6-1-1995

A Tighter Lower Bound for Optimal Bin Packing

Heng Yi Chao  
*Purdue University School of Electrical Engineering*

Mary P. Harper  
*Purdue University School of Electrical Engineering*

Russell W. Quong  
*Purdue University School of Electrical Engineering*

Follow this and additional works at: [http://docs.lib.purdue.edu/ecetr](http://docs.lib.purdue.edu/ecetr)
A Tighter Lower Bound for Optimal Bin Packing

Heng-Yi Chao
Mary P. Harper
Russell W. Quong

TR-ECE 95-15
June 1995

School of Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana 47907-1285
A Tighter Lower Bound for Optimal Bin Packing

Heng-Yi Chao, Mary P. Harper, and Russell W. Quong
School of Electrical Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285
Email: \{hengyi,harper,quong\}@ecn.purdue.edu
Abstract

In this paper, we present an $O(n \log n)$ algorithm to compute a tighter lower bound for the one-dimensional bin packing problem. We have simulated the algorithm on randomly generated bin packing problems with item sizes drawn uniformly from $(a, b]$, where $0 \leq a < b \leq B$ and $B$ is bin size. Using our lower bound, the average error of BFĐ is less than 2%. For $a + b \geq B$, the error is less than 0.003%.

Key words: bin packing, lower bound, best fit decreasing, harmonic partition, matching.
1 Introduction

The quality of an approximation algorithm $A$ is often measured by its asymptotic performance ratio [3] or worst-case performance ratio [6]. For an instance $L$ of a minimization problem $\Pi$, let $S^*(L)$ be the optimal solution and $A(L)$ be the solution obtained by using algorithm $A$. If a quantity implicitly depends on the problem instance $L$, then we drop $L$ from our notation. The asymptotic performance ratio of algorithm $A$ for minimization problem $\Pi$, is defined as [3, page 128]:

$$R(A) = \inf \left\{ r \geq 1 : \text{for some } k \in \mathbb{N}, \frac{A(L)}{S^*(L)} \leq r, \forall L \in \text{domain}(\Pi), \text{satisfying } S^*(L) \geq k \right\}$$

Comparing two algorithms solely using worst-case performance ratios can be misleading because the average-case performance may differ significantly from the worst-case performance.

For an instance $L$ of a minimization problem $\Pi$, if $lb \leq S^*(L) \leq ub$, then $lb$ is called a lower bound and $ub$ is called an upper bound on the optimal solution. These bounds are useful because $S^*$ may not be computed in polynomial time. Clearly, $lb$ (ub) should be as large (small) as possible, with the goal of having $lb = S^* = ub$. Note that an existing heuristic algorithm provides an upper bound on $S^*$. In this paper, we present an efficient algorithm to compute a tighter lower bound for the off-line one-dimensional bin packing problem (a packing algorithm is called on-line if it packs the items in the order they are received; otherwise it is off-line).

The one-dimensional bin packing problem [3] is the problem of partitioning (or packing) a set $L$ of $n$ objects (or items) into a minimum number of bins, such that the sum of the object sizes in each bin does not exceed the bin size, $B$. Let $s_i$ be the size of object $i$, $s_i \in (0, B]$. In discrete bin packing, $s_i$ and $B$ are integers; in continuous bin packing, $B = 1$ and $s_i$ are reals in $(0,1]$.

Because the bin packing problem is NP-complete, heuristic algorithms for the bin packing problem have been studied extensively [1, 4, 5, 6, 10]. Some of the basic algorithms include First-Fit (FF), Best-Fit (BF), Worst-Fit, and Next-Fit. The Best-Fit algorithm assigns each object sequentially to the most fully packed bin into which it fits. First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD) are variations of FF and BF in which the objects are sorted in decreasing (or non-increasing) order before being packed. Johnson [4, 5] provides the worst-case performance ratios, $R(FF) = \frac{17}{10}$ and $R(FFD) = \frac{11}{5}$. The BF and BFD algorithms have the same worst-case performance as FF and FFD [5]; however, BFD performs better than the others in practice. In this paper, we use our lower bound to empirically show that BFD performs very well for a wide range of object size distributions.

