7-22-1993

A Tighter Lower Bound for 1-D Bin-Packing

Heng-Yi Chao
Purdue University School of Electrical Engineering

Mary P. Harper,
Purdue University School of Electrical Engineering

Russell Quong
Purdue University School of Electrical Engineering

A Tighter Lower Bound for 1-D Bin-Packing

Heng-Yi Chao
Mary P. Harper
Russell Quong

TR-EE 93-27
July 1993

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907-1285

A Tighter Lower Bound for 1-D Bin-Packing

Heng-Yi Chao, Mary P. Harper, and Russell Quong
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907-1285
hengyi@ecn.purdue.edu
harper@ecn.purdue.edu
quong8ecn.purdue.edu

July 22, 1993

Abstract

For NP-complete problems, it may not be possible to find an optimal solution in polynomial time. However, efficient approximation algorithms for many NP-complete problems do exist. A good approach is to find tighter upper and lower bounds on the optimal solutions. As the gap between these two bounds approaches zero, optimality is reached. For minimization problems, any feasible solution serves as an upper bound. In general, researchers attempt to find heuristic algorithms to narrow the gap by decreasing the upper bound. Another approach is to narrow the gap by increasing the lower bound. To determine a good lower bound on a problem requires careful examination of the problem's characteristics.

In this paper, we present an efficient algorithm for calculating a lower bound on the number of bins needed in a bin-packing problem. It is obvious that the sum of the sizes of all objects is a lower bound. In addition to this, we consider objects that cannot share their bins with other objects. We consider a subproblem class which contains only objects whose sizes are greater than $\frac{L}{4}$, where L is the size of a bin, given the harmonic partition. An $O(n \log n)$ algorithm finds a lower bound for the subproblem, which in turn is a lower bound for the original problem. Notably, our approach also leads to a polynomial time algorithm for finding optimal solutions of some special cases of bin-packing.

Simulation data show that our lower bound provides an estimate of the optimal number of bins required which is equal to or better than the sum. The approximate error rate is normally less than 1% if our bound is used. The improvement can be as high as 79% when compared to the sum.

Key words: Heuristic Performance Ratio, Bin-Packing, Lower Bound, **Best-Fit** Decreasing, Harmonic Partition, Perfect-k-Way Merge, Matching.

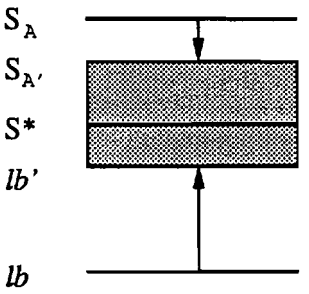


Figure 1: Upper and lower bounds on the optimal solution.

1 INTRODUCTION

For NP-complete problems, it may not be possible to find optimal solutions in polynomial time. However, efficient approximation algorithms for many NP-complete problems do exist. The quality of approximation algorithms is often measured by ***guaranteed worst-case performance ratios.*** Without loss of generality, we consider only minimization problems. Given instance I for a minimization problem P, let $S^A(I)$ denote (the cost of) the solution obtained by using algorithm A and $S^*(I)$ be the optimal (minimum) solution. Note that $S^A(I) \geq S^*(I)$. The worst-case performance ratio of a minimization problem P using algorithm A is defined as:

$$R(A) = \lim_{S^*(I)\to\infty} sup \frac{S^A(I)}{S^*(I)}$$

which is an upper bound on the performance ratio $\frac{S^A(I)}{S^*(I)}$ for all instances of problem P using algorithm A.

However, this bound has the following deficiencies:

1. The R(A) ratio may misjudge the quality of a heuristic. Comparing two algorithms solely using the R(A) ratio bounds can be misleading because the average case performance may differ significantly from the worst case performance. The R(A) ratio for an algorithm is more difficult to calculate as the complexity of the heuristic algorithm increases.
2. In most practical applications, a solution is acceptable, if the error is guaranteed to be within some range (say 5%). However, for many optimization problem algorithms, the R(A) ratio bound often overestimates the error. Clearly, more precise error measurement is important for selecting an appropriate algorithm.

For any problem instance I[1], lb is a lower bound and ub is an upper bound on the optimal

[1] In this paper, unless specified, all quantities are implicitly dependent on the problem instance I. Hence, I is dropped from the notation in situations where there is no ambiguity, e.g., S^* means $S^*(I)$.

solution, if $lb \leq S^* \leq$ ub. Note that no solution less than lb can be found, and there exists a solution equal to ub which can be found in polynomial time. It is clear that lb should be as large as possible, and ub should be as small as possible, with the goal of $lb = ub = S^*$. Note that any existing heuristic algorithm provides an upper bound on the optimal solution. We call the range between the maximum lb and the minimum ub, the ***uncertainty region*** (e.g., the shaded region in Figure 1) because no known polynomial time algorithm provides a solution within that region.

Because of the deficiencies of R(A) ratios and the following reasons, we believe a tight lower bound is important for approximation algorithms:

1. Clearly, a solution which equals the lower bound is optimal. Often, researchers attempt to find a new heuristic A' that provides a less costly solution than the solution given by some other well known heuristic A, moving the upper bound down (see Figure 1, $S_A \searrow S_{A'}$). Another approach is to determine a tight lower bound on the optimal solution, moving the lower bound up (see Figure 1, $lb \nearrow$ lb').
2. Usually researchers simulate their algorithms and other heuristics on randomly generated samples or benchmarks, and then compare the solutions to see their relative performance. On the other hand, a lower bound is an attribute of the problem instance itself and hence, provides an absolute, universal performance measurement for all heuristics in a problem class.
3. A lower bound serves as a termination condition in exhaustive search methods.
4. A tight lower bound is useful for finding a better solution efficiently. It can help to focus on a search path that leads to an optimal or near optimal solution.
5. To determine a tighter lower bound requires careful examination of the problem characteristics, which may lead to a better heuristic.

In this paper, we present a technique for finding a tighter lower bound for bin-packing problems, which may be extended to other NP-complete problems.

1.1 The Bin-Packing Problem

The 1-D bin-packing (BP) problem [GJ79] is the problem of ***packing a set of n objects into a minimum number of bins of fixed capacity, L.*** Let I be the set of objects to be packed and s_i be the ***size*** of object i. The BP problem is formalized as:

[Continuous Bin-Packing (CBP)]

Given: a set of objects I=$\{1...n\}$, $s_i \in (0,1], s_i \in \mathcal{R}$, $\forall i \in I$.

Objective: **assign each object to a unique bin, and minimize the total number of bins used.**

Constraint: The sum of the sizes of all objects assigned to a bin (i.e., its content) does not exceed 1.

[Discrete Bin-Packing (DBP)]

Given: a set of objects $I=\{1...n\}$, $s_i \in (0,L], s_i \in \mathcal{N}, \forall i \in I$.

Objective: assign each object to a unique bin, and minimize the total number of bins used.

Constraint: the content of each bin does not exceed L.

