ONLINE LIB PROBLEMS: HEURISTICS FOR BIN COVERING AND LOWER BOUNDS FOR BIN PACKING

L. FINLAY\textsuperscript{1} AND P. MANYEM\textsuperscript{2}

Abstract. We consider the NP Hard problems of online Bin Covering and Packing 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 — we call such a version, the LIB version of problems. Bin sizes can be uniform or variable. We look at computational studies for both the Best Fit and Harmonic Fit algorithms for uniform sized bin covering. The Best Fit heuristic for this version of the problem is introduced here. The approximation ratios obtained were well within the theoretical upper bounds. For variable sized bin covering, a more thorough analysis revealed definite trends in the maximum and average approximation ratios. Finally, we prove that for online LIB bin packing with uniform size bins, no heuristic can guarantee an approximation ratio better than 1.76 under the online model considered.

Introduction

Bin covering definition: In the classical one-dimensional Bin Covering 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 $a_i \in (0, 1]$. A bin is said to be covered if the sum of the item sizes in the bin adds up to at least one. The problem is to pack these $|L| = n$ items into bins such that the number of covered bins is maximised.

The solution is feasible even if the sum of item sizes in a bin is greater than one. In a covered bin, it is indeed possible for an item to be protruding out of the bin. There could also be items that are completely outside the bin, stacked on top

\textsuperscript{1} Centre for Industrial and Applied Mathematics (CIAM), University of South Australia, Mawson Lakes, SA 5095, Australia; e-mail: luke.finlay@unisa.edu.au
\textsuperscript{2} Centre for Informatics and Applied Optimisation (CIAO), University of Ballarat, Mount Helen, VIC 3350, Australia; e-mail: p.manyem@ballarat.edu.au

\textcopyright{} EDP Sciences 2001
of other items, and yet belonging to the bin — this is still feasible — however, this only pushes the solution farther away from optimality! The objective is to find a feasible solution that maximises the sum of the sizes of the covered bins — in the case of unit-sized bins, this is the same as maximising the number of covered bins.

Offline versus online bin covering: If the sizes of all items in $L$ are known in advance, this is known as offline bin covering. In the online version of bin covering, 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 length $a_{i+1}$ of the next item becomes known only after item $i$ has been assigned to its bin.

The online condition can be formulated as follows: 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,

$$[i \text{ is below } j \text{ in a used bin}] \implies [i < j] \quad (1)$$

In all versions of Bin Packing and Bin Covering, 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.

LIB Versions. The online versions of Bin Packing and Covering considered in this paper impose 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 can call this the LIB version, for Longest Item at the Bottom. The LIB constraint can be defined as

$$[i \text{ is below } j \text{ in a used bin}] \implies [a_i \geq a_j] \quad (2)$$

The dual of Bin Covering is Bin Packing, where the item sizes in a bin should total up to at most the bin size. The online LIB variation of Bin Packing is treated in Section 3.

Literature on Bin Covering. Some of the earliest works to appear in Bin Covering were by Assmann [1] and Assmann et al [2]. In [2], the authors provide polynomial time heuristics with an AAR (defined in Sec. 1.1) of $4/3$ for the offline problem and 2 for the online problem when all bins are of unit size. Csirik and Totik [9] show that there can be no polynomial time heuristic that guarantees an AAR better than 2 for online problems with unit-sized bins. Csirik et al provide two algorithms for offline bin covering in [6]. Woeginger and Zhang [15] provide a polynomial time heuristic for the online version with variable sized bins. For a survey of bin covering problems, see Csirik and Frenk [5]. Csirik et al [7] provide a polynomial time approximation scheme for the general version of Bin Covering, as well as algorithms that have bounded worst-case behaviour for instances with discrete item sizes — these algorithms are based on the Sum of Squares algorithm [8] for Bin Packing.

Online LIB Covering: Manyem [11] provides Next Fit (NF) and First Fit (FF) heuristics for the LIB version, and shows that both heuristics cannot guarantee an AAR better than $\Theta(n)$, where $n = |L|$ is the number of items in the input, even for
uniform-sized bins. NF and FF belong to the class of Any Fit algorithms, whereas HF is a bounded space algorithm, as explained in Section 1.4. In [13], Manyem et al provide a harmonic fit (HF) algorithm to the LIB version of the problem, and extend the $\Theta(n)$ negative result to the same. This paper introduces the Best Fit (BF) heuristic for this problem.


Applications

Bin Packing and Covering theory does help solve practical industry based problems such as assigning semiconductor wafer lots to customer orders [3]. 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.

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 [15].

Organisation of The Paper

The entire paper deals only with online, LIB problems. Bin Covering is a dual version of Bin Packing [2]. Bin sizes may be uniform or variable. Tables 1 and 2 summarise the notation and the acronyms used. The focus of computational testing in this paper is on approximation ratios, not on running times.

Section 1 is about uniform-sized Bin Covering (USBC). Sections 1.2 and 1.3 discuss computational results from implementing First Fit (FF) and Best Fit (BF) algorithms for USBC. The FF heuristic was introduced in [11]. This paper introduces the BF heuristic.

In Section 1.4, we present the performance results of the Harmonic Fit (HF) heuristic.

Section 2 studies the performance of an adaptation of the Woeginger-Zhang heuristic [15] for two different cases: (a) uniformly spaced bin sizes $B = \{1.0, .8, .6, .4 \text{ and } .2\}$, and (b) non-uniform bin sizes $B = \{1.0 \text{ and } 0.2\}$.

