform sized bin packing: We are given an infinite supply of unit-sized bins, and n items, each item with size in $(0, 1]$. Each item should be placed in a bin assigned to it (on top of items previously placed in that bin) as soon as it arrives. This placement cannot be changed later. A feasible solution is one where the sum of the sizes of the items in each used bin is at most one. The goal is to find a feasible solution that minimizes the number of used bins. In any used bin, longer items should be placed below shorter items (the LIB constraint).

The online condition essentially reduces to the following *online constraint*: 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,

$$[i \text{ is below } j \text{ in a used bin}] \implies [i < j]\,. \tag{1}$$

Similarly, the LIB *constraint* is stated as follows, for a used bin:

$$[i \text{ is below } j \text{ in a used bin}] \implies [a_i \geq a_j]\,. \tag{2}$$

2.1 Approximation ratios

Given an instance of bin packing (a list L), let OPT(L) and A(L) be the solution values obtained by the exact and approximation algorithms respectively.

As in [9], we define the *asymptotic approximation ratio* R_A^∞ for approximation algorithm A as

$$R_A^\infty = \lim_{s\to\infty} \sup_L \left\{ R_A^{(L)},\ \textsc{opt}(L) \geq s \right\}, \quad \text{where} \quad R_A^{(L)} = \frac{\textsc{a}(L)}{\textsc{opt}(L)}. \tag{3}$$

When bin sizes are variable, this generalizes to

$$R_{A,\mathcal{B}}^\infty = \lim_{s\to\infty} \sup_L \left\{ R_{A,\mathcal{B}}^{(L)},\ \textsc{opt}(L,\mathcal{B}) \geq s \right\}, \quad \text{where} \quad R_{A,\mathcal{B}}^{(L)} = \frac{\textsc{a}(L,\mathcal{B})}{\textsc{opt}(L,\mathcal{B})}, \tag{4}$$

where $\mathcal{B}$ is the collection of bin sizes (see Section 1). For a class of inputs $\mathcal{C}$, $R_A^{\mathcal{C}}$ and $R_{A,\mathcal{B}}^{\mathcal{C}}$ are similarly defined. Observe that $1 \leq R_A^\infty, R_{A,\mathcal{B}}^\infty \leq \infty$. The lower these ratios, the better the approximation algorithm.

2.2 Bin packing: next fit algorithm

The NF (Next Fit) algorithm always maintains a *current* bin b_0, and two parameters `topSize` and `totalSize`, which are the size of the topmost item in b_0, and the sum of the sizes of the items held by b_0, respectively. Initially, the current bin b_0 is simply any empty bin, `topSize` $= 1$ and `totalSize` $= 0$. A bin is said to be *uncovered* if the `totalSize` of the bin is < 1.

If an arriving item i has a size of at most $1 -$ `totalSize`, and `topSize` $>$ size$(i) = a_i$, then i is placed in b_0, `topSize` is set to a_i and `totalSize` is increased by a_i. Otherwise, b_0 is closed (never to be opened again), an empty

bin is opened and named as the *current bin* b_0, item i is placed in this (new) b_0, `topSize` and `totalSize` are set to a_i.

The running time of the NF algorithm is linear in n, the number of items in the list. It can be easily shown that the NF algorithm can have a worst case approximation ratio of $\theta(n)$. Consider the following list $L = (i : 1 \leq i \leq n)$, with the sizes a_i of items as follows $(0 < \epsilon \leq 1)$:

- $a_i = \epsilon$, if $i = 1 \pmod 4$,
- $a_i = 2\epsilon$, if $i = 2 \pmod 4$,
- $a_i = 3\epsilon$, if $i = 3 \pmod 4$, and
- $a_i = 4\epsilon$, if $i = 0 \pmod 4$.

Assume that $n\epsilon \leq 1$. As the items arrive in sequence, the NF algorithm would place them in bins starting from b_1 as follows:

$$\begin{array}{llll} b_1 : \{1\} & b_2 : \{2\} & b_3 : \{3\} & b_4 : \{4, 5\} \\ & b_5 : \{6\} & b_6 : \{7\} & b_7 : \{8, 9\} \\ & b_8 : \{10\} & b_9 : \{11\} & b_{10} : \{12, 13\} \\ & \ldots . & & \end{array} \tag{5}$$

Assuming that n is a multiple of four, the number of bins that the NF algorithm needs, k, is given by $(3n/4) + 1 = \theta(n)$. However, there exists a

solution that requires just four bins:

$$\begin{aligned} b_1 &: \{1, 5, 9, 13, \ldots\} \\ b_2 &: \{2, 6, 10, 14, \ldots\} \\ b_3 &: \{3, 7, 11, 15, \ldots\} \\ b_4 &: \{4, 8, 12, 16, \ldots\}\,. \end{aligned} \tag{6}$$

Bin b_i $(1 \leq i \leq 4)$ contains exactly $n/4$ items, each of size $i \times \epsilon$. Bin b_4 is filled closer to capacity than the other three bins, since each item in b_4 is of size 4ϵ. The sum of the sizes of items in $b_4 = (n/4) \times (4\epsilon) = n\epsilon$. Since we assumed that $n\epsilon \leq 1$, all four bins are filled at or below their capacities.

Lemma 1 *The asymptotic approximation ratio for the* NF *algorithm,* R^{∞}_{NF}, *is of the order* $\theta(n)$ *for the online* LIB *version of bin packing.*

Obviously, the approximation ratio R_A cannot be worse than n for any heuristic A for this problem.

2.3 Bin packing: first fit algorithm

Now consider the behaviour of the First Fit (FF) algorithm for the LIB version of bin packing. 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`(j) and `totalSize`(j), representing the size of the topmost item in b_j and the sum of the sizes of the items in b_j respectively.

The FF algorithm scans b_1 through b_m in that order. For each bin b_j, it checks if (i) $a_i \leq$ `topSize`(j), and (ii) $a_i \leq 1 -$ `totalSize`(j). The FF algorithm places item i in the first such bin b_j that satisfies both the conditions above and updates `topSize`(j) as well as `totalSize`(j), just as in the NF algorithm. If no such bin among b_1 through b_m satisfies these conditions, the FF algorithm opens a new bin b_{m+1} in which to place i.