The constraint that the content in each bin cannot exceed the bin's capacity is referred as the capacity constraint. For continuous bin-packing, the capacity of each bin is 1, and all sizes are real numbers in (0,1]. For discrete bin-packing, the capacity of each bin is L, and all sizes are integer numbers in (0,L], with $L \ll n$ normally. In this paper, we consider both continuous and discrete bin-packing, though the primary emphasis is on discrete bin-packing. Without loss of generality, we assume L is even (i.e., L=2M) and L=100. We say that two objects i and j are compatible, if they can be assigned to the same bin, i.e. $s_i + s_j \leq L$. We indicate that $i \preceq j \iff s_i \leq s_j$, $i \prec j \iff s_i < s_j$. Objects assigned to the same bin form a pseudo object.

Because the bin-packing problem is NP-complete, heuristic algorithms have been studied extensively [Joh74, GJ79, Baa88, CLR90]. Several of the basic algorithms are shown in table 1. Garey and Johnson's work [Joh73, Joh74, GJ79] provides worst-case performance ratios for two well-known bin-packing algorithms: R(FF)=17/10, R(FFD)=11/9. Note that BF and BFD algorithms have essentially the same worst-case performance as the FF and FFD algorithms [GJ79]. However, BFD performs better than the others in practice.

1.2 A Lower Bound for the Bin-Packing Problem

Let SUM be the sum of the sizes of all objects in the set. Obviously, SUM is a lower bound on S^* for CBP. Bentley and Johnson's experimental study [BJ83] used empty space to measure the average performance of FF, FFD algorithms, where samples are numbers drawn uniformly on the range $[0,v]$, $0 < v \leq 1$. They define the empty space (error) as "**the number** of **bins used** $-\sum_{i=1}^{n} s_i$". Though the worst case performance ratios of BF and BFD algorithms are essentially the same as FF and FFD algorithms [GJ79], in practice, the BFD algorithm provides much better solutions because it tends to eliminate the empty space in bins. The empty space obtained by performing BFD on samples uniformly distributed over the entire interval are normally less than 2% [GJ79, BJ83]. Though the error is quite small, it grows when v reaches 0.8 (a similar result is shown in Figures 16 and 18), an effect that the authors provided no explanation for.

First-Fit(FF)	Assign each object sequentially to the first bin into which it fits.
Next-Fit(NF)	The bins are filled one at a time and a new bin is started when the current object does not fit in the bin being packed.
Best-Fit(BF)	Assign each object sequentially to the bin into which it fits such that the remaining empty space is minimized.
Worst-Fit(WF)	Assign each object sequentially to the bin into which it fits such that the remaining empty space is maximized.
FFD, BFD, WFD	exactly the same as the FF, BF and WF algorithms respectively, except that the data is sorted in decreasing (or non-increasing) order before they are packed.

Table 1: Basic Bin-Packing Heuristics

We believe that the reason for the increased error in [BJ83] results from the fact that empty space is not the only important factor for measuring performance. In fact, our study reveals that the distribution of the sizes of objects also affects the error. In this paper, we develop an $O(n \log n)$ lower bound algorithm for CBP and DBP, and an $O(n + L \log L)$ lower bound algorithm for DBP. Our work considers $s_i \in (u, v]$, where $0 \leq u \leq v \leq L$. Simulation data using BFD algorithm shows that our method provides an equal or better estimate of the optimal solution than the sum. Additionally, our approach leads to a polynomial time algorithm that finds an optimal solution for some special non-trivial bin-packing problems, and BFD provides an optimal solution for these special problems.

The rest of this paper is organized as follows. In Section 2, we introduce the basic terminology and theorems used in this paper. In Section 3, we introduce a special class of the bin-packing problem and provide a greedy lower bound for that subproblem. In Section 4, algorithms for calculating a tighter lower bound are presented. In Section 5, we demonstrate the impact of this lower bound technique by simulating the BFD algorithm on sets of objects with sizes distributed over different intervals $(u,v]$ of $(0,L]$ for $0 \leq u \leq v \leq L$. Finally we draw conclusions in Section 6. Frequently used symbols in this paper are listed in Appendix A for convenience.

2 BASIC TERMINOLOGY AND THEOREMS

In this section, we describe some basic terminology that is used in this paper. All definitions and theorems also apply to CBP, if L=1 with the sizes of objects being real number in $(0,1]$. First, we

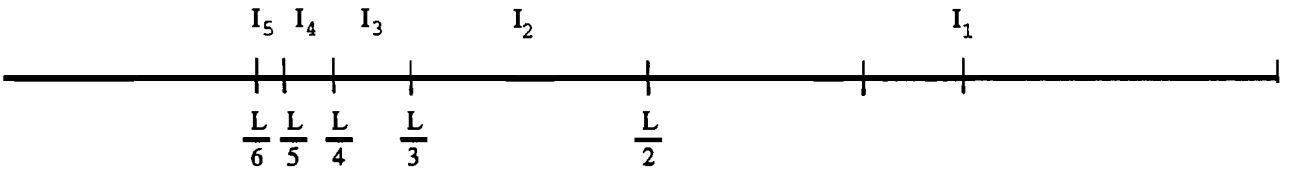


Figure 2: Harmonic Partition

introduce terminology for categorizing objects into classes based on sub-intervals of (0,L]. Second, we introduce the definition for approximate error rate. Finally, we define our new lower bound for the bin-packing problem.

2.1 Partitioning the Set of Objects

Many researchers have utilized subintervals to study the bin-packing problem. Yao [Yao80] classified objects and bins into four classes, $(0,\frac{1}{3}],(\frac{1}{3},\frac{2}{5}],(\frac{2}{5},\frac{1}{2}],(\frac{1}{2},1]$. Each object was first examined to determine its class and then assigned to a bin corresponding to the same class in a First-Fit fashion. C.C. Lee and **D.T.**Lee [LL85] proposed a Harmonic algorithm which partitions the interval (0,1] into M subintervals, $I_k = (\frac{1}{k+1},\frac{1}{k}]$, if $1 \leq$ k $<$ M and $I_M = (0,\frac{1}{M}]$. An object i is called an I_k-piece, if $s_i \in I_k$. A bin designated to pack I_k-pieces exclusively is called an I_k-bin. For each incoming object i, if it is an I_k-piece, then the algorithm places it in a non-full I_k-bin. Hence, typically k I_k-pieces are packed in an I_k-bin. We have modified Lee's definition of harmonic partition slightly to accommodate **DBP:**

Definition 1 (Harmonic Partition) A harmonic partition is the partition of the interval $(0,L]$ into harmonic intervals I_k, k = 1, 2, ..., where

$$I_k = \{x : \frac{L}{k+1} < x \leq \frac{L}{k}, x \in \mathcal{N}\}$$

is called the k-th harmonic interval. For $K \subseteq \mathcal{N}$,

$$I_K = \bigcup_{k \in K} I_k$$

Example 1 If L=100, then the harmonic intervals are I_1=(50..100], I_2=(33..50], I_3=(25..33], and $I_{2,3}$=(25,50] is the union of I_2 and I_3.