In Section 3, we prove a lower bound on the guaranteed approximation ratio for the uniform-sized bin packing problem.
Table 1. Notation (in alphabetical order)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>Size of item $i$</td>
</tr>
<tr>
<td>$b_j$</td>
<td>Bin number $j$</td>
</tr>
<tr>
<td>$\mathcal{B}$</td>
<td>Set of used bins</td>
</tr>
<tr>
<td>$B$</td>
<td>Set of available bin sizes, $s_1$ through $s_K$</td>
</tr>
<tr>
<td>$\text{diffBinSize}$</td>
<td>Difference between successive bin sizes in VSBC</td>
</tr>
<tr>
<td>$i$</td>
<td>Index for an item (usually)</td>
</tr>
<tr>
<td>$I_k$</td>
<td>Interval $k$, equal to $[(k+1)^{-1}, k^{-1}]$ in HF</td>
</tr>
<tr>
<td>$j$</td>
<td>Index for a bin (usually)</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of available bin sizes (cardinality of $\mathcal{B}$)</td>
</tr>
<tr>
<td>$L$</td>
<td>Input list of items, in a given sequence</td>
</tr>
<tr>
<td>$L_k$</td>
<td>Sublist of $L$ with item sizes in interval $I_k$ (order of item arrivals in $L_k$ is the same as in $L$)</td>
</tr>
<tr>
<td>$M$</td>
<td>Number of intervals into which $(0, 1]$ is divided (in HF algorithm)</td>
</tr>
<tr>
<td>$n$</td>
<td>Cardinality of list $L$</td>
</tr>
<tr>
<td>$p$</td>
<td>Percentage of times the heuristic and optimal solutions have equal value</td>
</tr>
<tr>
<td>$P$</td>
<td>$\frac{\text{Sum of the sizes of used bins}}{5}$ (in variable size bin cover)</td>
</tr>
<tr>
<td>$R_A$</td>
<td>Worst case approximation ratio for algorithm $A$</td>
</tr>
<tr>
<td>$R_A^\infty$</td>
<td>Worst case asymptotic approximation ratio for algorithm $A$</td>
</tr>
<tr>
<td>$s_1(s_K)$</td>
<td>Size of the largest (smallest) available bin size</td>
</tr>
<tr>
<td>$\text{topSize}(b_j)$</td>
<td>Size of the item at the top of bin $b_j$</td>
</tr>
<tr>
<td>$\text{totalSize}(b_j)$</td>
<td>Sum of the sizes of the items in bin $b_j$</td>
</tr>
<tr>
<td>$</td>
<td>(a,b)</td>
</tr>
</tbody>
</table>

Section 3 Notation Below:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>Number of double-stacked $L_1$ items</td>
</tr>
<tr>
<td>$m_i$</td>
<td>Maximum number of items from list $L_i$ that can be placed in a unit size bin</td>
</tr>
<tr>
<td>$m_{1i}$</td>
<td>Maximum number of $L_i$ ($i \geq 2$) items that can be placed on top of an $L_1$ item</td>
</tr>
<tr>
<td>$s_1$</td>
<td>Number of single-stacked $L_1$ items</td>
</tr>
<tr>
<td>$s_2$</td>
<td>Max. no. of $L_2$ items that could be placed on top of single-stacked $L_1$ items (actual no. could be less than $s_2$)</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Number of $L_2$ items that are multi-stacked</td>
</tr>
</tbody>
</table>

1. **Uniform Sized Bin Covering**

**Problem Statement:** Online USBC with LIB constraint. Given an infinite supply of unit sized bins and a list $L$ with $|L| = n$ items, each of size $(0, 1]$. Items should be placed in bins while maintaining the online and LIB constraints (1) and (2). Any number of items can be placed in a bin, regardless of whether the bin is overflowing. The objective is to maximise the number of covered bins.
1.1. Approximation Ratios

Given an instance (a list $L$) of Bin Covering, let $\text{OPT}(L)$ and $A(L)$ be the solution values obtained by the exact (or optimal) and approximation algorithms respectively. We define the asymptotic approximation ratio (AAR), $R_A^\infty$ for approximation algorithm $A$ as

$$R_A^\infty = \lim_{s \to \infty} \inf_L \left\{ R_A^{(L)}, \, \text{OPT}(L) \geq s \right\}, \text{ where } R_A^{(L)} = \frac{\text{OPT}(L)}{A(L)}$$

(3)

When bin sizes are variable, this generalizes to

$$R_{A,B}^\infty = \lim_{s \to \infty} \inf_L \left\{ R_{A,B}^{(L)}, \, \text{OPT}(L,B) \geq s \right\}, \text{ where } R_{A,B}^{(L)} = \frac{\text{OPT}(L,B)}{A(L,B)}$$

(4)

where $B$ is the collection of bin sizes. For a class of inputs $C$, $R_A^C$ and $R_{A,B}^C$ are similarly defined. Observe that $1 \leq R_A^\infty, R_{A,B}^\infty \leq \infty$. The lower these ratios, the better the approximation algorithm. Note that the above ratios are defined over all instances of the corresponding problems. Hence these are worst case ratios — worst among all the instances.

Note that $\text{OPT}(L)$ refers to the solution obtained by an optimal algorithm that (1) knows the entire input list and the sequence of items in advance, and (2) obeys the online and LIB constraints (1) and (2). Usually this solution is the same as that of an optimal offline algorithm, but this is not necessarily true when the LIB constraint comes into play. For example, if $a < b$ — that is, item $a$ arrives before item $b$ — and size$(a) < \text{size}(b)$, an optimal offline algorithm could place both items in the same bin with $a$ above $b$, but the optimal online algorithm that we use to produce $\text{OPT}$ can never do this, because this would violate the online constraint (1).
1.2. **Best Fit Algorithm for Bin Covering**

**First Fit (FF).** The FF algorithm [11] for USBC with online and LIB constraints has been proven to have an upper bound AAR of $\Theta(n)$. 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 (1) $a_i \leq topSize(j)$, and, (2) $totalSize(j) < 1$. FF places item $i$ in the first such bin $b_j$ that satisfies both the conditions above and updates $topSize(j)$ as well as $totalSize(j)$ — note that after such a placement, $totalSize(j)$ could be greater than one. If no such bin among $b_1$ through $b_m$ satisfies these conditions, FF opens a new bin $b_{m+1}$ in which to place $i$.

**Best Fit (BF).** The BF algorithm behaves similar to FF, but for the following differences:

- If there exists at least one used uncovered bin $b_j$, $1 \leq j \leq m$, such that
  1. $a_i \leq topSize(j)$,
  2. $totalSize(j) < 1$, and
  3. placing $i$ in $b_j$ causes overflow of the bin (the bin becomes covered),
  then $i$ is placed in that used bin for which the overflow is the least.
- If such a used bin as described above is unavailable, but there is a used bin that meets conditions (1) and (2) above, then item $i$ is placed in that used bin for which, after placing $i$, $totalSize(j)$ is the greatest.

BF is *greedier* than FF. It still cannot guarantee an AAR better than $\Theta(n)$ — however, it has a better average ratio (see Tables 3-4).

**Algorithm 1 (ALG1). Best Fit (online LIB Bin Covering version).**

*Given:* Items $1 \cdots n$ with sizes $a_1 \cdots a_n$, $0 < a_i \leq 1$ for $1 \leq i \leq n$.