In the rest of this paper, we present a tight lower bound for the bin packing problem. In Section 2, we discuss our lower bound strategies. In Section 3, we present our lower bound. In Section 4, we show that our lower bound is effective in practice.
2 Our Lower Bound Strategies

Our lower bound is based on several concepts: the subproblem principle, SUM, Lueker’s functions [8], harmonic partition [6], and special handling of large-sized objects which we classify as BIG. The subproblem principle is introduced in Lemma 1.

**Lemma 1 [Subproblem Principle]** The optimal solution of a subproblem $L_{\text{sub}} \subseteq L$ is a lower bound for the original problem $L$. It follows that a lower bound for $L_{\text{sub}}$ is also a lower bound for $S^*(L)$, because $lb(L_{\text{sub}}) \leq S^*(L_{\text{sub}}) \leq S^*(L)$.

We define SUM as $\left[\frac{\sum_{i \in L} s_i}{B}\right]$, which is an obvious lower bound on $S^*$. Many researchers [1, 9] have used $A(L) - \text{SUM}$, termed empty space (or wasted space), to measure the performance of a packing algorithm, $A$. Bentley et al. [1] used empty space to measure the average performance of FF and FF D for packing items drawn uniformly from the range $(0, b]$, $0 < b \leq 1$. The empty space remains consistently small until $b$ reaches a critical threshold between 0.8 and 0.9 (depending upon $n$), above which it grows without bound. In particular, the empty space grows roughly as $0.3\sqrt{n}$, for $b = 1.0$ [1, 7]. Shor [9] provided tighter lower bounds and upper bounds on the expected wasted space for the on-line BF and FF algorithms when packing items uniformly distributed on $[0,1]$. When there are many small objects, the empty space seems a good performance measurement. However, if the object sizes tend to be large, the simulation data in Section 4 show that the empty space is not a good performance measurement.

Lueker [8] considered packing items drawn uniformly from intervals $[a, b]$, $0 < a < b < 1$. He determined lower bounds on the optimum packing ratios for certain values of $a$ and $b$, where the optimum packing ratio is defined as the expected number of bins to the expected total item size. He considered four regions on the $a$-$b$ plane and gave a feasible function for each region, where a feasible function is a real-valued function $u(x)$ such that $\forall k \geq 1$,

$$\sum_{i=1}^{k} x_i \leq 1 \Rightarrow \sum_{i=1}^{k} u(x_i) \leq 1$$

(1)

The four regions and their corresponding feasible functions defined in [8] are shown in Table 1. A partial plot $(a \geq 1/5)$ of the regions on the $a$-$b$ plane defined by Lueker is shown in Figure 1. Note that these regions cover only part of the $a$-$b$ plane.

Lemma 2 describes how we can use Lueker’s functions to compute a lower bound. If $(a, b)$ is contained in one of the regions defined in Table 1, then a lower bound can be computed by using the feasible functions given in the table. We call this lower bound $LLB$ (Lueker’s Lower Bound).

**Lemma 2** Let $L$ be a bin packing instance containing $n$ items distributed over $(a, b]$, $0 \leq a < b \leq B$. If $u(x)$ is a feasible function on $(a, b]$, then $\left[\sum_{i=1}^{n} u(s_i/B)\right] \leq S^*$.
Table 1: The four regions and their corresponding feasible functions defined by Lueker, where $s = \left(\frac{1}{p} - \frac{1}{p+1}\right)/(\frac{1}{p+1} - a)$.

**Proof:** For each bin $k$ in an optimal packing, $\sum_{i \in k} s_i \leq B$ (i.e., $\sum_{i \in k}(s_i/B) \leq 1$), where $i \in k$ denotes item $i$ being packed in bin $k$. Hence, $\sum_{i \in k} u(s_i/B) \leq 1$ because $u(x)$ is feasible (see Equation 1). It follows that,

$$\sum_{i=1}^{n} u(s_i/B) = \sum_{k=1}^{n} \sum_{i \in k} u(s_i/B) \leq \sum_{k=1}^{n} 1 = S^*$$

and hence, $[\sum_{i=1}^{n} u(s_i/B)] \leq S^*$.