Algorithm 1. First Fit (online LIB bin packing version): *Given* items $1, \ldots, N$ with sizes $a_1, \ldots, a_N$, $0 < a_i \leq 1$ for $1 \leq i \leq N$; the *Running Time* is $\mathcal{O}(N^2)$.

```
1 nBin (number of bins used) = 1;
2 topSize[1] = 1;
3 totalSize[1] = 0;
4 for (item = 1 to N) do
5     allocated[item] = NO;
6     bin = 1;
7     While (bin ≤ nBin AND allocated[item] == NO) do
8           X = (topSize[bin] ≥ size[item]);
```

```
9           Y = (1-totalSize[bin] ≥ size[item]);
10          if (X == true AND Y == true) then
11              place item in bin;
12              update topSize[bin] and totalSize[bin];
13              allocated[item] = YES;
14          end if
15          bin = bin +1;
16    end While
17    if (allocated[item] == NO) then (item not placed in any bin)
18          nBin = nBin +1; (new, fresh, unused bin)
19          place item in nBin;
20          topSize[nBin] = size[item];
21          totalSize[nBin] = size[item];
22          allocated[item] = YES;
23    end if
24 end for
```

The behaviour of the FF algorithm is studied for these three mutually exclusive and collectively exhaustive cases with respect to the sizes of the items in L:

1. the sizes are MND (monotonic non-decreasing);
2. the sizes are MNI (monotonic non-increasing); and
3. neither of the above.

For the first two cases, we prove a worst case upper bound of two on the approximation ratio. For the last case, we provide computational results (where the approximation ratios are never worse than two).

2.4 First Fit: monotonic non-decreasing sizes

Here, the item sizes in L are MND (monotonic non-decreasing). The FF algorithm works as follows: if an arriving item i is longer than its predecessor $i-1$, then it is placed in its own (new) bin b_{m+1}. Otherwise, if $a_i = a_{i-1}$, then place i on top of $i-1$ in b_m (unless $a_i < 1 - \texttt{totalSize}(m)$, in which case i is placed in a new bin). The FF algorithm never needs to check bins b_1 through b_{m-1} here. Clearly, an exact algorithm (one that always returns an optimal solution) can do no better than FF. It follows that the approximation ratio of the FF algorithm for this class of instances, $R_{FF}^{(I)}$ (I for *Increasing*), is one.

2.5 First Fit: monotonic non-increasing sizes

Now suppose the item sizes in L are MNI (monotonic non-increasing). Here, even the NF algorithm can guarantee an upper bound of two on the approximation ratio asymptotically.

Let the *Space Used (*SU*)* factor of a bin be defined as the sum of the sizes

of items in the bin.

Observe that if an item with size ≥ 0.5 is placed in a bin, then the bin is at least half full. That is, for such a bin, the SU factor is at least 0.5. Since the input is MNI, the items in the input with size ≥ 0.5 will be at the beginning of the input list, and hence the first *few* bins will have SU ≥ 0.5. That is, the the bins used in the beginning will naturally have SU ≥ 0.5, simply because they contain items of size ≥ 0.5 in them.

What about the bins where all items are of size < 0.5? If the item at the bottom of a bin is of size < 0.5, then this is true for all items in the bin. We will attempt to show that except for perhaps the last used bin, all bins will have an SU factor ≥ 0.5, and hence asymptotically, the approximation ratio for this input class (MNI) is at most two.

Consider a bin b where the item at the bottom, say i, is of size < 0.5, that is, $a_i < 0.5$. Since the input is MNI, this implies that the next item in the input is of size < 0.5, that is, $a_{i+1} < 0.5$. Since $a_i + a_{i+1} < 0.5$, item $i+1$ can be placed in the same bin b.

If $a_i + a_{i+1} \geq 0.5$, then we have achieved our goal of obtaining SU$(b) \geq 0.5$ for this bin b. Otherwise, if $a_i + a_{i+1}$ is still less than 0.5, the space left in the bin is greater than half. Since the next item in the input, $i+2$, is of size at most a_{i+1}, and $0.5 \geq a_{i+1} \geq a_{i+2}$, it follows that $i+2$ can comfortably fit into bin b. This process continues until either no more items can fit into b, or there are no more items in the input list. If there is an item k (size a_k) that cannot fit into b, then it is of size less than half, and SU(b) should be

greater than half. This is because SU$(b) + a_k > 1$, and $a_k < 0.5$.

Thus we have shown that except for perhaps the last used bin, all other bins have an SU factor greater than half. This implies that

Lemma 2 *The worst case asymptotic approximation ratio for the* MNI *class of problem instances in online* LIB *bin packing is at most two, using the* NF *heuristic.*

2.6 First Fit (general case): computational testing

We now consider the general case where the inputs are neither monotonic non-increasing or non-decreasing (neither MNI nor MND). In this section, we provide results of computational testing, since it is still inconclusive whether the worst case (guaranteed analytical) approximation ratio over all possible instances for this problem has a constant upper bound.

The simulations were carried out on a 300 Mhz Intel PC running RedHat Linux 5.2 with 512 Mbytes memory. The code was written in C and compiled using GNU's gcc compiler. Since the exact algorithm runs in exponential time in the worst case and the FF heuristic runs in polynomial time, we did not collect statistics on running time. Instead, we focussed primarily on the approximation ratios. For the exact algorithm, a branch and bound (B & B) routine was implemented. Lower bounding techniques were used to prune branches of the B & B tree. At any partial solution, we let

1. u = number of bins used thus far;
2. v = sum of sizes of items not yet placed, satisfying at least one of these three conditions: (i) its size is greater than 0.5 , or (ii) its size is greater than the `topSize` of any of the used bins, or (iii) its size is greater than the empty space in any of the used bins;
3. w = sum of the sizes of bins not yet placed which do not belong to the categories mentioned above in Case (ii); and
4. x = space left over in used bins.

The lower bound was then computed as $u + \lceil v \rceil$ (if $x \geq w$), or $u + \lceil v + w - x \rceil$ (if $x < w$) .

The number of items in an instance, N, was varied from 10 to 30 in steps of five. For each N, we performed 1000–5000 runs of the simulation, depending on the time taken to run.

Table 1 presents results of

- the average R_{FF} (*the average ratio*),
- the worst R_{FF} (*the maximum ratio*), and
- the percentage of instances where R_{FF} was one (the lowest possible) (*the percentage of ones*).

No.(Items)	Max.Ratio	Ave.Ratio	Runs	% of Ones	Run.Time
10	1.500	1.015	5000	91.24	30 secs
15	1.333	1.020	5000	82.54	46 secs
20	1.250	1.023	5000	74.74	6.5 mins
25	1.273	1.025	5000	67.04	13 hours
30	1.250	1.026	1000	61.3	4 days
35	1.200	1.028	1000	52.3	18 days

TABLE 1: Bin Packing: approximation ratios for various list sizes

Note that as N increases, there is an overall decrease in the maximum approximation ratio (column 2). There is a steady increase in the average ratio (column 3). And most interestingly, in column 5, the percentage of instances where the ratio is one (the heuristic produces a solution as good as the exact algorithm) decreases steeply. This decrease is more remarkable in the case of uniform size bin covering in Section 3.2.3.

The last two observations regarding the percentage of ones suggest that as the size of the list grows, the FF heuristic is less likely to find an optimal solution to the problem. This is the likely cause of the increase in average ratio (although the maximum ratio seems to decrease).