*Running Time:* $O(n^2)$.

```plaintext
1 nBin (number of bins used) = 0;
2 for (item = 1 to n) do
3     bin = 1;
4     bestBin = 0;
5     firstBin = 0;
6     bestWaste = 1;
7     while (bin <= nBin) do
8         if (topSize[bin] >= size[item] AND totalSize[bin] >= 1) then
9             if (firstBin == 0) then
10                 firstBin = bin;
11         end if
12         if (totalSize[bin] + size[item] >= 1 AND
13             totalSize[bin] + size[item] - 1 < bestWaste) then
14             bestBin = bin;
15             bestWaste = totalSize[bin] + size[item] - 1;
16         end if
17     end while
18 end for
```
end if (from line 12)
end if (from line 8)
b \text{bin} = \text{bin} + 1;
end while (from line 7)
if (\text{bestBin} \neq 0) then
\begin{align*}
&\text{place item in bestBin;} \\
&\text{update topSize[bin] and totalSize[bin]};
\end{align*}
elseIf (\text{firstBin} \neq 0) then
\begin{align*}
&\text{place item in firstBin;} \\
&\text{update topSize[bin] and totalSize[bin]};
\end{align*}
else (item not placed in any previous bin)
\begin{align*}
&n \text{Bin} = n \text{Bin} + 1; \\
&(\text{new, fresh, unused bin}) \\
&\text{place item in nBin;} \\
&\text{topSize[nBin]} = \text{size[item]}; \\
&\text{totalSize[nBin]} = \text{size[item]};
\end{align*}
end if (from line 19)
end for

1.3. Computational Comparison of BF and FF Algorithms

Again, we emphasise that the focus of computational testing in this paper is on approximation ratios, not on running times. When comparing the Best Fit (BF) and First Fit (FF) algorithms, the maximum ratios are likely to be similar, since both belong to the Any Fit class of heuristics, but the average ratio could be better for BF. Item sizes were generated using a uniform distribution in the interval (0,1).

\begin{table}[!h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
List Size & Maximum Ratio & Average Ratio & No. of Runs & \% of Ones \\
\hline
10 & 4.0 & 1.414 & 5000 & 37.42 \\
15 & 5.0 & 1.425 & 5000 & 12.26 \\
20 & 3.0 & 1.417 & 1000 & 3.5 \\
\hline
\end{tabular}
\caption{Bin Covering First Fit Algorithm [11]}
\end{table}

It can be observed from Tables 3 and 4 that the average ratios for the Best Fit algorithm are better. The BF results have more number of items, hence the actual comparison with FF can only be done on the list sizes of 10, 15 and 20. The time taken for computation for \( n \geq 20 \) was very high. Hence a distributed architecture was used, with 4 to 8 computers working on the problem at any given time. The maximum ratio for BF almost looks quadratic with the maximum occurring at an \(|L| = n \) value between 15 and 17 items (except for the anomaly that occurs at \( n = 21 \) producing a maximum of 5). The BF maximum ratios are very similar to those of FF, as anticipated. The number of runs for 22 items was only 2000 due to the high amount of computation time involved.
### Table 4. Bin Covering Best Fit Algorithm

<table>
<thead>
<tr>
<th>List Size</th>
<th>Maximum Ratio</th>
<th>Average Ratio</th>
<th>No. of Runs</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>1.106</td>
<td>5000</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1.170</td>
<td>5000</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1.206</td>
<td>5000</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1.240</td>
<td>5000</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1.276</td>
<td>5000</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>1.282</td>
<td>5000</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>1.298</td>
<td>5000</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>1.316</td>
<td>5000</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>1.322</td>
<td>5000</td>
<td>1.5 secs</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>1.333</td>
<td>5000</td>
<td>9.9 min</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>1.337</td>
<td>5000</td>
<td>1 hour</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>1.343</td>
<td>5000</td>
<td>2.5 hours</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>1.343</td>
<td>5000</td>
<td>8.2 hours</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>1.342</td>
<td>5000</td>
<td>1.49 days</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>1.347</td>
<td>5000</td>
<td>4.82 days</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>1.349</td>
<td>4000</td>
<td>20 days</td>
</tr>
<tr>
<td>21</td>
<td>5</td>
<td>1.350</td>
<td>4000</td>
<td>8.2 hours</td>
</tr>
<tr>
<td>22</td>
<td>3.5</td>
<td>1.328</td>
<td>2000</td>
<td>47 days</td>
</tr>
</tbody>
</table>

In Figure 1, the average ratio for the BF algorithm seems to be asymptotically approaching 1.4, which is roughly the average ratio for FF (Table 3). Of course, list sizes larger than 22 would have to be experimented with to see if this trend continues, and more runs with a list size of 22 items would need to be carried out.

#### 1.4. Harmonic Fit Algorithm

The best fit and first fit algorithms fall into the *any fit* category. That is, any item can be placed in any bin. The harmonic fit (HF) algorithm places items into categories of bins. Each bin is still of unit size, but the bin can only accept items in a specific size range or size interval — hence HF is known as a *bounded space* heuristic. Items in different intervals cannot be mixed in the same bin. The number of intervals, $M$, is finite.

**The HF heuristic.** The HF algorithm for online USBC with the LIB constraint is as follows [13]:

- Divide the unit interval $(0,1]$ into $M$ intervals such that $(0,1] = \bigcup_{k=1}^{M} I_k$, where $I_k = \left(\frac{1}{k+1}, \frac{1}{k}\right]$, $1 \leq k \leq M - 1$ and $I_M = (0, \frac{1}{M}]$, where $M$ is a positive integer. That is, the breakpoints are defined as $\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots, \frac{1}{M}\}$.
- Given an input list $L = \{i : 1 \leq i \leq n\}$ of items with sizes $\{a_i\}$, divide $L$ into $M$ sublists $\{L_k \mid 1 \leq k \leq M\}$ based on size, such that an item
$i \in L_k$ if and only if its size $a_i$ falls in the corresponding size interval $I_k$. The correct sequence of arrivals (as in $L$) is also maintained within each sublist $L_k$. Each $L_k$ is treated henceforth as independent input.

• For each $L_k$, place items in bins using the First Fit (FF) heuristic.

It has been proven [13] that the HF algorithm has an upper bound for the AAR of $\Theta(n)$. Testing of this algorithm was taking a long time to compute, so parallel computing was used with a mix of Windows and Red Hat Linux machines. The parallel part of the code was written in Java and the code that calculates the approximation ratio was written in C.