Note that x $\in I_k \iff$ k $= \lfloor L/x \rfloor$. The concept of harmonic partition is attractive because of the fact that exactly k I_k-pieces can be packed into a single bin.

Lemma 1 *Exactly* k I_k*-pieces* can be placed in the same bin.

Proof: It follows directly from the definition. □

Corollary 1 Each I_1*-piece* has to be assigned to an individual bin.

Corollary 2 If the set of objects to be packed consists of only I_k*-pieces,* then it is optimal to pack k objects together.

Proof: Lemma 1 implies that k and at most k objects can be packed together. If there are n I_k-pieces, then packing every k I_k-pieces in a bin requires $\lceil \frac{n}{k} \rceil$ bins, which is a lower bound on the optimal solution; hence, it is optimal. □

In this paper, we subdivide objects into several classes in order to calculate a tighter lower bound for the bin-packing problem. The harmonic partition is extremely useful, but is not general enough for our purposes. Let X and Y be arbitrary subintervals of (0,L]. An object i is called an X-piece, if $s_i \in$ X (a Y-piece is defined similarly). The complement of a subinterval X is defined as $X = (0, L] - X$. $|X|$ indicates the number of X-pieces. A bin is called an (X,Y)-bin if it is composed of an X-piece and a Y-piece. This notation can be extended to arbitrary tuples. A bin-packing problem is denoted as BP(X), if all objects are X-pieces. Another useful fact for calculating tighter lower bound follows:

Lemma 2 At most one object in $I_{2,3}$ can be packed with an I_1*-piece.*

Proof: Suppose i and j are $I_{2,3}$-pieces that can be packed with an I_1-piece k, then $s_i + s_j + s_k > \frac{L}{4} + \frac{L}{4} + \frac{L}{2} = L$, which contradicts the capacity constraint. □

2.2 Approximate Error Rate

Let $S^A(I)$ be the number of bins used by algorithm A and $S^*(I)$ be the minimal number of bins needed for packing a set of objects I. The approzimate error *rate* for an algorithm A using *lb* as an estimate of the optimal solution is defined as:

$$r_A(lb) = \frac{S^A - lb}{lb} = (\frac{S^A}{lb} - 1) * 100\% \qquad (1)$$

We define the approximate error rate to evaluate the optimality of the solutions provided by BFD (see Section 5).

Lemma 3 $r_A(lb)$ is an upper bound on the actual error rate.

Proof: The actual error rate $= \frac{S^A - S^*}{S^*} = \frac{S^A}{S^*} - 1 \leq \frac{S^A}{lb} - 1 = r_A(lb)$

2.3 Lower Bound Calculation

The following lemma is useful for determining a tight lower bound for optimization problems and plays a central role in our development of a new lower bound for bin-packing problems. For a bin-packing problem instance, I, a subset **I'** of I is called a subproblem of I, and I is referred as the original problem.

Lemma 4 (Subproblem Principle) The optimal solution of a subproblem is a lower bound for the original problem. The lower bound of a subproblem is a lower bound for the original problem. That is, $\mathbf{I'} \subseteq \mathbf{I} \Longrightarrow lb(I') \leq S^*(I') \leq S^*(I)$.

Proof: By definition, $lb(I') \leq S^*(I')$. $S^*(I') \leq S^*(I)$ because $\mathbf{I'} \subseteq \mathbf{I}$. □

Obviously, $\mathbf{I'} = \{i \in \mathbf{I} : \mathrm{L}/4 < s_i \leq \mathrm{L}) \subseteq \mathrm{I}$; hence, $lb(I') \leq S^*(I)$ and provides a lower bound on $S^*(\mathrm{I})$. A greedy algorithm is given in the next section to determine a lower bound for the subproblem that contains only $I_{1,2,3}$-pieces of the original problem. This lower bound, in turn, is a lower bound for the original problem.

Next, we introduce our lower bound for a bin-packing problem.

Definition 2 (Lower Bound) $LB(u)$ is defined as $max\{SUM, BIG(u)\}$, where $\mathrm{SUM} = \lceil \sum_{i \in I} s_i/L \rceil$ and $BIG(u)$ is a lower bound on $S^*\{i \in \mathbf{I} : s_i > \mathrm{u})$.

Note that our lower bound, $\mathbf{LB(u)}$, is defined to be the maximum of two terms, which are themselves lower bounds. The definition of SUM is modified slightly to accommodate DBP. By corollary 1, each I_1-piece needs an individual bin. Hence, a good initial lower bound is $\max\{SUM, BIG(M)\}$, where $BIG(M) = |I_1|$. $\mathbf{LB(u)}$ is more precise, if u is small; however, it is more difficult to determine for $\mathrm{u} < \frac{L}{4}$. $\mathbf{LB(u)}$ is a lower bound on S^* as the following lemma shows:

Lemma 5 $\mathrm{S}^* \geq LB(u)$.

Proof: Each bin has a capacity of L. Hence,

$$S^* * \mathrm{L} \geq \sum s_i \Rightarrow \mathrm{S}^* \geq \sum s_i/L \Rightarrow S^* \geq \lceil \sum s_i/L \rceil = \mathrm{SUM}$$

$\mathrm{S}^* \geq BIG(u)$ by the subproblem principle. It follows that $\mathrm{S}^* \geq \max\{SUM, BIG(u)\}$, which is used as our lower bound, $\mathbf{LB(u)}$. □

3 BP($I_{1,2,3}$) and OPTIMAL MATCHING

We know that $|I_1|$ provides a good lower bound. However, if we consider I_2 and I_3 pieces, a tighter lower bound is obtained. To calculate $BIG(\frac{L}{4})$, we consider a special class of bin-packing problems,

$BP(I_{1,2,3})$ which contains only I_1, I_2, and I_3 pieces.

An object i is said to be ***absorbed*** by an I_1-piece, j, if i and j are assigned to the same bin. We say that a set of objects $X = \{x_1 \ldots x_m\}$ is absorbable, if there exist distinct I_1-pieces $Y=\{y_1 \ldots y_m\}$ such that Vi, x_i can be absorbed by y_i, and $M \equiv (X, Y) = \{(z;, y_i) : 1 \leq i \leq m)$ forms a matching. Note that each I_1-piece consumes one bin (at a cost of 1) and at most one $I_{2,3}$-piece can be absorbed by an I_1-piece without requiring a new bin (at a cost of 0). A greedy algorithm matches $I_{2,3}$-pieces to some I_1-piece, if possible, such that the remaining $I_{2,3}$-pieces consume a minimum number of bins (which are referred as extra bins) in addition to those required by I_1-pieces. Let LEFT contain all $I_{2,3}$-pieces and RIGHT contain all I_1-pieces. For each $i \in$ LEFT, $N(i)$ denotes the set of I_1-pieces that are compatible with i. For any absorbable set X, $f(X) \equiv S^*(LEFT - X)$ is called the cost of X. Note that f(X) represents the minimum extra bins required by the remaining $I_{2,3}$-pieces after absorption. The total number of bins used by an optimal packing is given by the following equation:

$$S^*(I) = |RIGHT| + \min\{S^*(LEFT - X) : X \text{ is an absorbable subset of LEFT}) \quad (2)$$

$$= |RIGHT| + S^*(LEFT - X^*) \quad (3)$$

$$= |RIGHT| + f(X^*) \quad (4)$$

where X* is an absorbable subset that achieves minimum cost. Note that X* is maximal, otherwise it can be enlarged to eliminate more objects in LEFT. If X* is absorbed by Y*, then $M^* \equiv (X^*, Y^*)$ is said to be an optimal matching for $BP(I_{1,2,3})$. An algorithm that finds an optimal matching is given as follows:

[Absorb(K)]

1. sort K into nonincreasing order by size
2. $X, Y \leftarrow 0$
3. for each object i in K
   (a) if $N(i) \neq \emptyset$,
   let j be the largest object in $N(i)$
   $X \leftarrow X + i$, $Y \leftarrow Y + j$, $I_1 \leftarrow I_1 - j$.