The minimum ratio is of course one for any list size, for any number of instances. The running times (last column) are not the CPU times, but the overall time, just to provide an idea of the order of time taken to run

the instances of the heuristic and the exact (branch and bound) algorithm combined. Needless to say, this depends on the load on the CPU from other users.

The numbers in Table 1 suggest that an approximation ratio of less than two occurs "most of the time", although it might still be possible that a ratio greater than two could occur in some rare instances.

Conjecture 3 *The worst case asymptotic approximation ratio of the modified* FF *heuristic for the online uniform sized* LIB *bin packing problem is at most two.*

3 Bin covering

Problem statement: online LIB uniform sized bin covering: We are given an infinite supply of unit-sized bins, and n items, each item with size in $(0, 1]$. Each item should be placed in a bin assigned to it (on top of items previously placed in that bin) as soon as it arrives. This placement cannot be changed later. A bin is *covered* if the sum of the sizes of the items in the bin is at least one. The goal is to maximize the number of covered bins. The solution is feasible

- even if the total sum of item sizes in a bin is greater than one, and

- only if item(a) is placed below item(b) in a bin whenever size(a) > size(b) (the LIB constraint, Section 1).

As in the previous section, `topSize`(j) is the size of the item at the top of bin j, and `totalSize`(j) is the sum of the sizes of the items in bin j. When there is no confusion, we shall use `topSize` and `totalSize` without the index j. In a covered bin j, `totalSize`(j) is at least as high as the size of bin(j). A unit-sized bin is *covered* if its `totalSize` is at least one.

In a covered bin, it is indeed possible for an item to be protruding or sticking out of the bin. There could also be items that are completely outside the bin, stacked on top of other items, and yet *belonging* to the bin — this is still feasible — however, this only moves the solution away from optimality. The objective is to find a feasible solution that *maximizes the sum of the sizes of the covered bins* — in the case of unit-sized bins, this is the same as maximizing the number of covered bins.

Two of the earliest works to appear on bin covering were by Assmann [1] and Assmann et al. [2]. In [2], they provide polynomial time heuristics with an asymptotic worst case ratio of 4/3 for the offline problem and 2 for the online problem when all bins are of unit size. Csirik and Totik [5] show that there can be no polynomial time heuristic that guarantees an asymptotic approximation ratio better than 2 for online problems with unit-sized bins. Csirik et al. [6] provide two algorithms for offline bin covering. Woeginger and Zhang [10] provide a polynomial time heuristic for the online version with variable sized bins. For a survey of bin covering problems, see [7] and

(especially for online problems) [8].

This section is organized as follows (see Table 2). Section 3.2 considers uniform sized online LIB bin covering, where all bins are of the same size. Section 3.3 briefly reviews the algorithm in [10] — this Section (3.3) and the ones that follow do not require bin sizes to be uniform. All of Section 3 enforces the LIB constraint except for subsection 3.3.1.

In Section 3.4, we present the case when shorter items (that were previously placed) can be removed to make way for longer items and then put back in the bin. In other words, when an item i (of length a_i) arrives, a bin b_j is assigned to it, its position in the bin is determined according to its length, and the item is placed in its computed position in b_j . For example, if i is determined to be the second longest item in b_j , then it is placed second from the bottom, before the next item $i + 1$ of length $a_{i+1} \in L$ arrives for placement. Let this version be named as VSBC$_A$ (Variable Sized Bin Covering, the subscript A for *allowing* placement changes) — the LIB constraint is still enforced here.

We first observe that VSBC$_A$ is as approximable as VSBC$_G$, the online version of VSBC with *no* LIB constraint. This is then extended to include all offline problems in one-dimensional bin packing and bin covering which require that the items be placed in bins in the order of their lengths.

In Section 3.5, we consider the case when an item, once placed, cannot be removed from the bin. Here an arriving item has to be placed at the top of its assigned bin. The LIB constraint is enforced. Call this version VSBC$_P$

Subsection	Bin Size	LIB Enforcement	Rearrangement of Items Allowed Within a Bin
3.2	uniform	yes	no
3.3	variable	no	no
3.4	variable	yes	yes
3.5	variable	yes	no

TABLE 2: Problems in online bin covering

(P for *preventing* placement changes). We present results for this case very similar to those in [10]. The problem considered here is simpler than VSBC$_G$ due to our belief in Conjecture 7 — we assume a discrete set of sizes for items whose sizes are less than s_k, the smallest bin size.

3.1 Approximation ratios

When bin sizes are uniform, (3) is modified as follows, for bin covering:

$$R_A^{\infty} = \lim_{s\to\infty} \sup_L \left\{ R_A^{(L)},\ \mathrm{OPT}(L) \geq s \right\}, \quad \text{where} \quad R_A^{(L)} = \frac{\mathrm{OPT}(L)}{\mathrm{A}(L)}. \qquad (7)$$

Similarly, when bin sizes are variable, (4) is modified to

$$R^{\infty}_{A,\mathcal{B}} = \lim_{s\to\infty} \sup_{L} \left\{ R^{(L)}_{A,\mathcal{B}},\ \textsc{opt}(L,\mathcal{B}) \geq s \right\}, \quad \text{where} \quad R^{(L)}_{A,\mathcal{B}} = \frac{\textsc{opt}(L,\mathcal{B})}{\mathrm{A}(L,\mathcal{B})}. \tag{8}$$

As in Section 2.1, $1 \leq R^{\infty}_{A}, R^{\infty}_{A,\mathcal{B}} \leq \infty$.

3.2 Bin covering: uniform bin size

In this section we provide a variation of the First Fit heuristic to the problem of online LIB bin covering with uniform sized bins (USBC). Let each bin be of unit size. The algorithm we use here is exactly the same as Algorithm 1 except that line 10 is replaced as follows:

```
if (topSize[bin] ≥ size[item] AND totalSize[bin] <1) then
```

since we no longer compare `totalSize[bin]` with the item size to place an arriving item. If we consider placing item i on top of item j in bin b, we proceed with this placement only if $a_i \leq a_j$ and the bin is filled below its capacity (that is, the bin is yet to be covered).

3.2.1 Algorithm analysis

First observe that if the items arrive in the order of strictly increasing size, First Fit will place each item in its own separate bin. Clearly this is the best any algorithm can do. Thus the solution produced by First Fit is optimal for such an input. If the items arrive in the sequence $a_1, \ldots, a_n, a_{n+1}, \ldots, a_{n+m}$ where the sequence $a_1, \ldots, a_{n+1}$ is strictly increasing, and $a_{n+1} = \cdots = a_{n+m} = 1$, the number of bins covered by First Fit as well as an optimal algorithm is m. When $m = 0$, the optimal algorithm covers no bins.