The final results are shown below in Table 5 with only the Best number of intervals shown. The ratios obtained vary with the number of intervals $M$ chosen for the HF heuristic. The Best number of intervals is given by the one with the lowest maximum ratio and is defined as $M_b$ on the table. In most cases this was three. Only 2763 runs were done due to the lengthy computation times involved. The time taken (column 4 of the table) was the amount of time a single computer working on the problem would have taken if it had worked on all the runs of the problem. The % of Ones, ($p$ in the last column) is the percentage of runs where

![Figure 1. Average Ratios for BF Algorithm](image-url)
the heuristic performed optimally. The *Max Ratio* column defines the maximum ratio that was attained by the heuristic.

<table>
<thead>
<tr>
<th>List size (n)</th>
<th>Best No. of intervals (M_b)</th>
<th>No. of Runs</th>
<th>Run Time</th>
<th>Max. Ratio</th>
<th>Ave. Ratio</th>
<th>% of Ones (p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>2763</td>
<td>14 sec</td>
<td>3</td>
<td>1.503</td>
<td>53.963</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2763</td>
<td>13 sec</td>
<td>4</td>
<td>1.629</td>
<td>42.816</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2763</td>
<td>17 sec</td>
<td>4</td>
<td>1.742</td>
<td>32.863</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2763</td>
<td>7 sec</td>
<td>5</td>
<td>1.830</td>
<td>25.480</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>2763</td>
<td>14 sec</td>
<td>5</td>
<td>1.890</td>
<td>18.567</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>2763</td>
<td>11 sec</td>
<td>5</td>
<td>1.924</td>
<td>12.450</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>2763</td>
<td>10 sec</td>
<td>5</td>
<td>1.934</td>
<td>8.035</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>2763</td>
<td>9 sec</td>
<td>5</td>
<td>1.943</td>
<td>4.958</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>2763</td>
<td>14 sec</td>
<td>6</td>
<td>1.934</td>
<td>3.511</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>2763</td>
<td>3.6 min</td>
<td>6</td>
<td>1.913</td>
<td>2.316</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>2763</td>
<td>5 min</td>
<td>6</td>
<td>1.906</td>
<td>1.484</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>2763</td>
<td>29 min</td>
<td>7</td>
<td>1.899</td>
<td>0.760</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>2763</td>
<td>1.3 hrs</td>
<td>7</td>
<td>1.868</td>
<td>0.507</td>
</tr>
<tr>
<td>18</td>
<td>50</td>
<td>2763</td>
<td>7.6 hrs</td>
<td>7</td>
<td>1.899</td>
<td>0.471</td>
</tr>
<tr>
<td>19</td>
<td>50</td>
<td>2763</td>
<td>30 hrs</td>
<td>6</td>
<td>1.885</td>
<td>0.036</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>2763</td>
<td>5.5 days</td>
<td>7</td>
<td>1.817</td>
<td>0.072</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>2763</td>
<td>16 days</td>
<td>6</td>
<td>1.800</td>
<td>0.072</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>2763</td>
<td>71 days</td>
<td>7</td>
<td>1.799</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5. Bin Covering Harmonic Fit Algorithm

The harmonic fit algorithm does not perform well relatively, as can be observed from Tables 4 and 5. The best fit algorithm performs much better in terms of average as well as maximum ratios. Similar results were obtained in the case of uniform-sized bin packing [13], where for list sizes up to 25, FF (another *any fit* heuristic), performed better than HF. In Figure 2, the average ratio seemed to have a local maximum at a list size of 12. More runs and a larger list size would be needed to verify this.

### 2. Variable Sized Bin Covering

**Problem Statement:** Online VSBC with LIB constraint. Variable sized bin covering (VSBC) involves covering bins of varying sizes. The bin sizes belong to the set $B = \{s_1 = 1 > s_2 > s_3 > \ldots > s_k > 0\}$ and there can be an infinite number of bins of each size. The objective is similar to USBC except that now, our goal is to maximise the sum of the sizes of the covered bins.

**Heuristic for Online VSBC with LIB.** The heuristic being used is the Woeginger-Zhang heuristic [15], adapted to the LIB situation. The adaptation to LIB has appeared in [11], and is reproduced below:
Figure 2. Average Ratios for HF Algorithm

- As each item \( i \) arrives, if \( a_i \geq s_k \), it is placed in the largest possible bin such that the bin is covered.
- If \( a_i < s_k \), then it is placed in any uncovered bin. However...
- If \( a_i < s_k \) and all bins used so far have been covered, then the item is placed in a new bin of size \( s_k \).

It has been proven in [11] that an upper bound \( UB \) of the AAR for the above algorithm is obtained from:

\[
UB = \max\{q, 2\}, \quad \text{where} \quad q = \max\left\{ \frac{s_j}{s_{j+1}} : s_j, s_{j+1} \in B \right\}
\]  

That is, the upper bound for all instances of the problem must be at least two regardless of the bin sizes.

If the bin sizes are equally spaced, then the upper bound will be equal to 2 regardless of the number of bin sizes. If there are \( k \) bin sizes, then \( s_i = \{(k - i + 1)/k, \, 1 \leq i \leq k\} \), and hence the maximum \( q \) will be \( s_{k-1}/s_k = 2 \) regardless of the value of \( k \).
However if \( k \) was infinite, the upper bound would be equal to one because every item would be able to cover a bin of exactly the same size. We restrict ourselves to the non-trivial case where \( \mathcal{B} \) is a finite set.

Therefore the upper bound depends on the set \( \mathcal{B} \). When \( \mathcal{B} = \{1.0, 0.8, 0.6, 0.4, 0.2\} \), the upper bound would be two. For the case when \( \mathcal{B} = \{1.0, 0.2\} \), the upper bound would be five. Computational tests on both of these sets of bins were performed to test the upper bounds and the performance of the heuristic.

2.1. Uniformly Spaced Bin Sizes

Consider the problem of VSBC where \( \mathcal{B} = \{1.0, 0.8, 0.6, 0.4, 0.2\} \). The upper bound as defined in equation (5) as 2.0. Computational studies have verified this. Parallel computing was used to speed up the computation since one computer doing all the work would have taken over 60 days. The computation times were due mainly to the time to compute the optimal solution using a branch and bound technique. The computation of the heuristic took much less than a second for all list sizes that were tested. Item sizes were generated using a uniform distribution in the interval \((0,1]\).