Theorem 1 (Optimal Matching) *Absorb($I_{2,3}$)* finds an optimal matching for *BP($I_{1,2,3}$)* in O(n log n) time.

Proof: Let $M^* \equiv (X^*, Y^*)$ be an optimal matching (hence, it must be maximal). Consider the currently largest object i in LEFT.

1. If $N(i) = 0$, then there is no compatible I_1-piece for i; hence, $i \notin X^*$.

2. If $N(i) \neq \emptyset$ and j is the largest element in N(i), we claim that $(i, j) \in M^*$. It is proven by contradiction and by swapping objects in bins without increasing the cost.
   (a) First we prove that i belongs to an optimal matching. Suppose $i \notin X^*$.
      - If $j \notin Y^*$, then (i, j) can be added to M*, which contradicts the fact that M* is maximal.
      - If $j \in Y^*$, then $\exists k \in X^*$ absorbed by j (i.e. $(k, j) \in M^*$). Then $M = M^* - (k, j) + (i, j)$ is also optimal.
   (b) Next we show that i is absorbed by j. Assume i is absorbed by $l \neq j$ (i.e., $(i, 1) \in M^*$). Note that $l \preceq j$ because j was the largest in $N(i)$.
      - If $j \notin Y^*$, Then $M = M^* - (i, l) + (i, j)$ is also optimal.
      - If $j \in Y^*$, then $3k \in X^*$ absorbed by j. Then $M = M^* - (i,l) - (k,j) + (i,j) + (k,l)$ is also optimal[2].

   Hence, there exists an optimal matching that contains (i, j).

By continuing the above argument for the currently largest object in LEFT, we can decide whether it should be included in an optimal matching or not and which I_1-piece it should be matched to. It follows that Absorb(LEFT) finds an optimal matching for $BP(I_{1,2,3})$. Because each $I_{2,3}$-piece requires $O(\log n)$ time to find the largest compatible I_1-piece, the time complexity for Absorb($I_{2,3}$) is $O(n \log n)$. □

Corollary 3 *If (X^*, Y^*) is an optimal matching for $BP(I_{1,2,3})$, it is optimal to pack each $x_i \in X^*$ with the corresponding $y_i \in Y^*$.*

Proof: Because (X*, Y*) is an optimal matching, $f(X^*) \leq f(X)$ for any absorbable set $X \subseteq$ LEFT. Suppose $X \subseteq$ LEFT is absorbed by $Y \subseteq$ RIGHT in an optimal packing P. Then $S^* = |RIGHT| + f(X)$ by equation 4. Let Q be a packing with X* absorbed by Y*, then $f(X^*) \leq f(X) \Rightarrow |RIGHT| + f(X^*) \leq |RIGHT| + f(X) = S^*$. So, Q is also an optimal packing. □

Corollary 4 An optimal solution is obtained in $O(n \log n)$ for $BP(I_{1,2})$.

Proof: Note that the problem is a special case of $BP(I_{1,2,3})$. Hence, Absorb(LEFT) finds an optimal matching (X*, Y*) for the problem in O(n log n) time by theorem 1. By corollary 3, it is optimal to pack each $x_i \in X^*$ with the corresponding $y_i \in Y^*$. Because the sizes of all objects are greater than L/3, the objects remaining in LEFT are all I_2-pieces after absorption. Packing any two of the objects remaining in LEFT is optimal by corollary 2. □

A similar result holds for $BP(I_{1,3})$, except that three of the remaining I_3-pieces should be packed in a bin after absorption.

[2] k, l are compatible because $s_k + s_l \leq s_k + s_j \leq L$.

4 A NEW LOWER BOUND

In Section 3, we have given an algorithm for absorbing a maximal subset of $I_{2,3}$-pieces such that the remaining $I_{2,3}$-pieces require a minimum number of extra bins in addition to those bins consumed by I_1-pieces. The following lemma is used for calculating a lower bound on the minimum number of bins required by the remaining $I_{2,3}$-pieces.

Lemma 6 ***Consider $BP(I_{2,3})$, which contains only $I_{2,3}$-pieces. Assume i and j are the smallest two objects and $s_i = a, s_j = \beta, a \leq \beta$. Let*** $Z = \begin{cases} [\alpha, M] & if\ \alpha > \frac{L}{3} \\ (L - \alpha - \beta, M] & \text{if } \frac{L}{4} < \alpha \leq \frac{L}{3} \end{cases}$

Then it is optimal to pack two 2-pieces together.

Proof: For $BP(I_{2,3})$, at most three objects can be packed together because the size of each object is greater than $\frac{L}{4}$. However,

1. if $a > \frac{L}{3}$, then all objects are I_2-pieces; hence, it is optimal to pack two objects together by corollary 2.
2. if $\frac{L}{4} < a \leq \frac{L}{3}$, then $Z = (L - a - \beta, M]$. Hence
   - Each Z-piece can be packed with at most one object in a bin. Assume that k is a Z-piece that can be packed with two objects *l* and m in a bin, then $s_k + s_l + s_m \geq s_k + \alpha + \beta > L$ (because **i** and ***j*** are the smallest two objects and k is a Z-piece), which contradicts the capacity constraint. Hence, k cannot be packed with two other objects in a bin.
   - If there were two $(Z, \bar{Z})$-bins[3] in an optimal packing, then the objects in those two bins can be exchanged such that the two Z-pieces are in the same bin.

Example 2 If $a = 27, \beta = 30$, then Z = (43,501. Let k be a Z-piece, then $s_k \geq 44$ and $s_k + \alpha + \beta \geq 44 + 27 + 30 = 101 > 100$.

Figure 3 shows how the set of $I_{1,2,3}$-pieces are partitioned into disjoint subsets. (X*, Y*) is an optimal matching and Z is the set of Z-pieces. $BIG(\frac{L}{4})$ is calculated based on packing each $x_i \in X^*$ with the corresponding $y_i \in$ Y*, then packing two Z-pieces together.