Secondly, assume that a certain bin is uncovered. Let T_o and T_f be the initial and final values of `totalSize`, before and after the placement of item i respectively. Thus for this bin $T_o < 1$ and `topSize` (the size of an item j currently at the top of the bin) < 1. Suppose that placing a new item i on top of j covers the bin. Since a_i (size of i) $\leq$ `topSize`, it follows that $a_i < 1$. Hence

$$T_o < 1 \leq T_f = T_o + a_i < 1 + a_i < 2\,, \tag{9}$$

$$\text{Aspect Ratio of bin} = \frac{T_f}{\text{bin size}} = \frac{T_f}{1} < 2\,. \tag{10}$$

Clearly the aspect ratio should be at least one for a covered bin.

Lemma 4 *The Aspect Ratio of a covered bin is in the interval* $[1, 2)$.

3.2.2 Uniform bin size: non-approximability

It is easy to show that the FF (First Fit) heuristic in Section 3.2 can have an approximation ratio of $\theta(n)$ in the worst case, where n is the list size. Consider the following input ($0 < \epsilon < 1$ and $n\epsilon/3 \leq 0.1$):

$$0.3,\ 0.2,\ 0.1,\ 0.3,\ 0.2,\ 0.1+\epsilon,\ 0.3,\ 0.2,\ 0.1+2\epsilon,\ \ldots,\ 0.3,\ 0.2,\ 0.2-\epsilon,\ 1.0\,.$$

The FF algorithm would place this input in bins in the following order, and thus the number of bins covered is only one (the last bin):

$$\begin{aligned}
\text{Bin 1:} &\quad (0.3,\, 0.2,\, 0.1)\,,\\
\text{Bin 2:} &\quad (0.3,\, 0.2,\, 0.1+\epsilon)\,,\\
\text{Bin 3:} &\quad (0.3,\, 0.2,\, 0.1+2\epsilon)\,,\\
&\ \vdots\\
\text{Last-but-one bin:} &\quad (0.3,\, 0.2,\, 0.2-\epsilon)\,, \quad \text{and}\\
\text{Last bin:} &\quad (1.0)\,.
\end{aligned}$$

However, there exists a solution that places all items of size 0.3 together, and all items of size 0.2 together. Four items of size 0.3 are required to cover a bin, there are about $n/3$ such items, and hence the number of bins covered by the 0.3-sized items is $\lfloor (n/3)/4 \rfloor = \lfloor n/12 \rfloor$. Similarly, five items of size 0.2 are required to cover a bin, there are about $n/3$ items in this category, and thus the number of bins covered is $\lfloor (n/3)/5 \rfloor = \lfloor n/15 \rfloor$.

The last item of size 1.0 will be in its own bin. Thus the number of bins covered is at least $\lfloor n/15 \rfloor + \lfloor n/12 \rfloor + 1 = \theta(n)$. We note that the FF algorithm covers only one bin for this input.

Lemma 5 *The approximation ratio for the* FF *heuristic for the online uniform sized bin covering (*USBC*) problem with* LIB *is of order* $\theta(n)$.

It can also be shown that no algorithm for the above problem can guarantee an asymptotic approximation ratio less than two. Let i be a positive integer equal to at least two ($i \geq 2$). For each i, let j be an integer in the interval $[1, 2i]$. Let the size of an item e_i be $(1+2i)^{-1}$. Consider the following series of inputs (lists), one for each (i, j) pair:

$$\text{List } L(i,j) = \underbrace{1-e_i, \ldots, 1-e_i}_{i \text{ times}}, \underbrace{e_i, \ldots, e_i}_{j \text{ times}}.$$

Henceforth, we simply refer to an item with a size of a_1 (a_2) as an a_1 (a_2) item. For a given i, there are i items of size $a_1 = 1 - e_i$ in the list $L(i, j)$, followed by a sequence of j items of size $a_2 = e_i$. Since j varies from one through $2i$, there are $2i$ lists for each i. Let A be an algorithm for USBC. Since $2(1 - e_i) > 1$ for any i, two a_1 items (of size $1 - e_i$) are sufficient to cover a bin, and thus the aspect ratio of such a *double stacked* bin (a bin with two a_1 sized items) is greater than one. On the other hand, in any list $L(i, j)$, the j items of size $a_2 (= e_i)$ are insufficient to cover a bin because $je_i < 1$ for all (i, j) pairs. Also, the aspect ratio of a bin with one a_1 item and one a_2 item is exactly one.

When A is given the input $L(i, j)$, at the end of the sequence of a_1 sized items (items of size $1-e_i$), let the number of double stacked bins be m. Thus m bins have been covered so far, and there are $i - m$ uncovered bins, each with one item of size a_1. If and only if $j \geq i - m$, these $i - m$ bins will be covered when A is done.

We divide the problem into two distinct cases (see Table 3).

Case 1: $i > j$ optimal solution: Since the aspect ratio of a bin with one a_2 item and one a_1 item is one, an optimal algorithm would maximize the number of such bins in the solution. However, this is limited by the number j of a_2 items. The optimal solution on such a list L would be:

- $\lfloor (i-j)/2 \rfloor$ double stacked bins, each with two a_1 items, and
- j bins, each with one a_1 item and one a_2 item,

giving an optimal solution value of $L^* = \lfloor (i+j)/2 \rfloor$ covered bins.

However, the approximation algorithm A can behave in two different ways, depending on the number of double stacked bins after the arrival of the ith item (which is the last a_1 sized item of the input).

Case 1a: number of double stacked bins $m \leq (i-j)/2$. Recall that a double stacked bin has two items of size a_1, and is covered. The number of double stacked items is thus $2m$, and the number of single stacked

a_1 items of size $1 - e_i$ is $i - 2m$. However, out of these $i - 2m$ uncovered bins, only $i - j$ of them can be covered by the a_2 items of size e_i. The number of a_2 items needed to cover the single stacked a_1 items is insufficient, or just sufficient. Thus, $A(L)$, the number of bins covered by A, is $m + j$. The approximation ratio for algorithm A is equal to

$$R_A = \frac{L^*}{\mathrm{A}(L)} = \frac{\lfloor (i+j)/2 \rfloor}{m+j}. \tag{11}$$

When $m = 0$ (no a_1 item is double stacked), since i/j can be as high as i (recall that $j \in [1, 2i]$), R_A becomes

$$R_A = \frac{\lfloor (i+j)/2 \rfloor}{j} \leq \frac{1}{2}(1 + \frac{i}{j}) \leq \frac{1}{2} + \frac{i}{2} = \theta(i). \tag{12}$$

At the other extreme, if $m = \lfloor (i-j)/2 \rfloor$, R_A becomes one, since this is the optimal solution described at the beginning of Case 1.

Thus when m is in the range $[0, \lfloor (i-j)/2 \rfloor]$, the worst value for R_A is $\max(1, \theta(i)) = \theta(i)$.