<table>
<thead>
<tr>
<th>List size ((n))</th>
<th>No. Runs</th>
<th>Time</th>
<th>Max Ratio</th>
<th>Average Ratio</th>
<th>% of Ones ((p))</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5000</td>
<td>0</td>
<td>2</td>
<td>1.076</td>
<td>68.8</td>
</tr>
<tr>
<td>4</td>
<td>5000</td>
<td>0</td>
<td>2</td>
<td>1.087</td>
<td>51.76</td>
</tr>
<tr>
<td>5</td>
<td>5000</td>
<td>0</td>
<td>2</td>
<td>1.093</td>
<td>36.94</td>
</tr>
<tr>
<td>6</td>
<td>5000</td>
<td>0</td>
<td>2</td>
<td>1.099</td>
<td>25.76</td>
</tr>
<tr>
<td>7</td>
<td>5000</td>
<td>0</td>
<td>1.8</td>
<td>1.103</td>
<td>18.24</td>
</tr>
<tr>
<td>8</td>
<td>5000</td>
<td>0</td>
<td>1.75</td>
<td>1.106</td>
<td>11.86</td>
</tr>
<tr>
<td>9</td>
<td>5000</td>
<td>14 secs</td>
<td>1.625</td>
<td>1.111</td>
<td>8.28</td>
</tr>
<tr>
<td>10</td>
<td>5000</td>
<td>4 min</td>
<td>1.6</td>
<td>1.114</td>
<td>4.74</td>
</tr>
<tr>
<td>11</td>
<td>5000</td>
<td>33 min</td>
<td>1.75</td>
<td>1.115</td>
<td>4.08</td>
</tr>
<tr>
<td>12</td>
<td>5000</td>
<td>3.4 hours</td>
<td>1.455</td>
<td>1.118</td>
<td>2.08</td>
</tr>
<tr>
<td>13</td>
<td>5000</td>
<td>15 hours</td>
<td>1.462</td>
<td>1.120</td>
<td>1.3</td>
</tr>
<tr>
<td>14</td>
<td>5000</td>
<td>2.6 days</td>
<td>1.4</td>
<td>1.122</td>
<td>0.9</td>
</tr>
<tr>
<td>15</td>
<td>5000</td>
<td>11.3 days</td>
<td>1.455</td>
<td>1.123</td>
<td>0.6</td>
</tr>
<tr>
<td>16</td>
<td>5000</td>
<td>45 days</td>
<td>1.462</td>
<td>1.127</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Table 6. Variable Sized Bin Covering with Five Bin Sizes

The maximum ratio was 2.0 for list sizes between 3 and 6, and beyond that, it decreased with increasing list size. At the same time, the average ratio increased. The percentage of ones decreased, as in the case of several algorithms for USBC (such as FF and HF in Tables 3 and 5 respectively).
2.2. Non-uniformly Spaced Bin Sizes

Consider the problem of VSBC where $B = \{1.0, 0.2\}$. The upper bound is defined above in equation (5) as 5.0. Computational studies have verified this. More computer resources became available for this experiment and we were able to use an additional twenty Pentium-4, two Ghz machines for this problem. The parallel computing Java code performed about 11 days’ worth of experimentation in under 9 hours.

<table>
<thead>
<tr>
<th>List size $(n)$</th>
<th>No. Runs</th>
<th>Time</th>
<th>Max Ratio</th>
<th>Average Ratio</th>
<th>% of Ones $(p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5000</td>
<td>0</td>
<td>5</td>
<td>1.618</td>
<td>49.58</td>
</tr>
<tr>
<td>4</td>
<td>5000</td>
<td>0</td>
<td>5</td>
<td>1.807</td>
<td>27.84</td>
</tr>
<tr>
<td>5</td>
<td>5000</td>
<td>0</td>
<td>5</td>
<td>1.902</td>
<td>14.92</td>
</tr>
<tr>
<td>6</td>
<td>5000</td>
<td>0</td>
<td>3.75</td>
<td>1.993</td>
<td>7.3</td>
</tr>
<tr>
<td>7</td>
<td>5000</td>
<td>0</td>
<td>3.75</td>
<td>2.056</td>
<td>3.14</td>
</tr>
<tr>
<td>8</td>
<td>5000</td>
<td>0</td>
<td>4</td>
<td>2.116</td>
<td>1.6</td>
</tr>
<tr>
<td>9</td>
<td>5000</td>
<td>0</td>
<td>4</td>
<td>2.158</td>
<td>0.64</td>
</tr>
<tr>
<td>10</td>
<td>5000</td>
<td>0</td>
<td>3.4</td>
<td>2.212</td>
<td>0.3</td>
</tr>
<tr>
<td>11</td>
<td>5000</td>
<td>0</td>
<td>3.667</td>
<td>2.241</td>
<td>0.14</td>
</tr>
<tr>
<td>12</td>
<td>5000</td>
<td>0</td>
<td>3.75</td>
<td>2.277</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>5000</td>
<td>6 secs</td>
<td>3.571</td>
<td>2.299</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>5000</td>
<td>1 min</td>
<td>3.5</td>
<td>2.326</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>5000</td>
<td>17 min</td>
<td>3.444</td>
<td>2.346</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>5000</td>
<td>1.5 hours</td>
<td>3.444</td>
<td>2.375</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>5000</td>
<td>5.5 hours</td>
<td>3.273</td>
<td>2.383</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>5000</td>
<td>18 hours</td>
<td>3.333</td>
<td>2.404</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>5000</td>
<td>2.5 days</td>
<td>3.231</td>
<td>2.417</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>5000</td>
<td>7.1 days</td>
<td>3.154</td>
<td>2.432</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7. Variable Sized Bin Covering with Two Bin Sizes

The maximum ratio obtained was 5, although it was decreasing as the list size increased. The percentage of ones dropped off sharply, and as before, the average ratio increased.

3. Lower Bounds in Bin Packing

3.1. Preliminaries

We now turn our attention to the online LIB Bin Packing problem with unit-sized bins. We prove that no algorithm can guarantee an AR (approximation ratio) of less than 1.76, under the online model considered. Computing resources have been used to obtain theoretical results.
Problem Statement: Online LIB Uniform-Sized Bin Packing (USBP). Given an infinite supply of unit-sized bins, and a list $L$ with $|L| = n$ 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. In addition, the LIB constraint (2) should be obeyed for any used bin. A feasible solution is one where the sum of the item sizes in each used bin is at most one. The goal is to find a feasible solution that minimises the number of used bins.

Literature on Bin Packing. Coffman et al [4] provide a comprehensive review of heuristics in Bin Packing. The original harmonic algorithm for bin packing called Harmonic$_M$ was introduced in [10], which also featured a slightly modified version of this algorithm called Refined-Harmonic. Without the LIB constraint, and using NF to pack the items, these algorithms were shown to have AAR’s of 1.692 and 1.636 respectively. For the non-LIB case, the best lower bound obtained so far [14] is 1.53, that is, no heuristic for this problem can guarantee an approximation ratio of less than 1.53 under the online model considered.