4.1 Calculation of BIG

We now calculate an estimate of the number of mutually incompatible objects, BIG $\equiv |I_1| + \max\{m_1, m_2\}$, as follows:

[3] A (Z, $\bar{Z}$)-bin is a bin containing one Z-piece and one $\bar{Z}$-piece.

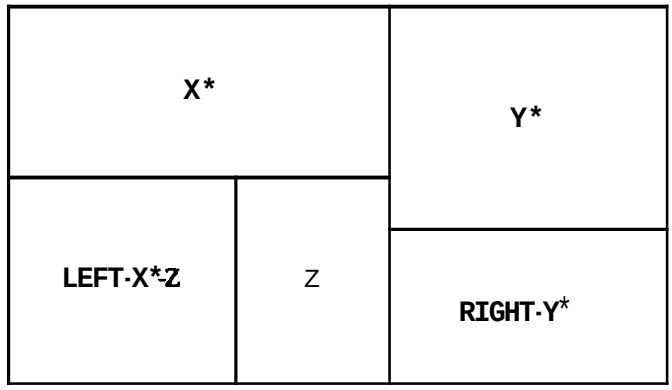


Figure 3: The set of $I_{1,2,3}$-pieces are partitioned into X*, Y*, Z, LEFT $-$ X* $-$ Z, RIGHT $-$ Y*, where (X*, Y*) is an optimal matching and Z is the set of 2-pieces.

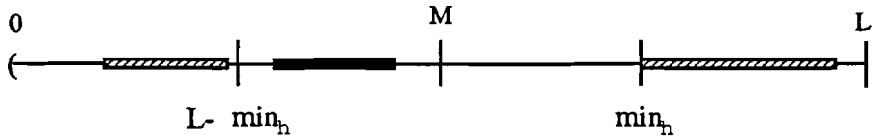


Figure 4: The region that cannot be absorbed and requires additional bins

1. $|I_1|$ bins are needed for I_1-pieces.
   It follows directly from corollary 1.
   Next, we discuss the extra bins needed in addition to those bins required by I_1-pieces.

2. $m_1 = \max\{\lceil \frac{A_k}{k} \rceil : 2 \le k \le k_m\}$, where
   min_h= the size of the smallest I_1-piece and $k_m = \lfloor \frac{L}{L - min_h + \epsilon} \rfloor$ (E = the smallest expressible number for CBP and $\epsilon = 1$ for DBP). None of the (L $-$ min_h, MI-pieces (whose sizes are in the black section in Figure 4) can be packed with any I_1-piece. They must consume extra bins. Let

$$A_k = |I_2| + \ldots + |I_k|, \ 2 \le k \le k_m$$

   ($|I_{k_m}|$ = number of I_{k_m}-pieces with sizes $>$ L $-$ min_h). Note that A_k represents the number of ($\frac{L}{k+1}$, MI-pieces, which require at least $\lceil \frac{A_k}{k} \rceil$ bins because of the capacity constraint. Hence, at least m_1 extra bins are needed for the (L $-$ min_h, MI-pieces.
   Example 3 Suppose min_h=80, i.e., the size of each I_1-piece $\ge$ 80, then $k_m = \lfloor \frac{100}{21} \rfloor$ = 4. None of the (20,501-pieces can be packed with any I_1-piece. Assume $|I_2|$=10, $|I_3|$=20, $|I_4|$=30, then A_2 = 10, A_3 = 30, A_4 = 60. Note that there are A_2 = 10 objects whose size $>$ 33, A3 = 30 objects whose size $>$ 25, A_4 = 60 objects whose size $>$ 20. So, at least $\max\{\lceil \frac{10}{2} \rceil, \lceil \frac{30}{3} \rceil, \lceil \frac{60}{4} \rceil\} = \max\{5, 10, 15\} = 15$ bins are required for these objects.

3. $m_2 = \lceil \frac{z}{2} \rceil + \max\{\lceil \frac{y}{2} \rceil, \lceil \frac{x+y}{3} \rceil\}$ [4],
where $x = |I_3|, y = |I_2| - |Z|$ and z = $|Z|$ after absorption. Ignoring objects whose sizes are $\leq L/4$, the problem becomes $BP(I_{1,2,3})$.

- **Absorption**
  Theorem 1 shows that $Absorb(I_{2,3})$ finds an optimal matching for $BP(I_{1,2,3})$ and it is optimal to merge (x_i, y_i) in the matching by corollary 3.
- **Z-pack**
  Assume i and j are the smallest two $I_{2,3}$-pieces after absorption and $s_i = a, s_j = \beta$ with $\alpha \leq \beta$. Let $Z = (L - \alpha - \beta, M]$, if $\frac{L}{4} < \alpha \leq \frac{L}{3}$ and $Z = [\alpha, M]$, if $\alpha > f$. By lemma 6, it is optimal to pack two Z-pieces together, requiring at least $\lceil z/2 \rceil$ extra bins where $z = |Z|$.
- For the remaining $I_{2,3}$-pieces (after absorption and packing of Z-pieces), at most three of them can be packed together. If there are x I_2-pieces and y I_3-pieces left, then at least $\max\{\lceil \frac{x}{2} \rceil, \lceil \frac{x+y}{3} \rceil\}$ are required for these objects, which is similar to the calculation of m_1.

So, at least m_2 extra bins are needed to pack the $I_{1,2,3}$-pieces in the set, and m_2 is a lower bound for the original set by lemma 4 (the subproblem principle).

BIG is used in the two algorithms we have developed for determining the lower bound for any instance of bin-packing. The first algorithm is applicable for both CBP and DBP, while the second assumes the discrete case to achieve faster performance.

4.2 Algorithm 1

Algorithm 1 in Figure 6 follows directly from previous discussions. It classifies objects into I_k-pieces, sorts I_1, calculates SUM, and then calculates BIG using the methods discussed in Section 4.1. Consider the time complexity of algorithm 1. Sorting objects in I_1 and $I_{2,3}$ requires $O(n \log n)$. Additionally, calculation of BIG requires $O(n \log n)$ time because each $I_{2,3}$-piece needs $O(\log n)$ time to find its largest compatible I_1-piece. Hence, the overall time complexity is $O(n \log n)$.