Case 1b: number of double stacked bins $m \geq (i-j)/2$. See that m cannot be greater than $i/2$. This is similar to Case 1a, except that there are sufficient a_2 items to cover the $i - 2m$ number of single stacked a_1 items. Thus $\mathrm{A}(L) = m + (i - 2m) = i - m$. Note that the *left over* a_2 items cannot cover a bin on their own, since $je_i < 1$.

When $m = \lceil (i-j)/2 \rceil$, the solution is the same as the optimal solution described above with $R_A = 1$. At the other extreme, when $m = \lfloor i/2 \rfloor$, since $j \leq 2i$ and $j < i$ (by assumption), R_A becomes

$$R_A = \frac{L^*}{\mathrm{A}(L)} = \frac{\lfloor (i+j)/2 \rfloor}{i-m} = \frac{\lfloor (i+j)/2 \rfloor}{\lceil i/2 \rceil} \leq \frac{i+j}{i} = 1 + \frac{j}{i} < 2\,. \quad (13)$$

Hence when m is in the range $[\lceil (i-j)/2 \rceil, \lfloor i/2 \rfloor]$, the worst R_A value is $\max(1,2) = 2$.

Case 2: $i \leq j$ Optimal Solution: Since there is a sufficient number of a_2 items to cover each a_1 item, an optimal algorithm would place each a_1 item in its own bin, and then cover each such bin with an a_2 item. Thus the optimal solution has a value of $L^* = i$ covered bins. The remaining a_2 items, $j - i$ in number, cannot cover a bin.

There is only a single case for the approximation algorithm A, since there are always sufficient a_2 items available to cover the single stacked a_1 items. The analogue of Case 1a does not exist here. Thus, $A(L)$, the number of bins covered by A, is given by $(m) + (i - 2m) = i - m$. The approximation ratio R_A is given by $i/(i-m)$.

If $m = 0$ (no double stacked items), $\mathrm{A}(L) = i$, and hence this is just the optimal solution with $R_A = 1$. At the other extreme, if $m = \lfloor i/2 \rfloor$, R_A becomes

$$R_A = \frac{L^*}{\mathrm{A}(L)} = \frac{i}{i-m} = \frac{i}{\lfloor i/2 \rfloor} \leq \frac{i}{(i-1)/2} = 2\frac{i}{i-1}\,. \quad (14)$$

	Solution Value		R_A	
	Heuristic A	L^* (Optimal)	$m = 0$	$m = \lfloor i/2 \rfloor$
Case 1a	$m + j$	$\lfloor (i + j)/2 \rfloor$	$\theta(i)$	
Case 1b	$i - m$	$\lfloor (i + j)/2 \rfloor$		< 2
Case 2	$i - m$	i	1	2

TABLE 3: Asymptotic approximation ratios (deterministic heuristic)

For large i, R_A approaches the limit 2. As defined in (7), $R_A^\infty = 2$.

Table 3 summarizes the results.

Discussion: We have just demonstrated the behaviour of a deterministic approximation algorithm A on a randomized input. With no prior knowledge of j, the number of e_i in the input, A has to make a deterministic decision on the number m of bins to be double stacked with a_1 items, before the a_2 items start arriving.

Heuristic A can follow one of the following three deterministic strategies.

1. Looking at column 4 in the table, if A decides to single stack all a_1 items, $m = 0$, in which case we observe from Table 3 that R_A can be as good as one, but as bad as $\theta(i)$. Hence the (worst) approximation ratio here is $R_A = \theta(i)$ (and so is R_A^∞).

2. At the other extreme (the last column in the table), if A decides to double stack all i number of $1 - e_i$ items ($i - 1$ number when i is odd), R_A can be as good as (or as bad as) two. Thus the worst ratio R_A (and hence R_A^∞) is two.

3. If A decides that a certain fraction $f = 2m/i$ of a_1 items will be double stacked, then the guaranteed R_A (and R_A^∞) can be as bad as $\theta(i)$ or two, depending on the values of f and j:

   (a) if $f \geq 0.5$, this will fall into Cases 1b or 2 depending on j, and thus the guaranteed asymptotic approximation ratio R_A^∞ is two;

   (b) if $f < 0.5$, all cases (1a, 1b and 2) are possible, which means a guaranteed R_A^∞ value of $\theta(i)$.

Thus, of all three strategies for A discussed above, the best strategies are 2 and 3a, since these have the lowest guaranteed approximation ratio R_A^∞ of two.

Theorem 6 *No (deterministic) approximation algorithm for the Online Uniform Sized Bin Covering problem with* LIB *can guarantee an asymptotic approximation ratio of less than* 1.5 *(see Appendix A), unless* P = NP.

Conjecture 7 *No approximation algorithm for the Online Uniform Sized Bin Covering problem with* LIB *can guarantee an asymptotic approximation ratio that is a constant, unless* P = NP.

3.2.3 Computational testing

In light of our observation in Conjecture 7 above, since there is unlikely to be a polynomial time heuristic that guarantees a constant bound on the approximation ratio, we carried out simulations to study the performance of the modified FF heuristic in practice. Again, the focus is on the approximation ratios rather than the running times.

The tests conducted were similar to those for bin packing (Section 2.6). The FF and B & B (exact) algorithms were implemented and tested against each problem instance generated. Unlike bin packing which is a minimization problem, bin covering maximizes the objective function. Hence, *upper bounding* techniques were used to prune branches of the B & B tree. At any vertex of the B & B tree, the upper bound was computed *prior* to traversing any of the sub-trees rooted at this vertex. Thus if the upper-bound test failed, all these sub-trees were pruned. The sub-tree determines in which of the used bins (or a new bin) the next item will be placed.

At any partial solution (when not all items in the list have been placed in bins), let:

1. x = number of bins used;
2. y = number of bins covered;
3. $pI[b]$ = sum of sizes of items that could be placed on top of (used, uncovered) bin b (allowed by size restrictions);

4. F = set of used uncovered bins for which $pI[b] > 0$;
5. $pB[i]$ = number of bins in F on top of which item i could be placed (item i has not been placed yet);
6. nB = sum of the sizes of items yet to be placed, for which $pB[i] = 0$ (these items have to be placed in new, unused bins);
7. oB = sum of the sizes of items yet to be placed, for which $pB[i] > 0$ (these items could be placed in bins in F);
8. u = total occupied space in bins in F; and
9. v = total empty space in bins in $F = |F| - u$.

The upper bound was then computed as

$$\begin{aligned} y + \lfloor u + oB \rfloor + \lfloor nB + u + oB - \lfloor u + oB \rfloor \rfloor \quad & \text{if } oB \le v\,, \\ \text{or} \quad y + \lfloor nB + oB - v \rfloor + |F| \quad & \text{if } oB > v\,. \end{aligned}$$

Three different values for N, the number of items in an instance, were considered: 10, 15 and 20. For each N, we performed 1000–5000 runs of the simulation, depending on the time taken for the runs (last column of Table 4). This table presents results of: (a) the average R_{FF} in column 3; (b) the worst R_{FF} in column 2; and (c) the percentage of instances where R_{FF} was one (the lowest possible) in column 4.