Our lower bound also considers partitioning the interval $(0, B]$ into sub-intervals. Lee and Lee [6] proposed a harmonic algorithm which partitions $(0,1]$ into $M$ intervals: $I_k = (\frac{1}{k+1}, \frac{1}{k})$ for $1 \leq k < M$ and $I_M = (0, \frac{1}{M})$. An object $i$ is called an $I_k$-item if $s_i \in I_k$, and a bin designated to pack $I_k$-items exclusively is called an $I_k$-bin. We have modified the definition of a harmonic partition to accommodate discrete bin packing over the interval $(0, B]$. The $k$-th harmonic interval is denoted $I_k$, $k = 1, 2, \ldots$, where $I_k = \{x : \frac{B}{k+1} < x \leq \frac{B}{k}\}$. The term $I_{\gamma,\ldots,\delta}$ is the union of intervals $I_{\gamma} \cup \ldots \cup I_{\delta}$. For example, if $B = 100$, we have $I_1 = (50..100]$, $I_2 = (33..50]$, $I_3 = (25..33]$, and $I_{2,3} = I_2 \cup I_3 = (25,501$. At most one $I_{2,3}$-item can be packed with an $I_1$-item. Also, given only $I_k$-items, it is optimal to pack $k$ of them to a bin choosing items arbitrarily. We define $L_i$ to be the set of $I_i$-items in $L$ and $L_{\gamma,\ldots,\delta}$ to be the set of $I_{\gamma,\ldots,\delta}$ items in $L$.

If object sizes are not uniformly distributed over $(0,B]$ or tend to be large, $SUM$ becomes unrealistically optimistic. A better lower bound must consider the largest, and hence, the most difficult-to-pack items, that is, $L_{1,2,3}$. Hence, we derive a lower bound, BIG, for $S^*(L_{1,2,3})$, which is so named because it takes into account the big items. BIG is a lower bound on $S^*$ since, by the
Figure 1: A partial plot \((a \geq 1/5)\) of the regions on the \(a-b\) plane defined by Lueker.
subproblem principle, \( \text{BIG} \leq S^*(L_{1,2,3}) \leq S^* \). A lower bound on \( S^*(L_{1,2,3}) \) is the best we can do, because determining \( S^*(L_{1,2,3}) \) is NP-complete (we can reduce three-dimensional matching [3, page 50] to bin packing \( L_{1,2,3} \) in which each bin contains three objects).

Our lower bound \( \text{OB} \) contains three major components, \( \text{SUM} \), \( \text{LLB} \) and \( \text{BIG} \). \( \text{OB} \) is a lower bound as it is the maximum of the lower bounds.

**Lemma 3** \( \text{OB} = \max(\text{SUM}, \text{LLB}, \text{BIG}) \) is a lower bound for \( S^* \).

We now show how to compute \( \text{BIG} \), a lower bound of \( S^*(L_{1,2,3}) \). The main idea behind the computation of \( \text{BIG} \) is to perform a sequence of optimal steps for packing objects in \( L_{1,2,3} \) such that these packed pseudo objects cannot be packed with any other objects. Then we compute a lower bound for packing the leftover items and add it to the number of bins required by the packed pseudo objects.