Manyem et al [11–13] treat the online LIB version of Bin Packing. They show that the worst case approximation ratio of the Next Fit (NF) algorithm is in $\Theta(n)$. They provide a modified FF algorithm with a guaranteed upper bound of three on the asymptotic approximation ratio (AAR) and computational results for their heuristic. As for lower bounds, Manyem et al [13] show that

**Lemma 1.** For the online LIB uniform-sized Bin Packing (USBP) problem, the FF, BF and HF heuristics cannot guarantee an AAR better than two. ■

3.2. Setting Up an Instance of the Problem

We now turn our attention to creating a problem instance in online LIB USBP that can grow to an infinite size. We need a problem instance that can grow infinitely large, in order to compute the AAR defined in Section 1.1. Observe that we should now compute the approximation ratio as $A(L, B)/OPT(L, B)$, since Bin Packing is a minimisation problem.

Consider a list $L$ that contains three sublists $(L_1, L_2, L_3)$, in that order. The size of items in sublist $L_i$ is $a_i$, $i = 1, 2, 3$. Eventually, we settle on specific item sizes $a_1 = 0.48$, $a_2 = 0.043$ and $a_3 = 0.047$ — however, it makes sense to carry out a more general analysis first.

Let $|L_i| = n_i$, the number of items in sublist $L_i$. To ensure that the LIB constraint is used, we assume that

$$0 < a_2 < a_3 < a_1$$

In a list $L$ with two sublists $L_1$ and $L_2$, the LIB constraint does not come into effect — the online constraint does the same job. For this reason, we assume that $n_1, n_2, n_3 \geq 1$.

The online and LIB constraints provide for the following rules for the placement of items of different sizes in the same bin:

1. An $L_2$ item can be placed on top of an $L_1$ item,
(2) An $L_3$ item can be placed on top of an $L_1$ item, and
(3) No other “mixed item” placements are allowed in the same bin.

Let $m_1$ ($m_2$, $m_3$) be the maximum number of $L_1$ ($L_2$, $L_3$) items that can be placed in a bin. Thus $m_1 = \lceil 1/a_1 \rceil$ (and $m_2 = \lceil 1/a_2 \rceil$, $m_3 = \lceil 1/a_3 \rceil$).

Let us define \textit{multi-stacking} as placing the maximum number of the same category of items in a bin or bins. For example, if all $L_2$ items are multi-stacked, they would occupy $\lceil n_2/m_2 \rceil$ bins. With multi-stacking, items of different sizes cannot be placed in the same bin. Assume that

$$1/3 < a_1 < 0.5$$

Hence $m_1 = 2$. Let $m_{12}$ ($m_{13}$) be the maximum number of $L_2$ ($L_3$) items that can be placed on top of an $L_1$ item. Thus,

$$m_{12} = \left\lfloor \frac{1-a_1}{a_2} \right\rfloor \quad \text{and} \quad m_{13} = \left\lfloor \frac{1-a_1}{a_3} \right\rfloor$$

The following additional constraints are imposed on item sizes $a_1$, $a_2$ and $a_3$:

- If a bin contains two $L_1$ items, then no more items can be placed in the bin:
  $$2a_1 + a_2 > 1 \quad \text{and} \quad 2a_1 + a_3 > 1$$

- It is more optimal to place $L_3$ ($L_2$) items on top of $L_1$ items rather than multi-stacking $L_2$ ($L_3$) items. In other words, a bin with one $L_1$ item and $m_{12}$ ($m_{13}$) number of $L_2$ ($L_3$) items is more tightly packed than a bin containing $m_2$ ($m_3$) number of $L_2$ ($L_3$) items:
  $$a_1 + m_{12}a_2 > m_2a_2$$
  and
  $$a_1 + m_{13}a_3 > m_3a_3$$

- Similarly, a bin with one $L_1$ item and $m_{12}$ ($m_{13}$) number of $L_2$ ($L_3$) items is more tightly packed than a bin containing two $L_1$ items:
  $$a_1 + m_{12}a_2 > 2a_1$$
  and
  $$a_1 + m_{13}a_3 > 2a_1$$

- A bin with one $L_1$ item and $m_{13}$ number of $L_3$ items is more tightly packed than a bin containing one $L_1$ item and $m_{12}$ number of $L_2$ items. In other words, in a bin with one $L_1$ item, it is better to fill in $L_3$ items than $L_2$ items:
  $$a_1 + m_{13}a_3 > a_1 + m_{12}a_2$$

The parameters $m_2$, $m_3$, $m_{12}$ and $m_{13}$ are chosen in such a manner as to obtain the “best” ($a_1$, $a_2$, $a_3$) combination. A particular ($a_1$, $a_2$, $a_3$) combination is considered to be the best if it provides the highest value for the lower bound $LB$, as explained in (22) in Section 3.5. These four parameters can of course, vary widely for example, since $1/3 < a_1 < 0.5$, and hence $0 < a_2 < 0.5$, the value of $m_2$ can range from 2 to a potentially large value. Thus our search process is limited by available computing resources, and hence should be planned with judicious use of computing power in mind.
Once the parameters $m_2$, $m_3$, $m_{12}$ and $m_{13}$ are fixed, one can obtain the sizes $a_1$, $a_2$ and $a_3$ by solving an IP (integer program) with (6)-(12) as constraints — any linear objective function consisting of $a_1$, $a_2$ and $a_3$ can be used as long as the IP does not yield an unbounded solution. We used the CPLEX solver to solve the IP. Of course, to solve the integer program, one should replace each $<$ ($>$) constraint by a $\leq$ ($\geq$) constraint by adding a suitable tolerance constant such as 0.01 or 0.001 to the left side (right side) of a $\leq$ ($\geq$) constraint.

For a specific $(a_1, a_2, a_3)$ combination, the approximation ratio is computed as explained below. We consider the computation of the heuristic solution value first.

### 3.3. Heuristic Solution

A certain heuristic (or a certain strategy) can be described by a tuple $(p_1, p_2)$, where $p_1$ is the ratio of $L_1$ items that are double-stacked (two in a bin), and $p_2$ is the ratio of $L_2$ items that can be placed on top of $L_1$ items. A heuristic that, for instance, double-stacks a constant number of $L_1$ items and single-stacks the remaining $L_1$ items will asymptotically reduce to a $(0, p_2)$ heuristic as $|L_1|$ increases — hence such heuristics that double-stack a constant number of $L_1$ items, as opposed to a percentage of them, need not be considered.