No.(Items)	Max.Ratio	Ave.Ratio	Runs	% of Ones	Run.Time
10	4.00	1.414	5000	37.42	39 secs.
15	5.00	1.425	5000	12.26	27 minutes
20	3.00	1.417	1000	3.5	3 days

TABLE 4: Bin covering: approximation ratios for various list sizes

Note that the bin covering exact (branch and bound) algorithm runs slower than its counterpart for bin packing, which is the reason we were able to test only instances of shorter list sizes here. The maximum ratio of 5.0 in the second row of the table rules out a guaranteed approximation ratio less than 5.0 for the FF heuristic.

Also observe the steep drop in the percentage of ones (the proportion of the instances where the FF heuristic produces a solution as good as that of the exact algorithm) with increase in list size — there is also a sharp drop compared with the values in column 6 of Table 1 for bin packing. This steep drop, in our opinion, also supports the prediction in Conjecture 7.

3.3 Bin covering: variable bin size

Problem statement: VSBC$_G$ (variable sized bin covering problem, generic version): We are given a collection $\mathcal{B}$ of distinct bin sizes s_1 through s_k, $s_1 > s_2 > \cdots > s_k$, and a list $L = (i : 1 \leq i \leq n)$ of items,

with size $a_i \in (0,1]$. It is assumed that $s_1 = 1$. A feasible bin cover for L and $\mathcal{B}$ is an assignment of the items in L to a set of bins with sizes in $\mathcal{B}$. The goal is to find a feasible bin cover that maximizes the sum of the sizes of the covered bins (and this sum is less than or equal to the sum of the sizes of the items they hold). The LIB constraint is *not* enforced.

Woeginger and Zhang [10] show that VSBC$_G$ is NP-hard and provide heuristics for the problem. Their heuristic is reproduced below for reference.

3.3.1 Woeginger-Zhang heuristic for VSBC$_G$

Let K be the number of bin sizes in $\mathcal{B}$ that are strictly greater than 0.5, and define

$$\begin{aligned} &q_j = s_j/s_{j+1}, \quad \text{where} \quad s_j,\ s_{j+1} \in \mathcal{B} \text{ and } 1 \le j \le k-1, \\ &Q(\mathcal{B}) = \{q_j : 1 \le j \le k-1\}. \end{aligned} \tag{15}$$

Also define

$$q(\mathcal{B}) = \max_j \{q_j : q_j \in Q(\mathcal{B})\}. \tag{16}$$

Let m be the smallest non-negative integer to satisfy $2^{-m-1}q \le q-1$. A partition of $(0,1]$ into $k(m+1)$ intervals $\mathcal{I}_{j,l}$ with $1 \le j \le k$ and $0 \le l \le m$ is defined as:

- for $1 \le j \le k-1$ and $0 \le l \le m$, let $\mathcal{I}_{j,l} = (b_{j+1}/2^l, b_j/2^l]$;

- for $0 \leq l \leq m-1$, let $\mathcal{I}_{k,l} = (b_1/2^{l+1}, b_k/2^l]$;
- $\mathcal{I}_{k,m} = (0, b_k/2^m]$.

The items in the first type of interval above are placed in bins of size b_{j+1}. Items in the second and third interval types are placed in bins of size $b_1 = 1$. An arriving item is placed in a used uncovered bin corresponding to its interval. If no such bin is available, the item is placed in a new, unused bin.

3.4 VSBC: allowing placement changes

Problem: The problem considered here, VSBC$_A$, is similar to VSBC$_G$, except that the LIB constraint is enforced, and although an arriving item i can be placed only in the bin to which it is assigned, items in that bin can be rearranged so that item i is placed in its correct position in the bin as determined by its length.

The approximation algorithm from [10] presented above only needs to be slightly modified, by adding an *insertion sort* routine. When an item i (length a_i) arrives, it is irrevocably assigned to bin b_j. Suppose the number of items in b_j including i is p, and let $l+1$ be the position of i from the top of b_j according to its length (or, the number of items in b_j that are shorter than i is l). Hence l items need to be taken out of the bin and put back, to

insert item(i) in its correct position in the bin. Hence the additional time spent during each arrival from L is $2l$.

Now $l \leq n$, where $n = |L|$, and thus the additional time at each arrival is $2l \leq 2n$. There are n arrivals in all, and this implies an additional time of $\mathcal{O}(n^2)$, polynomial in the size of L. The approximation ratio obtained by the heuristic in [10] is unaffected by the modification proposed here.

For any given instance $(L, \mathcal{B})$, since our modification to the heuristic in [10] causes

1. an additional running time of $\mathcal{O}(n^2)$ that is polynomial in the size of L, and
2. no change to the approximation ratio obtained,

the results of this section are summarized as follows.

Remark 8 *With the modifications proposed above to the approximation algorithm in [10],* VSBC_A *becomes as approximable as* VSBC_G.

It is easy to see that the sorting algorithm proposed in this section can be applied to any offline problem in one-dimensional bin packing and bin covering. The items can be required to be placed in bins in the order of their lengths, and results similar to Lemma 8 will still apply. The additional

requirement of lengthwise placement in bins does not degrade the approximability of an offline problem. This can be generalized as follows.

Given an offline problem P in one-dimensional bin packing or bin covering, and an approximation heuristic A for P, let P_l be the problem modified from P by requiring that the items in each bin be placed in the order of their lengths. Let A_s be the heuristic obtained by applying sorting to the items in each bin at the end of an application of A. For any given instance of an item list L and bin sizes $\mathcal{B}$, the approximation ratios obtained by applying A to P and A_s to P_l are the same.

Remark 9 *Problem P_l is as approximable as P.*

3.5 VSBC: disallowing placement changes

Problem: This problem (VSBC_P) is similar to VSBC_G defined in Section 3.3, except that the LIB constraint is enforced. Unlike VSBC_A, rearrangement of items within bins is not permitted. The sequence in which the items are placed in a bin is never changed.

An arriving item i of size a_i must be placed at the top of its assigned bin b_j unless

- length is an issue (if i is longer than item l previously at the top of b_j before i's arrival); or

- bin b_j is already covered,

in which two cases, i is assigned to an unused bin. The problem is further simplified due to Conjecture 7 (see Case 2 below).

3.5.1 Approximation algorithm for VSBC$_P$

We now give an approximation algorithm for this version of VSBC. When an item i of size $a_i \in L$ arrives, it is assigned to a bin as follows.