Let \( A = A_B \cup A_S \). We say \( A_B \) (As) is a big (small) set if packing \( A_B \) always requires \( |A_B| \) bins and for each item \( i \in A_B \), we can pack at most one other item in \( A_S \) with \( i \). For example, \( |L_1| \) bins are needed to pack \( L_1 \) (because each \( I_1 \)-item requires a bin), and each \( I_2,3 \)-item can be packed with at most one \( I_1 \)-item. Hence, \( L_1 \) is a big set \( (A_B) \), and \( L_{2,3} \) is a small set \( (A_S) \). Assume we pack \( A_B \) into bins with various items from \( A_S \) via packing \( P \). Let \( P(A_S, A_B) \) be the set of leftover items from \( A_S \) not packed with \( A_B \). Let \( P(A_S, A_B) = \{ P(A_S, A_B) : \text{for all packings } P \} \). Let \( U = P(A_S, A_B) \) and \( U' = P'(A_S, A_B) \) for two packings \( P \) and \( P' \). Assume \( U \) and \( U' \) are sorted into decreasing order. We say \( U \) is strongly minimal over \( P(A_S, A_B) \) if \( \forall U' \in P(A_S, A_B), |U| \leq |U'| \) and \( s_i \leq s'_i \), where \( 1 \leq i \leq |U| \). Namely, \( U \) is minimal both in cardinality and on an item-by-item basis over all sets of leftover small items. Analogously, \( U \) is strongly maximal if \( \forall U' \in P(A_S, A_B), |U| \geq |U'| \) and \( s_i \geq s'_i \), where \( 1 \leq i \leq |U'| \). The next lemma indicates why computing a strongly minimal set is useful for bin packing.

**Lemma 4** If \( U \) is strongly minimal over \( P(A_S, A_B) \), then \( S^*(U) \leq S^*(U'), \forall U' \in P(A_S, A_B) \).

**Proof:** In the optimal packing of \( U' \), replacing \( s'_i \) with \( s_i \) is valid as \( s_i \leq s'_i \). Thus, \( S^*(U) \neq S^*(U') \).

The next theorem indicates that after packing big items, if the set of the remaining items is strongly minimal, the packing is optimal.

**Theorem 1** If \( U = P(A_S, A_B) \) is strongly minimal, then \( S^*(A) = |A_B| + S^*(C') \).

**Proof:** Let \( P' \) be any packing of \( A \) and let \( U' = P'(A_S, A_B) \). Both \( P \) and \( P' \) require \( |A_B| \) bins to pack \( A_B \). If we optimally pack \( U \) and \( U' \), we need \( |A_B| + S^*(U) \) and \( |A_B| + S^*(U') \) bins, respectively. However, as \( U \) is strongly minimal, \( S^*(U) \leq S^*(U') \) by Lemma 4. Thus, packing \( P \) followed by optimally packing \( U \) cannot be improved upon.

Note that \( L_1 \) is a big set and \( L_{2,3} \) is a small set. If we can find a strongly maximal set \( X \) of \( L_{2,3} \) and its corresponding set \( Y \) in \( L_1 \), then \( L_{2,3} \setminus X \) is strongly minimal, and it is therefore optimal to pack each pair of objects in \( (X, Y) \) by Theorem 1.
3 BIG, A New Lower Bound for $S^*(L_{1,2,3})$

In this section, we describe how to calculate BIG, a lower bound for $S^*(L_{1,2,3})$. We first pack $L_1$ with items from $L_{2,3}$ using an optimal matching algorithm which leaves a strongly minimal set of "unmatched" items in $L_{2,3}$, denoted $U_{2,3}$. We then determine a lower bound for $S^*(U_{2,3})$, and add it to $|L_1|$ because the $L_1$-items must occupy $|L_1|$ bins no matter how they are packed. In Section 3.1, we give the matching algorithm and prove its optimality; in Section 3.2, we determine a lower bound for the unmatched items, $U_{2,3}$.

3.1 Optimal Matching

We say that two sets $X = \{x_1, \ldots, x_m\} \subseteq L_{2,3}$ and $Y = \{y_1, \ldots, y_l\} \subseteq L_1$ form a matching if for $1 \leq i \leq m$, each pair $x_i$ and $y_i$ can be placed in the same bin. Note that $(X, Y)$ forms a matching [2] in the common sense that each $L_{2,3}$-item can be packed with at most one $L_1$-item. We use the greedy algorithm MATCH$(L_{1,2,3})$ to match the items in $L_{2,3}$, sorted in order of decreasing size, with the largest possible remaining $L_1$-item. Intuitively, MATCH is driven by $L_{2,3}$, not $L_1$. When an item in $L_{2,3}$ is matched with an item in $L_1$, it is moved to the set $X$ and its match is moved to the set $Y$. Algorithm MATCH computes the sets $X$ and $Y$ of matched $L_{2,3}$ and $L_1$ items, respectively. When the algorithm is complete, $U_1$ contains the unused $L_1$-items and $U_{2,3}$ contains the unused $L_{2,3}$-items. Note that this process is equivalent to performing FFD or BFD on $L_{1,2,3}$-items such that those items in $L_{2,3}$ not assigned to bins containing $L_1$-items define the set $U_{2,3}$.

```
Algorithm MATCH(L_{1,2,3})
    sort L_{2,3} = \{v_1 \geq \cdots \geq v_m\};
    X := Y := \emptyset;
    for each v_i = v_1 to v_m in L_{2,3}
        T := L_1 items matching v_i;
        if (T is not empty) then
            t := the largest item in T;
            move t from L_1 to Y
            move v_i from L_{2,3} to X
        end
    end
    U_1 := L_1;
    U_{2,3} := L_{2,3};
```