Theorem 2 ***Algorithm 1 gives a lower bound on the optimal solution of BP.***

Proof: The calculation of BIG follows directly from the arguments in Section 4.1, which is a lower bound on $S^*\{i \in I : s_i > \frac{L}{4}\}$, and hence, is a lower bound on S*. □

[4] $y \leftarrow y - 1$, if z is odd

[LB]

1. calculate **SUM**
2. calculate m_1
   (a) Let min_h= the size of the smallest I_1-piece, $k_m = \lfloor \frac{L}{L-min_h+\epsilon} \rfloor$
   (b) count $|I_k|, 1 \leq k \leq k_m$
   ($|I_{k_m}|$ = number of I_{k_m}-pieces with sizes $> L - min_h$).
   (c) calculate $A_k = |I_2| + \ldots + |I_k|$, $2 \leq k \leq k_m$
   (d) $m_1 = \max\{\lceil \frac{A_k}{k} \rceil : 2 \leq k \leq k_m\}$
3. calculate m_2
   (a) Absorb($I_{2,3}$)
   (b) Assume i and j are the smallest two $I_{2,3}$-pieces left and $s_i = \alpha$, $s_j = \beta$, $\alpha \leq \beta$. Let $Z = \begin{cases} [\alpha, M] & if\ \alpha > \frac{L}{3} \\ (L-\alpha-\beta, M] & if\ \frac{L}{4} < \alpha \leq \frac{L}{3} \end{cases}$
   (c) $x = |I_3|$, $y = |I_2| - |Z|$, $z = |Z|$
   (d) $m_2 = \begin{cases} \lceil \frac{z}{2} \rceil + \max(\lceil \frac{y}{2} \rceil, \lceil \frac{x+y}{3} \rceil) & \text{if } z \text{ is even} \\ \lceil \frac{z}{2} \rceil + \max(\lceil \frac{y-1}{2} \rceil, \lceil \frac{x+y-1}{3} \rceil) & \text{if } z \text{ is odd} \end{cases}$
4. BIG = $|I_1| + \max\{m_1, m_2\}$
5. return $\max\{SUM, \text{BIG}\}$

Figure 5: An algorithm for calculating a lower bound in $O(n \log n)$ time.

[Algorithm 1]

1. classify objects into appropriate I_k groups
2. sort I_1
3. return **LB()**

Figure 6: Algorithm 1 provides a lower bound for DBP and CBP in $O(n \log n)$ time.

4.3 Algorithm 2

For Discrete Bin-Packing, the capacity of each bin is L and all sizes are integer numbers in (0,L]. Normally L $\ll$ n. In this section, we describe algorithm 2, a variation of algorithm 1, which takes advantage of this property. Its time complexity is $O(n + L \log L)$ which is faster in many applications. If we partition the set of objects into groups, much search effort can be eliminated by dealing with groups instead of individual objects. We define a ***bucket*** as a maximal subset of equal-sized[5] objects. The set of objects I is partitioned into disjoint buckets, B_k, $1 \leq k \leq L$, and $|B_k|$ is the number of objects in bucket B_k.

We say that a merge is perfect, if its content is precisely L. A perfect merge with k components is called a ***perfect-k-way merge.*** The following theorem shows that it is always optimal to do perfect-two-way merging, if possible. It is convenient to view a packing P as a partition of the set of objects I, where members of each subset are assigned to the same bin.

Theorem *3 Optimality is preserved under Perfect-two-way Merges.*
Proof: Let P be an optimal packing. Suppose P contains two subsets (bins) $(x_1 \ldots x_m)$ and $(y_1 \ldots y_n)$ with $s_{x_1}+s_{y_1}$=L. Note that:p

$$\sum_{i=1}^{m} s_{x_i} \leq L \Longrightarrow \sum_{i=2}^{m} s_{x_i} \leq L - s_{x_1} \tag{5}$$

$$\sum_{j=1}^{n} s_{y_j} \leq L \Longrightarrow \sum_{j=2}^{n} s_{y_j} \leq L - s_{y_1} \tag{6}$$

Let Q be another packing such that Q is the same as P except that Q contains bins (x_1,y_1) and $(x_2 \ldots x_m, y_2 \ldots y_n)$ (namely, swapping y_1 & $x_2 \ldots x_m$, see Figure 7). Q is valid because

- In Bin 1, $s_{x_1} + s_{y_1} = L$.
- In Bin 2, $\sum_{i=2}^{m} s_{x_i} + \sum_{j=2}^{n} s_{y_j} \leq 2L - s_{x_1} - s_{y_1} \leq L$.

Hence, Q is another optimal packing since it uses the same number of bins as P. Continuing the same argument, there exists an optimal packing by doing perfect-two-way merging before packing. a

Note that the pseudo objects formed by perfect-two-way merging have size L, which are I_1-pieces and are unable to absorb any $I_{2,3}$-pieces. This theorem is used in algorithm 2 to reduce the problem size whenever applicable.

[5] When L is large, this definition can be modified such that a bucket contains objects with sizes within some interval.

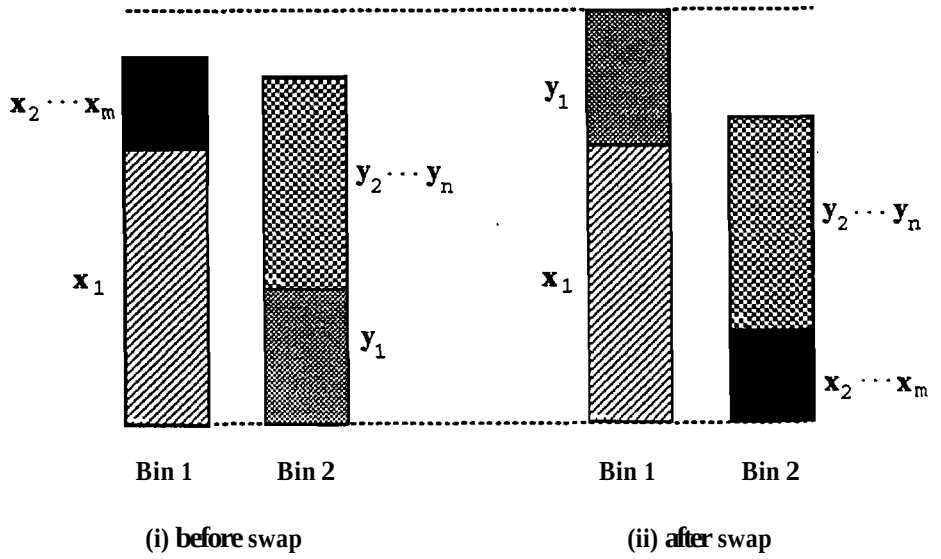


Figure 7: Swapping the contents of two bins where $s_{x_1} + s_{y_1} = L$.

[Algorithm 2]

1. classify objects into appropriate buckets
2. do all possible Perfect-Tuo-Way merges
3. return LBO

Figure 8: Algorithm 2 provides a lower bound for DBP in $O(n + L \log L)$ time.

Algorithm	Classification	Sorting	Calculate SUM	Calculate BIG	Overall Time	Memory
1	$O(n)$	$O(n \log n)$	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
2	$O(n)$		$O(n)$	$O(L \log L)$	$O(n + L \log L)$	$O(n + L)$

Table 2: Comparison of Algorithms 1 and 2

Algorithm 2 in Figure 8 further partitions the set of objects into L buckets. Although both algorithms call the same procedure LB, the implementation of LB can be quite different for CBP and DBP. Consider merging two buckets B_s and B_t, if $|B_s| = a, |B_t| = b, a < b$ before merging, then $|B_s| = 0, |B_t| = b - a, |B_{s+t}| = |B_{s+t}| + a$ after merging. Because we are only concerned about the number of objects in buckets, algorithm 2 does not require truly merging objects. Consider the time complexity of algorithm 2. The $I_{2,3}$-pieces are partitioned into $\frac{L}{4}$ buckets. For each $s \in (\frac{L}{4} - \frac{L}{2}]$, it requires $O(\log L)$ time to find the largest $t \leq L - s$. Hence, the time complexity of calculating BIG for DBP in algorithm 2 is reduced to O(L log L) which is independent of n.