Case 1: (item i is at least as long as the shortest bin s_k): Assign i to a new (unused) bin whose size is the maximum among all bin sizes in $\mathcal{B}$ that are at most as long as i. In other words, assign i to an unused bin of size s_x, where

$$s_x = \max_j \{s_j \in \mathcal{B} : s_j \leq a_i\}\,. \tag{17}$$

Case 2: (item i is shorter than the shortest bin s_k): This case is almost the same as bin covering with uniform bin size (Section 3.2), except that all the item sizes and the (uniform) bin size are multiplied by a factor of s_k. In light of Conjecture 7 and the accompanying Theorem 6, we simplify the problem by assuming that item sizes in this case are discrete. In particular, let the item sizes be elements of a set $C = \{c_1, \ldots, c_m\}$, with $s_k > c_1 > \cdots > c_m$. Only bins of size s_k are used here.

3.5.2 Algorithm analysis

Let

$$\begin{aligned} &r_i = (s_k + c_i)/s_k\,, \quad \text{where} \quad c_i \in C \text{ and } 1 \le i \le m\,,\\ &R = \{r_i : 1 \le i \le m\}\,,\\ &q_j = s_j/s_{j+1}\,, \quad \text{where} \quad s_j,\ s_{j+1} \in \mathcal{B} \text{ and } 1 \le j \le k-1\,,\\ &Q = \{q_j : 1 \le j \le k-1\}\,. \end{aligned} \tag{18}$$

Define

$$q = \max_j\{q_j : q_j \in Q\}\,, \quad r = \max_i\{r_i : r_i \in R\}\,. \tag{19}$$

Observe that q (r) is at least one.

Case 1: (item i is at least as long as the shortest bin s_k): Suppose i is placed in bin s_x. From (18) and (19),

$$\frac{a_i}{s_x} \;\le\; \frac{s_{x-1}}{s_x} \;=\; q_{x-1} \;\le\; q\,. \tag{20}$$

From this, we get

$$a_i \le q s_x\,. \tag{21}$$

The contents of bins in Case 1, $C^{(1)}$, can be broken down into $C_C^{(1)}$ and $C_O^{(1)}$:

$$\sum_i a_i = C^{(1)} = C_C^{(1)} + C_O^{(1)} = C_C^{(1)}\,, \tag{22}$$

where $C_C^{(1)}$ and $C_O^{(1)}$ represent the sum of the contents of covered and uncovered bins respectively in Case 1. Observe that the assignment of item i to b_x covers b_x. Hence there can be no uncovered bins in this case (thus $C_O^{(1)}$ is zero in (22)).

If (21) is summed up over all items i arriving in Case 1, we get, combining with (22),

$$C_C^{(1)} = C^{(1)} = \sum_i a_i \leq q \sum s_x = qB^{(1)}\,, \tag{23}$$

where $B^{(1)}$ is the sum of the sizes of all bins used in Case 1.

Case 2: (a_i is shorter than the shortest bin s_k): Let $\beta(c_i)$ be the set of bins that hold items of length c_i. At any time, the number of uncovered bins in $\beta(c_i)$ can be at most one, since an uncovered bin should be covered before we start filling an unused bin.

Let N_i be the number of covered bins in $\beta(c_i)$. The `totalSize` of a covered bin in $\beta(c_i)$ is less than $s_k + c_i$, since at most one item can be protruding out of a bin. Thus $C_C^{(2)}(i)$, the sum of the contents of covered bins in $\beta(c_i)$ (in Case 2), is bounded by

$$C_C^{(2)}(i) \leq (N_i)(s_k + c_i) = N_i s_k r_i = B_i^{(2)} r_i \leq B_i^{(2)} r\,, \tag{24}$$

where $B_i^{(2)}$ is the sum of the sizes of all covered bins that hold items of size c_i. The last inequality in (24) follows from (19).

It follows from the above that $C_C^{(2)}(i) \leq rB_i^{(2)}$. When we take a summation of this over all sizes c_1 through c_m, we get

$$C_C^{(2)} = \sum_i C_C^{(2)}(i) \leq \sum_i rB_i^{(2)} = r\sum_i B_i^{(2)} = rB^{(2)}\,. \tag{25}$$

Since there are m sizes from c_1 through c_m, ms_k is a trivial upper bound on $C_O^{(2)}$, the sum of the sizes of uncovered bins (in Case 2).

Combining the above two upper bounds, we obtain the upper bound on $C^{(2)}$ as

$$C^{(2)} = C_C^{(2)} + C_O^{(2)} \leq rB^{(2)} + ms_k\,. \tag{26}$$

3.5.3 Approximability

As mentioned in [10], $C_C + C_O$ is a trivial upper bound on $\textsc{opt}(L, \mathcal{B})$. We conclude that

$$\begin{aligned} \textsc{opt}(L,\mathcal{B}) &\leq (C_C) + (C_O) \\ &= (C_C^{(1)} + C_C^{(2)}) + (C_O^{(1)} + C_O^{(2)}) \\ &= C_C^{(1)} + C_O^{(1)} + C_C^{(2)} + C_O^{(2)} \\ &= C^{(1)} + C^{(2)}\,. \end{aligned} \tag{27}$$

Let $t = \max(q, r)$. From (23) and (26), it follows that

$$\begin{aligned} \textsc{opt}(L,\mathcal{B}) &\leq C^{(1)} + C^{(2)} \\ &\leq qB^{(1)} + rB^{(2)} + ms_k \\ &\leq tB^{(1)} + tB^{(2)} + ms_k\,. \end{aligned} \tag{28}$$

Now $A(L, \mathcal{B})$, the value of the solution obtained by the approximation algorithm, is $B^{(1)} + B^{(2)}$. Thus, from (28),

$$R_{A,\mathcal{B}} = \frac{\text{OPT}(L, \mathcal{B})}{\text{A}(L, \mathcal{B})} = \frac{t(B^{(1)} + B^{(2)}) + ms_k}{B^{(1)} + B^{(2)}}, \tag{29}$$

where $R_{A,\mathcal{B}}$ is the approximation ratio for algorithm A and given bin sizes $\mathcal{B}$. From the asymptotic point of view, ms_k can be treated as a constant.

Theorem 10 *For* VSBC$_P$, $R^{\infty}_{A,\mathcal{B}}$, *the asymptotic approximation ratio, is upper bounded by* $t = max(q, r)$, *where* q *and* r *are defined in (18) and (19).*

4 Discussion and remarks

In this paper, we have considered problems in one-dimensional bin packing and bin covering with the additional constraint that the items be placed in bins in the order of their lengths. The longest item is required to be placed at the bottom (the LIB constraint). All problems considered are NP-hard.

In LIB bin packing with uniform sized bins, we showed that the Next Fit algorithm with an input of n items cannot have an R_A better than $\theta(n)$. We presented a modified version of the First Fit algorithm for this problem. We proved an upper bound of 2 for two special cases of the problem, namely,

if item sizes are monotonic non-decreasing or non-increasing (MND or MNI). Since the general case remains inconclusive as to whether there is a constant upper bound on R_{FF}, we have provided results from computational testing (Section 2.6). In about 20,000 instances of computation for this problem, the maximum R_A was only 1.5, well within the bound of two foreseen in Conjecture 3.