To guarantee that BIG is a lower bound, it is important that the set of leftover $L_{2,3}$-items, $U_{2,3}$, is strongly minimal. Theorem 2 proves that the matching $(X, Y)$ produced by MATCH$(L_{1,2,3})$ is part of an optimal packing of $L_{1,2,3}$.
**Theorem 2** Let $X$ be the set of $L_{2,3}$-items matched by $\text{match}(L_{1,2,3})$. Let $X'$ be the set of $L_{2,3}$-items matched by any other matching algorithm $A'$. Let $X = \{x_1, \ldots, x_n\}$ and $X' = \{x'_1, \ldots, x'_{n'}\}$ where $X$ and $X'$ have been sorted into decreasing (non-increasing) order. Then $X$ is strongly maximal over all $X'$.

**Proof:** To simplify the notation, we take $x_i$ to represent both the object and its size. We use induction on $i$, showing $x_i \geq x'_i$, thus we are comparing the $i$th largest element matched by $A$ with that of $A'$. Recall that $\text{match}(L_{1,2,3})$ repeatedly matches the largest remaining item in $L_{2,3}$ with the largest item, $y_i$, remaining in $L_1$. Let $y_i$ be the $i$th $L_1$-item matched by $\text{match}(L_{1,2,3})$ and $X = \{x_1, \ldots, x_n\}$.

**Inductive Step:** Assuming the largest $k$ items in $X$ are maximal, $x_i \geq x'_i$, for $1 \leq i \leq k$, we show the $(k + 1)$st item in $X$ is maximal, i.e., $x_{k+1} \geq x'_{k+1}$ assuming $X'$ has a $(k + 1)$st item.

Assume the contrary, namely $x'_{k+1} > x_{k+1}$. By the inductive hypothesis, $x_k \geq x'_k$, so we have $x_k \geq x'_k \geq x'_{k+1} > x_{k+1}$. $A'$ is able to match $x'_{k+1}$ to some item in $L_1$, but $\text{match}(L_{1,2,3})$ is unable to match $x'_{k+1}$, otherwise it would have matched $x'_{k+1}$ and not $x_{k+1}$. Hence, at this point, all remaining items in $L_1$ must be greater than $(B - x'_{k+1})$ in size. As the $k$ matched items $\{x_1, x_2, \ldots , x_k\}$ all have a size greater than or equal to $x'_{k+1}$, each item in $Y_k$ must be less than or equal to $(B - x'_{k+1})$, namely $Y_k$. However, $A'$ has already matched $k$ items greater than or equal to $x'_{k+1}$, namely $\{x_1, \ldots , x_k\}$, and these items must be matched with $Y_k$. Thus, no remaining item in $L_1$ is small enough to be matched with $x'_{k+1}$, so $A'$ cannot possibly match $x'_{k+1}$, yielding a contradiction. Thus, we must have $x_{k+1} \geq x'_{k+1}$.

A similar argument shows $|X| \geq |X'|$. That is, if $n < n'$, $X$ cannot contain an $x_{n+1}$ item, but the above reasoning shows that $X'$ cannot contain a corresponding $x'_{n+1}$ item either.