Corollary *5 Algorithm 2 gives a lower bound on the optimal solution of DBP.*

Proof: Algorithm 2 is equivalent to algorithm 1 except that we deal with buckets in decreasing order. The algorithm also employs perfect-two-way merging to help reduce the problem size. Optimality is preserved by doing perfect-two-way merges by theorem 3. •

A comparison of time and space complexities of these two algorithms is shown in table 2. Asymptotically, they have the same performance when $L = O(n)$. Algorithm 1 has the advantage that it is not sensitive to the precision of L and requires less memory. Algorithm 2 is linear, if $n = O(L \log L)$, which holds for many applications.

SIMULATION ANALYSIS

We demonstrate the impact of this lower bound technique by simulating BFD algorithm on sets of n objects with sizes uniformly distributed over intervals (u,v] of (0,L] for $0 \leq u \leq v \leq L$. Four typical (u,v]-intervals are shown in Figure 9. The centerline is depicted as $s = M = \frac{L}{2}$. The shaded regions represent the portion that cannot undergo perfect two-way merging. The objects in this region cause SUM to fail as a good estimate of the optimal solution.

We examine the number of bins used (designated as SOL in the figures) and the approximate error rates of BFD algorithm using SUM and LB as estimates of the optimal solution. Recall the

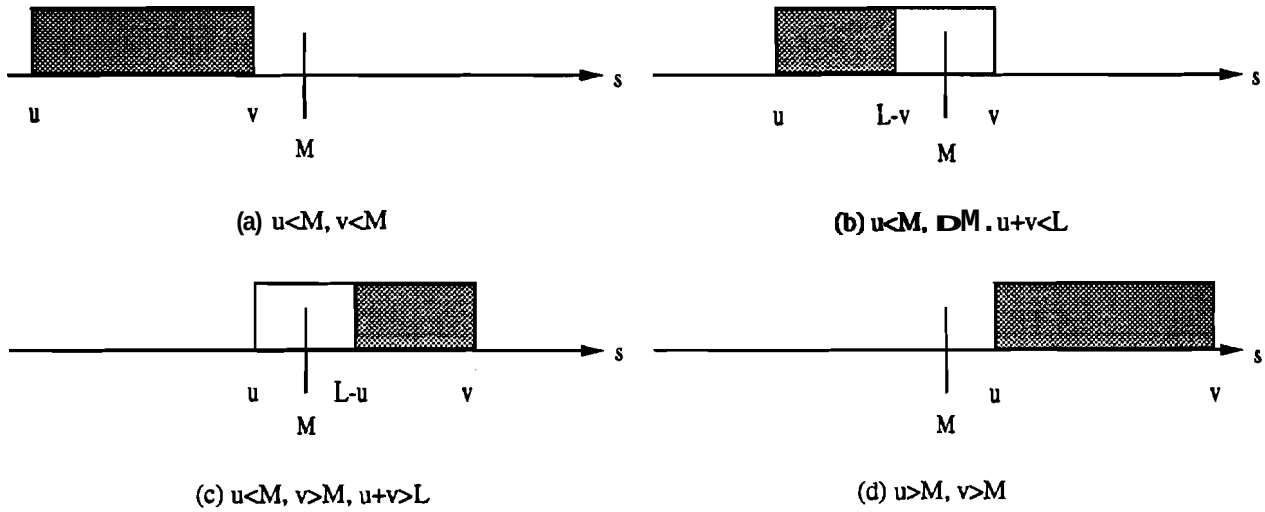


Figure 9: Four typical (u,v]-intervals. (a) u < M, v < M; (b) u < M, v > M, u + v < L; (c) $u < M, v > M, u + v > L$; and (d) $u > M, v > M$.

approximate error rate using lb as an estimate of the optimal solution is:

$$r(lb) = \frac{SOL - lb}{lb} = (\frac{SOL}{lb} - 1) * 100\%$$

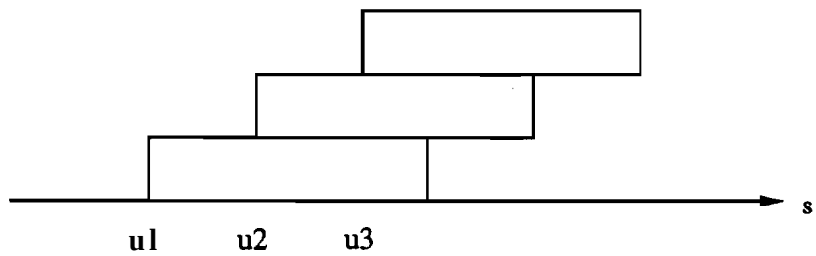
where SOL is the number of bins used by the algorithm.

Simulations were done on a SUN/4 SPARC workstation, using random, the random number generator provided by UNIX, with n = 30000 and L = 100. Ten random instances are packed for each parameter under consideration. We consider the following three cases:

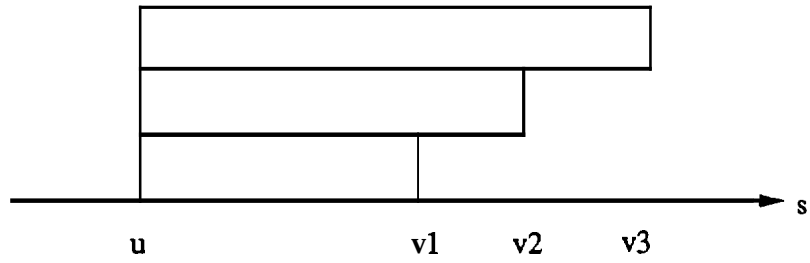
1. **fixed length** : $l = \frac{L}{k}$, k = 2...8. (results shown in Figures 11-14)
2. **fixed left bound** : $u = 0$ and $\frac{L}{k}$, k = 2...8. (results shown in Figures 15-18)
3. **fixed right bound** v = L and $(1 - \frac{1}{k})L$, k = 2...8. (results shown in Figures 19-22)

5.1 Simulation Results

Because the sizes of objects are numbers drawn from a uniform distribution in (u,v], SUM increases linearly as we move right on the horizontal axis. SUM is a good estimate of S*only when there are many small objects that can fill in the empty space of bins consumed by large objects. The curve of LB, though not shown, is easily constructed by taking the maximum of the two curves. SUM is the dominant component in LB when there are more small objects, while BIG is the dominant component when there are fewer small objects. For graphs in Figures 11, 12, 19, and 20, SUM



(a) fixed length l=v-u



(b) fixed left bound u

(c) fixed right bound v

Figure 10: [u,v]-intervals **considered in our simulation. (a) fixed length** l**, (b) fixed left bound u and (c) fixed right bound** v.

and BIG intersect at critical points where BIG becomes the dominant component in LB as we move to the right on the horizontal axis in the graphs. Observe that r(SUM) reaches its (global) minimum, when the (u,v]-interval is symmetric with respect to the center line, because of the uniform distribution.