Interestingly, in the testing of both bin packing and bin covering problems, as the problem size increases, the maximum ratio decreases, the average ratio increases, and the percentage of instances where the heuristic produces a solution as good as the optimal solution drops (the drop is steep in bin covering, see Section 3.2.3).

We have provided a similar First Fit heuristic for uniform sized bin covering with LIB. We showed that its worst case approximation ratio is $\theta(n)$. In Theorem 6, we showed another non-approximability result, namely that there can be no heuristic for this problem that can guarantee an upper bound less than two unless P = NP. Computational results for this problem are provided in Section 3.2.3, and are similar to those given in Section 2.6. The worst value of R_A obtained in these experiments was 5.0.

In light of the non-approximability predicted in Conjecture 7, we assume a simplified problem in the case of online, LIB (longest item in the bottom) VSBC — we assume that item sizes are discrete (as opposed to continuous item sizes in the case of uniform sized bins) for sizes less than the smallest bin size. The guarantee on R_A achieved here in Theorem 10 is $\max(q, r)$,

where q and r are defined in (18) and (19).

The following observations are made without a formal proof regarding the online LIB bin covering problem:

- What if there exist items in L with size greater than one (the size of the largest bin)? The best we can do here is to place each item in its own bin of size $s_1 = 1$. It is easy to see that no algorithm can do better than this for items of such size. If the number of such items is $|L_1|$, so is the number of bins used. The total size of these bins is $|L_1|$, and this term is added to the numerator and denominator of (29) in computing the approximation ratio. Therefore, if items in L larger than s_1 are too numerous compared with other items, $R_{A,\mathcal{B}}$ in (29) will approach one, implying optimality of the approximation algorithm.

- Worst Case Input: Suppose the items arrive in the order of strictly increasing length, for lengths greater than as well as less than s_k. In such a case, the items have to be placed one item per bin. Items of length $< s_k$ will be placed in bins of size s_k. For items of length $\geq s_k$, each of these will be placed in an unused bin, and to obtain the best possible value for the objective function, observe that the item should be placed as per (17). Thus our algorithm is as good as any other algorithm for this input.

5 Further research

Approximation results need to be uncovered for online versions of all problems in bin packing and bin covering where it is required that items be placed in bins in the order of their lengths (or weights). Item lengths can be discrete or continuous. Bin sizes can be uniform, variable, discrete or continuous. The algorithms proposed in this paper for variable-sized bin covering should be computationally tested for their performance. Conjecture 7 is an open problem that needs resolution — the lower and upper bounding techniques developed in the branch and bound algorithms could prove valuable in this context.

Acknowledgements: The author benefited from useful discussions with Marcsimon Visser of the University of Twente. Thanks to Janos Csirik for making available his papers. We gratefully acknowledge support from the Sir Ross and Sir Keith Smith Foundation in Adelaide, South Australia.

References

[1] S.F. Assmann. *Problems in Discrete Applied Mathematics.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1983. E203

[2] S.F. Assmann, D.S. Johnson, D.J. Kleitman and J.Y.-T. Leung. "On a Dual Version of the One-dimensional Bin Packing". *Journal of Algorithms*, 5:502–525, 1984. E203

[3] E.G. Coffman, M.R. Garey and D.S. Johnson. "Bin Packing Approximation Algorithms: A Survey". In D. Hochbaum, editor, *Approximation Algorithms for* NP*-Hard Problems*, pages 46–93. PWS Publishing Company, Boston, MA, 1996. E189

[4] E.G. Coffman, J. Csirik and G.J. Woeginger. "Approximate Solutions to Bin Packing Problems". Technical Report Woe-29, Institut fur Mathematik B, TU Graz, Steyrergasse 30, A-8010 Graz, Austria, February 1999. E189

[5] J. Csirik and V. Totik. "On-line Algorithms for a Dual Version of Bin Packing". *Discrete Applied Mathematics*, 21:163–167, 1988. E203, E232

[6] J. Csirik, J.B. Frenk, M. Labbe and S. Zhang. "Two Simple Algorithms for Bin Covering". *Acta Cybernetica*, 14(1):13–25, 1999. E203

[7] J. Csirik and J.B. Frenk. "A Dual Version of Bin Packing". *Algorithms Rev.*, 1:87–95, 1990. E203

[8] J. Csirik and G.J. Woeginger. "On-line Packing and Covering Problems". Technical Report SFB-83, Institut fur Mathematik B, TU Graz, Steyrergasse 30, A-8010 Graz, Austria, 1996. E204

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*. Freeman (New York), 1979. E192

[10] G.J. Woeginger and G. Zhang. "Optimal On-Line Algorithms For Variable-Sized Bin Covering". *Operations Research Letters*, 25:47–50, 1999. E203, E204, E205, E218, E219, E220, E225

A Erratum: Correction to the proof leading to Theorem 6

In Section 3.2.2, the lower bound mentioned in Theorem 6 should be 1.5, not 2. The error is due to the fact that only extreme values of m are considered in Cases 1b and 2 (pages E211–E213).

Heuristic A could adopt different strategies. Here is one strategy that guarantees an upper bound of 1.5 on the approximation ratio.

Double stack two-thirds of the items in L_1 and single stack the remaining one-third. Again, let i be the number of items in L_1. Observe that at most $i/3$ bins can be covered by placing the L_2 items on top of L_1 items. The number of (double-stacked) bins covered by L_1 items alone is equal to $i/3$. There are several cases here, depending on the value of j (the number of items in L_2):

$j \geq i$: Optimal algorithm will cover i bins. The heuristic will cover $i/3 + i/3 = 2i/3$ bins. Thus the approximation ratio $= 1.5$.

$i/3 \leq j < i$: Optimal algorithm will cover $j + (i-j)/2 = (i+j)/2$ bins. The heuristic will cover $i/3 + i/3 = 2i/3$ bins. Thus the approximation ratio $= 3(i+j)/4i = 0.75(1 + j/i) < 0.75(2) = 1.5$.

$j < i/3$: Optimal algorithm will cover $j + (i-j)/2 = (i+j)/2$ bins. The heuristic will cover $i/3 + j = (i+3j)/3$ bins. Thus the approximation ratio $= (1.5)(i+j)/(i+3j) \leq 1.5$.

The guaranteed approximation ratio is the maximum among the three cases described above, and hence equal to 1.5 as now appears in Theorem 6.

Discussion: The lower bound on the guaranteed approximation ratio for this problem, uniform sized bin covering with LIB, remains at two, since Csirik and Totik [5] have proven a lower bound of two for the simpler problem (the non-LIB case). Their proof applies to our problem as well.

Acknowledgement: The author thanks Kevin White for bringing the error to his attention.