As $\text{match}(L_{1,2,3})$ partitions $L_{2,3}$ into $X$ and $U_{2,3}$, we know $U_{2,3}$ is strongly minimal because $X$ is strongly maximal. Thus, Theorem 1 indicates the following steps give an optimal packing of $L_{1,2,3}$:

1. Run $\text{match}(L_{1,2,3})$.
2. Pack each of the $L_1$-items remaining in $U_1$ in a separate bin.
3. Pack each pair $x_i$ and $y_i$ from $X$ and $Y$ together in a separate bin.
4. Optimally pack the $U_{2,3}$ using $S^*(U_{2,3})$ bins. This step is NP-complete, in general.

Steps 2 and 3 pack the $L_1$-items and items in $X$ using $|L_1|$ bins. Thus, we have:

$$S^*(L_{1,2,3}) = |L_1| + S^*(U_{2,3})$$

7
Lemma 5 Both BFD and FFD provide an optimal solution when given only $I_{1,2}$ (or $I_{1,3}$) items.

Proof: In this case, both BFD and FFD mimic MATCH($I_{1,2,3}$) and $U_{2,3}$ consists of only $I_2$ (or $I_3$) items. Then each would pack the $U_{2,3}$ objects two (or three) to a bin, avoiding the NP-complete problem of packing mixed $I_{2,3}$-items. 

3.2 After Matching

Although we could apply SUM to $U_{2,3}$, SUM can perform quite poorly as packing the $I_{2,3}$-items often leaves much empty space in bins. We obtain a better bound by considering the unpacked $I_2$-items separately. Let $U_2$ be the set of $I_2$-items in $U_{2,3}$; let $u_2 = |U_2|$ and $u_{2,3} = |U_{2,3}|$. Packing $U_2$ alone requires $\lceil u_2/2 \rceil$ bins. We can pack at most three items in $U_{2,3}$ to a bin, requiring at least $\lceil u_{2,3}/3 \rceil$ bins. Thus, a lower bound for $S^*(U_{2,3})$ is $\lceil \max(\frac{u_2}{2}, \frac{u_{2,3}}{3}) \rceil$.

In practice, we can improve this bound slightly because we must often pack large $I_2$-items with only one other object. For example, if $B = 100$ and $U_{2,3} = \{27, 30, 34, 38, 42, 45, 48\}$, we can pack neither 45 nor 48 with two other objects. Let $s_i, s_j$ be the sizes of the smallest $I_{2,3}$-items with $s_i \leq s_j$. We define the Z-interval (a sub-interval of $I_2$ in most cases) as:

$$Z = \begin{cases} 
[s_i, \frac{B}{2}] & \text{if } s_i \in I_2 \\
(B - s_i - s_j, \frac{B}{2}) & \text{if } s_i \in I_3
\end{cases}$$

Lemma 6 says that the Z-interval contains items which can be packed with at most one other item; hence, it is best to pack them with other Z-items. Figure 2 depicts the partitioning of $I_{1,2,3}$-items into classes.

Lemma 6 It is optimal to pack every two Z-items together.

Proof: First we show a Z-item can be packed with at most one other item. If $s_i \in I_2$, all we have are $I_2$-items. If $s_i \in I_3$, consider packing a Z-item (of size $s_j$), with two items of size $s_i$ and $s_j$. Then $s_2 + s_1 + s_m \geq s_2 + s_i + s_j > B$, exceeding the bin capacity.

Consider any Z-item. By itself, it forms a set of one big item. If we pack it with the largest remaining $U_{2,3}$ item (possibly a Z-item itself), the remaining items in $U_{2,3}$ are strongly minimal, and Theorem 1 indicates this is optimal.

Hence, the new lower bound BIG for $S^*(I_{1,2,3})$ is obtained by the following steps:

1. Run MATCH($I_{1,2,3}$) (requiring $|L_3|$ bins).
2. Pair up and remove the Z-items from $U_{2,3}$ (requiring 12/21 bins).
3. Choose the larger lower bound for the remaining non-Z-items in $U_{2,3}$ using:
   (a) $m_1 = \lceil \max(\frac{u_2}{2}, \frac{u_{2,3}}{3}) \rceil$.
   (b) $m_2$ = a lower bound computed by using Luiker's functions.
Figure 2: The set of $L_{1,2,3}$-items are partitioned into $X$, $Y$, $Z$, $U_{2,3}$-items, and $U_1$ after running $\text{MATCH}(L_{1,2,3})$ and determining the $Z$-interval, where $B$ is the bin size.