Here is how the heuristic $H(p_1, p_2)$ behaves (see Table 1 for Notation):

- As items in $L_1$ arrive, the heuristic $H$ will double-stack $d_1$ of these items and single stack the rest ($s_1 = n_1 - d_1$). $d_1$ is the number of single-stacked $L_1$ items. $d_1$ is equal to $\lfloor p_1 n_1 \rfloor$ if $\lfloor p_1 n_1 \rfloor$ is even, and $(\lfloor p_1 n_1 \rfloor - 1)$ otherwise.

- When items in $L_2$ begin arriving, $H$ will attempt to place $s_2$ of these into bins single-stacked with an $L_1$ item, and the remaining $L_2$ items ($\beta = n_2 - s_2$ in number) will be multi-stacked. If $s_1$ is insufficient to carry the placement of $s_2$ items of size $a_2$, that is, $s_1 < s_2/m_{12}$, then $\beta$, the number of multi-stacked $L_2$ items will increase from $n_2 - s_2$ to $n_2 - s_1 m_{12}$. Let $\alpha = \lfloor p_2 n_2 \rfloor$. Then $s_2$ is given by

$$s_2 = \alpha, \text{ if } \alpha \text{ is divisible by } m_{12}, \text{ and } s_2 = m_{12} \left\lfloor \frac{\alpha}{m_{12}} \right\rfloor \text{ otherwise.} \quad (13)$$

- Lastly, when items in $L_3$ arrive, $H$ will place as many of them on top of singly-stacked $L_1$ items as possible. The remaining $L_3$ items will be

---

1We wish to emphasise that we focus on asymptotic behaviour here. For example, one could argue that there is a strategy where $p_2$ depends on $p_1$ and $n_1$, such as: (1) if $n_1 \leq 100$ and $p_2 \leq 0.2$, then $p_2 = 0.1$, (2) if $n_1 \leq 100$ and $p_1 > 0.2$, then $p_2 = 0.5$, (3) if $n_1 > 100$ and $p_1 > 0.4$, then $p_2 = 0.8$, and so on. However, the number of such cases is finite, and hence one ultimately reaches a case that considers all values of $n_1$ greater than a finite positive integer $n_1^0$ for which only one value of $p_1$ can be chosen — and for example, suppose $p_2$ can be chosen as follows: (a) if $p_1 \leq 0.3$, then $p_2 = 0.8$, (b) if $0.3 < p_1 \leq 0.6$, then $p_2 = 0.7$, and (c) if $0.6 < p_1 \leq 1.0$, then $p_2 = 0.45$. Thus asymptotically, we have encountered these heuristics in this example: $(p_1 \leq 0.3, p_2 = 0.8)$, $(0.3 < p_1 \leq 0.6, p_2 = 0.7)$, and $(0.6 < p_1 \leq 1.0, p_2 = 0.45)$ — the cases when $n_1 < n_1^0$ are not considered in our lower bound analysis.
multi-stacked. Recall that $L_3$ items cannot be placed on top of $L_2$ items due to the LIB constraint.

Depending on the relative numbers of $L_1$, $L_2$ and $L_3$, the number of bins $h$ used by the heuristic solution is computed as below in (a)-(c):

(a) If $s_1 \geq s_2/m_{12} + \lceil n_3/m_{13} \rceil$, then

$$h = d_1/2 + s_1 + \lceil \beta/m_2 \rceil$$

(14)

(b) If $s_1 < s_2/m_{12} + \lceil n_3/m_{13} \rceil$ but $s_1 \geq s_2/m_{12}$. Those $L_2$ items intended to be placed on top of (single-stacked) $L_1$ items can indeed be placed so. However, the remaining single-stacked $L_1$ bins are insufficient to accomodate the $L_3$ items:

$$h = s_1 + \left\lceil \frac{n_3}{m_3} \right\rceil + \left\lceil \frac{n_2 + m_{13}(s_1 - s_2/m_{12})}{m_2} \right\rceil$$

(15)

(c) Neither of the above two cases. The single-stacked $L_1$ bins are insufficient even to accomodate the $s_2$ number of $L_2$ items meant to go on top of $L_1$ items — hence making the placement of $L_3$ items on top of single-stacked $L_1$ items impossible:

$$h = s_1 + \left\lceil \frac{n_3}{m_3} \right\rceil + \left\lceil \frac{\beta + n_2 - s_1 m_{12}}{m_2} \right\rceil$$

(16)

3.4. Optimal Solution

An optimal algorithm OPT (one that produces an optimal solution) will behave as follows:

If $m_{13}n_1 \leq n_3$

Single-stack all $L_1$ items;
Place as many $L_3$ items as possible in bins with $L_1$ items;
Multi-stack the remaining $L_3$ items, and
Multi-stack all $L_2$ items.

Else

(Now we have $m_{13}n_1 > n_3$)

If $n_1 \leq \lfloor n_3/m_{13} \rfloor + \lfloor n_2/m_{12} \rfloor$

Single-stack all $L_1$ items;
Place ALL $L_3$ items in bins with $L_1$ items;
Place as many $L_2$ items as possible in bins with $L_1$ items, and
Multi-stack the remaining $L_2$ items.

Else

(Now $\lfloor n_2/m_{13} \rfloor + \lfloor n_3/m_{12} \rfloor < n_1$)

Single-stack $\lfloor n_2/m_{13} \rfloor + \lfloor n_3/m_{12} \rfloor$ number of $L_1$ items;
Double stack the remaining $L_1$ items;
Place ALL $L_3$ items in bins with single-stacked $L_1$ items, and
Place ALL $L_2$ items in bins with single-stacked $L_1$ items.