Hence, $BIG = |L_1| + \lceil z/2 \rceil + \max(m_1, m_2)$, which is computed by the algorithm $\text{CALC-BIG()}$. If there are an odd number of $Z$-items, we pack the leftover $Z$-item with the largest non-$Z$ item in $U_{2,3}$. The running time of $\text{CALC-BIG()}$ is dominated by the call to $\text{MATCH}$, which must sort the $L_{2,3}$-items. Hence, $\text{CALC-BIG}$ requires $O(n \log n)$ time and is quite practical to run.

Algorithm $\text{CALC-BIG()}$

\begin{verbatim}
match($L_{1,2,3}$);
    $z :=$ the number of $Z$-items from $U_{2,3}$;
    \text{remove the $Z$-items from $U_{2,3}$};
    if ($z$ is odd) then
        \text{discard the largest item from $U_{2,3}$};
        $z := z + 1$;
    end
    $u_3 :=$ the number of $I_2$-items in $U_{2,3}$;
    $u_{2,3} := |U_{2,3}|$;
    $m_1 := \lceil \max\left(\frac{u_3}{3}, \frac{u_{2,3}}{3}\right) \rceil$;
    compute a lower bound $m_2$ for $U_{2,3}$ using Lueker's functions in Table 1;
    return $|L_1| + \lceil \frac{z}{2} \rceil + \max(m_1, m_2)$;
end
\end{verbatim}

4 Experimental Results and Analysis

To test the effectiveness of our lower bound, $OB$, we have run simulations on a RS/6000 workstation running AIX 3 using the bin size $B = 100$. Samples were generated by the random number generator $\text{random()}$ in the standard C library and packed by using the BFD algorithm. The object size is randomly generated over the interval $(a,b]$ for $0 < a < b < B$. For each pair of $a,b$ values, we generated ten instances, where $n = |L| = 30,000$. We simulated 50500 random bin-packing instances in total. Note that Lueker's functions are used twice in the computation of $OB = \max$
(SUM, LLB, BIG), once for computing LLB when \((a, b)\) is in the regions defined by Lueker \[8\] (see Table 1), and once again as \(m_2\), a component of the lower bound for \(S^*(U_{2,3})\) (after matching and Z-Items are removed) in BIG. Because the three components in OB perform well in different regions on the \(a-b\) plane, we divide the plane into the three sub-regions:

\[
R_1 = \{(a, b): a + b \geq B\},
R_2 = \{(a, b): a \geq B/4, a + b < B\},
R_3 = \{(a, b): a < B/4, a + b < B\}.
\]

The three regions (for \(B=1\)) are shown in Figure 1. \(R_1\) corresponds to the triangle \(P_3P_5P_6\); \(R_2\) corresponds to the triangle \(P_2P_3P_4\); and \(R_3\) corresponds to the polygon \(P_1P_2P_4P_5\).

The number of times each component in OB wins in each region is shown in Table 2, where the winner is determined as follows:

\[
\text{if } SUM = OB \text{ then } \text{SUM wins} \\
\text{else if } LLB = OB \text{ then } \text{LLB wins} \\
\text{else BIG wins} \\
\text{end}
\]

BIG is the winner only when it contributes to a tighter lower bound. In region \(R_1\), BIG is the dominant component in OB; in region \(R_2\), LLB wins 79.5% and BIG wins 20.05% of the time; in region \(R_3\), SUM is the dominant component in OB. In region \(R_3\), Lueker's functions help in 136 out of 645 instances when BIG wins, contributing the \(m_2\) term.