End If
End If

Accordingly, the value of $opt$, the optimal solution value, is computed as below in (a)-(c):

(a) If $m_{13} n_1 \leq n_3$ (there is a sufficient number of $L_3$ items to be placed on top of single-stacked $L_1$ items), then

$$opt = n_1 + \left\lceil \frac{n_3 - m_{13} n_1}{m_3} \right\rceil + \left\lceil \frac{n_2}{m_2} \right\rceil$$

(b) If $m_{13} n_1 > n_3$ and $n_1 \leq \left\lceil \frac{n_3}{m_{13}} \right\rceil + \left\lceil \frac{n_2}{m_{12}} \right\rceil$. That is, all $L_3$ items can be placed in bins single-stacked with $L_1$ items, but there may be a few $L_2$ items that OPT will be unable to place on top of single-stacked $L_1$ items. In such a case,

$$opt = n_1 + \left\lceil \frac{n_2 - m_{12} (n_1 - \left\lceil \frac{n_3}{m_{13}} \right\rceil)}{m_2} \right\rceil$$

(c) If $n_1 > \left\lceil \frac{n_3}{m_{13}} \right\rceil + \left\lceil \frac{n_2}{m_{12}} \right\rceil$, then

$$opt = \left\lceil \frac{n_3}{m_{13}} \right\rceil + \left\lceil \frac{n_2}{m_{12}} \right\rceil + \left\lceil \frac{n_1 - \left\lceil \frac{n_3}{m_{13}} \right\rceil - \left\lceil \frac{n_2}{m_{12}} \right\rceil}{2} \right\rceil$$

3.5. Approximation Ratios and Lower Bound

Observe that we should now compute the approximation ratio as $A(L, B)/OPT(L, B)$ — see Section 1.1 — since Bin Packing is a minimisation problem. Since the bins are of uniform size, we can simplify $A(L, B)$ and $OPT(L, B)$ to $A(L)$ and $OPT(L)$ respectively.

For a given heuristic $H(p_1, p_2)$, characterised by the tuple $(p_1, p_2)$, the guaranteed approximation ratio is the maximum of the approximation ratios $(h/opt)$ over all instances of the problem — each instance of the problem is specified by an $(n_1, n_2, n_3)$ tuple, where $n_i = |L_i|, 1 \leq i \leq 3$. However, the infinite number of $(n_1, n_2, n_3)$ tuples fall into a subset of nine cases (3 cases each for the heuristic and optimal solutions), as far as approximation ratios are concerned.

For the 3 cases for the heuristic solution, let us name the values as $h_1, h_2$ and $h_3$. Similarly, the optimal solution values for the three cases are named as $opt_1, opt_2$ and $opt_3$. The 9 cases can be named as case $(i, j)$, where $i$ ($j$) represents one of the three cases in the heuristic (optimal) solutions. The maximum approximation ratio $r(i, j)$ is computed for each of the 9 cases, and the overall maximum ratio over all 9 cases gives the guaranteed approximation ratio $R(a_1, a_2, a_3, p_1, p_2)$ for a given $H(p_1, p_2)$ heuristic:

$$R(a_1, a_2, a_3, p_1, p_2) = \max_{1 \leq i \leq 3, 1 \leq j \leq 3} r(i, j)$$
The lower-bound $LB(a_1, a_2, a_3)$, for a given $(a_1, a_2, a_3)$ combination, is obtained by considering all possible heuristics $H(p_1, p_2)$ as follows:

$$LB(a_1, a_2, a_3) = \frac{\min_{0 \leq p_1 \leq 1, 0 \leq p_2 \leq 1} R(a_1, a_2, a_3, p_1, p_2)}{}$$  \hfill (21)

Of course, $LB(a_1, a_2, a_3)$, obtained by using any $(a_1, a_2, a_3)$ combination, can be taken as a lower bound $LB$ for the online LIB Bin Packing problem. However, one wishes to obtain a better (that is, higher) value for the lower bound — which is why, time permitting, one can experiment with different $(a_1, a_2, a_3)$ combinations that obey the constraints (6)-(12), and choose the best lower bound $LB$ for the problem:

$$LB = \max_{a_1, a_2, a_3} LB(a_1, a_2, a_3)$$  \hfill (22)

For a fixed $(p_1, p_2)$, in each of the nine $(i, j)$ cases, some of the $(n_1, n_2, n_3)$ values will be valid and not others, due to the conditions set forth for each case. For each case $(i, j)$, an optimisation problem was solved as below:

$$\text{maximise } h_i/\text{opt}_j$$  \hfill (23)

subject to

$$\text{condition } i$$  \hfill (24)

$$\text{condition } j$$  \hfill (25)

For instance, if $i = j = 1$, then condition $i$ is $s_1 \geq s_2/m_{12} + \lceil n_3/m_{13} \rceil$ and condition $j$ is given by $m_{13}n_1 \leq n_3$.

### 3.6. Experiments and Convergence of LB

The nature of the objective function did not permit seeking an optimal solution using any of the standard optimisation packages. Hence the optimisation (search) procedure was coded in C language and implemented in a computer running Linux (RedHat 7.1). It was ensured that the search process did converge as the problem size, measured by $\max(n_1, n_2, n_3)$, grew — see Table 8.

After some experimentation, we chose $a_1 = 0.48, a_2 = 0.043$ and $a_3 = 0.047$ — thus fixing $m_1 = 2, m_2 = 23, m_3 = 21, m_{12} = 12$, and $m_{13} = 11$. As for $p_1$ and $p_2$,

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.7738</td>
</tr>
<tr>
<td>200</td>
<td>1.7803</td>
</tr>
<tr>
<td>500</td>
<td>1.7857</td>
</tr>
<tr>
<td>1000</td>
<td>1.7874</td>
</tr>
<tr>
<td>10000</td>
<td>1.7890</td>
</tr>
</tbody>
</table>

Table 8. Variation in Lower Bound for Worst Case Approximation Ratio
we varied them in the $[0, 1]$ interval in steps of 0.02. The experiments resulted in an $LB$ value of 1.7738. To conclude this section, we can state that, after allowing for rounding errors,

**Theorem 2.** Under the online model considered for the LIB bin packing problem with unit-sized bins, no algorithm can guarantee an asymptotic competitive ratio less than 1.76.

### 4. Further Research

For online non-LIB uniform sized Bin Packing, the best lower bound obtained so far is 1.53 [14]. Naturally, one would expect the lower bound for the constrained problem (the LIB case) to be higher — our proof confirms this, though it falls short of the lower bound of two conjectured in [13]. The reason for such a conjecture lies in the results in Lemma 1. Further research, possibly by investigating different lists, might bring the lower bound closer to two.

Another important open problem in online LIB uniform sized Bin Covering (USBC) is the resolution of the following conjecture in [11]:

**Conjecture 3.** No polynomial-time (deterministic) approximation algorithm for the Online USBC problem with LIB can guarantee an asymptotic approximation ratio that is a constant, under the considered online model.

Compare this with the tight bound of two for the non-LIB version — the upper bound was proved in [2], and the lower bound was proved in [9].

The first author was supported by a grant from the University of South Australia. The second author was supported by a grant from the Sir Ross and Sir Keith Smith Foundation in Adelaide, Australia. The authors benefited from discussions with David Panton. A preliminary version of this paper appeared in the proceedings of the fourteenth Australasian workshop on Combinatorial Algorithms (AWOCA 2003).

### References