Because the actual error rate is unknown, we define the approximate error rates, \(r(SUM) = (BFD - SUM)/SUM\) and \(r(OB) = (BFD - OB)/OB\), where BFD is the solution obtained by using BFD algorithm. Note that an approximate error rate is an upper bound on the actual error rate. For many cases, the errors caused by the poor estimate of SUM are eliminated by our tighter lower bound OB. The minimum (Min), average (Mean), maximum (Max), and standard deviation (a) of \(r(SUM)\) and \(r(OB)\) values for the three regions are summarized in Table 3. The small standard deviation values for \(r(OB)\) show that OB provides a consistently good lower bound.

Figure 3 depicts \(r(SUM)\) and \(r(OB)\) for regions \(R_2\) and \(R_3\) where each data point shown represents the average of ten instances. Consider the worst-case of \(r(SUM)\) and \(r(OB)\) in region \(R_2\). The peak value of \(r(SUM)\) is 47.06% while the peak value of \(r(OB)\) is 24.39%.

a The worst-case of \(r(SUM)\) occurs when the interval is \([34,34]\). If there are \(2m 34s\), then SUM \(= [0.34 + 2m]\) and BFD \(= S^* = m\). Hence, \(r(SUM) = m/0.68m - 1 = 47.06\%\).

a The worst-case of \(r(OB)\) occurs when the interval is \([33,34]\). If there are \(2m 33s\) and \(2m 34s\), then OB \(= [4m/3]\). It is optimal to pack two 33s and one 34s in a bin, yielding
Table 2: Winners in OB for the three regions on the a-b plane. There are two components in BIG after matching and removing the Z-items: \( m_1 = \max(\frac{n_2}{2}, \frac{n_3}{3}) \) and \( m_2 \) is the lower bound for packing \( U_{1,3} \) computed by using Lueker's functions.

<table>
<thead>
<tr>
<th>Region</th>
<th>( SUM ) wins</th>
<th>( LLB ) wins</th>
<th>( BIG ) wins</th>
<th>( m_1 ) wins</th>
<th>( m_2 ) wins</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>17 (0.07%)</td>
<td>0</td>
<td>25483 (99.93%)</td>
<td>24903 (97.66%)</td>
<td>580 (2.27%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>subtotal</td>
<td>25500</td>
<td></td>
</tr>
<tr>
<td>R₂</td>
<td>26 (0.42%)</td>
<td></td>
<td>1255 (20.08%)</td>
<td>1250 (20%)</td>
<td>5 (0.08%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>subtotal</td>
<td>6250</td>
<td></td>
</tr>
<tr>
<td>R₃</td>
<td>17159 (91.51%)</td>
<td>946 (5.05%)</td>
<td>645 (3.44%)</td>
<td>509 (2.71%)</td>
<td>136 (0.73%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>subtotal</td>
<td>18750</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: The statistics for \( r(SUM) \) and \( r(\text{OB}) \).
\[ S^* = m + \lfloor m/2 \rfloor. \] However, BFD would pack two 34s and then pack three 33s in a bin, yielding \( BFD = m + \lceil 2m/3 \rceil \). Hence, \( r(\text{OB}) \approx \frac{5m}{3} - 1 = 25\% \). Note that \( \frac{BFD}{S^*} = (m + \lceil 2m/3 \rceil)/(m + \lfloor m/2 \rfloor) \approx 10/9 \) and \( \frac{OB}{S^*} = \frac{4m/3}{(m + \lfloor m/2 \rfloor)} \approx 8/9 \).

5 Conclusion

We have presented an efficient algorithm to compute a tighter lower bound \( OB \) for the one-dimensional bin packing problem. \( OB \) is the maximum of \( \text{SUM} \), \( \text{LLB} \), and \( \text{BIG} \), which itself is a lower bound for packing objects of size greater than \( B/4 \). When many large objects exist, \( \text{BIG} \) gives a very good estimate on the optimal solution. For many instances \( L \) with large objects \( \text{BFD}(L) = \text{OB} \), empirically proving that BFD produces optimal packings even when large objects dominate the packing. BFD appears to produce optimal packings for a very wide range of object size distributions.
Figure 3: Average $r($SUM$)$ and $r($OB$)$ for $0 \leq a < b \leq B, a + b < B$. 

13
References