Notice, in Figure 17, that BFD appears optimal when the left bound $u > \frac{L}{3}$, and we prove this in the following corollary.

Corollary 6 BFD *provides an optimal solution for* $BP(I_{1,2})$.

Proof: BFD sorts all objects by size in nonincreasing order into a queue Q. For each object i in Q, it then searches the ***used*** bins to find a bin whose empty space is closest to s_i. If there is not such a bin, a new bin is created and i is assigned to that bin. Note that placing an object i in a bin with content s is equivalent to merging i with a (pseudo) object j with $s_j = s$. Thus, BFD finds the largest compatible (pseudo) object j (a bin I with content s_j) for each object i in Q and merge (i,j) (place i in bin I). If we divide Q into two sub-queues R and L such that R contains all the I_1-pieces and L contains the remaining I_2-pieces, it is easy to see, from previous discussion, that BFD gives an optimal packing. □

Corollary 7 BFD *provides an optimal solution for* $BP(I_{1,3})$.

In the following sections, we describe the behavior of r(SUM) and r(LB) for various cases and give intuitive explanations of the phenomena.

5.1.1 Intervals with Fixed Length

In case 1, depicted in Figure 10(a), we consider intervals (u,v] of fixed length, $l = \frac{L}{k}, 2 \leq k \leq 9$. The simulation results are shown in Figures 11-14. For small u, SUM is the dominant component in LB, while BIG is the dominant component for large u. The gap between the two curves for small u is narrowed when l decreases. For $l = \frac{L}{2}$, the mean r(LB) drops to zero when $u > \frac{L}{4}$ (see Figure 13), which corresponds to the case in Figure 9(c), where (u,M)-pieces are almost completely absorbed by I_1-pieces; hence, $S^* \approx$ LB.

As we move the interval to the right on the graph, r(SUM) changes as follows (see Figures 13 and 14):

- r(SUM) is small for small u, where most of the empty space caused by large objects is filled by small objects.
- As u increases, r(SUM) increases to a local maxima when most objects are medium sized.

- Then r(SUM) starts to decrease because SUM increases and SOL remains almost unchanged (as can be seen by the middle "flat" portion of SOL in Figures 11 and 12).
- r(SUM) reaches its minimum value with SOL $\approx SUM, r(SUM) \approx 0$, when the interval is symmetric with respect to the center line.
- Passing this global minima, r(SUM) again increases because more I_1-pieces cannot be perfectly merged with other pieces, causing the empty space to increase. r(SUM) reaches the (global) maximum when $u = M$. At this point, all objects are I_1-pieces and the empty space is at a maximum.
- r(SUM) decreases as we continue to move the interval to the right along the *s* axis, because SUM increases while SOL = n remains unchanged.

The maximum and minimum values of r(SUM) and their corresponding u values for fixed length intervals are summarized in table 3. We form two conjectures about r(SUM) by examining the table and the figures. Conjecture 1 follows from the fact that r(SUM) is minimum when the interval is symmetric with respect to the center line, and conjecture 2 results because the maximum r(SUM) increases as the length of the interval decreases.

Conjecture 1 (Maximum and Minimum r(SUM)) *If the length l of $(u,v]$-interval is fixed, then the mazimum r(SUM) occurs at $u = M$ and the minimum r(SUM) occurs at* $u = \frac{L-l}{2}$.

Conjecture 2 *The maximum r(SUM) increases as the length of the interval decreases.*

Notice that when $u = M$, r(SUM) is maximum and SOL = n. As the length of the interval decreases, SUM decreases and $r(SUM) = \frac{n}{SUM} - 1$ increases.

5.1.2 Intervals with Fixed Left Bound

In case 2, depicted in Figure 10(b), we consider intervals with a fixed left bound, u = 0 and $\frac{L}{k}, 2 \leq k \leq 8$. The simulation results are shown in Figures 15-18. For $u = \frac{L}{2}$ and $\frac{L}{3}$, BFD is optimal by lemma 6, so r(LB)=0.

Before summarizing the results, we discuss the simulation, results for various values of u:

1. If $u = \frac{L}{2}$, all objects are I_1-pieces; hence, SOL=n and r(LB)=0. However, SUM increases linearly as v increases; hence, r(SUM) decreases linearly as v increases.
2. If $u = \frac{L}{3}$, then
   - if $v < \frac{2L}{3}$, r(SUM) decreases as v increases.

length (l)	maximum $r(SUM)$	u position	minimum $r(SUM)$	u position
$\frac{L}{2}$	33%	50	0%	24
$\frac{L}{3}$	50%	50	0%	33
$\frac{L}{4}$	58%	50	0%	37
$\frac{L}{5}$	65%	50	0%	39
$\frac{L}{6}$	70%	50	0%	41
$\frac{L}{7}$	73%	50	0%	42
$\frac{L}{8}$	77%	50	0%	43
$\frac{L}{9}$	79%	50	0%	44

Table 3: Maximum and minimum values of r(SUM) and their corresponding u positions for intervals with fixed length.

- if v = L − $u = \frac{2L}{3}$, r(SUM) reaches its minimum value.
- if v > $\frac{2L}{3}$, more I_1-pieces cannot be packed perfectly, so r(SUM) increases as v increases. Note that r(SUM) increases more slowly when v becomes large because the objects that waste space in the bins become large, and the empty space is small.

3. The curves for u = $\frac{L}{k}$, $4 \leq k \leq 9$ basically have the same shape except that the approximate error rate is small for small u. This happens because there are more small objects to fill the empty spaces in bins used by large objects. Hence, we consider only the case when u = $\frac{L}{4}$. r(SUM) reaches a minimum at v = L − u = $\frac{3L}{4}$. Also observe that $r(LB) \approx 0$ when $\frac{L}{4} < v \leq \frac{3L}{8}$ and v ≥ $\frac{5L}{8}$. Hence, we make the following conjecture:
   Conjecture 3 BFD is **optimal** *for* $BP(\frac{L}{4}, \frac{3L}{8})$.
   For $BP(\frac{L}{4}, \frac{3L}{8})$, $|I_3| \approx 2|I_2|$. Hence, when BFD first packs two I_2-pieces together and then fills the empty space with an I_3-piece (e.g., two objects with size 34 are packed first, then an object with size 32 is added), empty space is minimized.
4. If u = 0, both r(SUM) (< 0.35%) and r(LB) (< 0.04%) are small; an effect similar to the experimental study in [BJ83]. When v ≤ 92, there are more small objects than empty space; hence, bins are full and SUM provides a good estimate. When v > 92, there are more large objects, and there are not enough small objects to fill the empty space, causing the error to increase.

From these simulation results, depending on the relative position of v and L − u, we draw the following conclusions:

- v < L − u (corresponding to the case in Figure 9(b)): As v increases, more (u,M]-pieces perfectly absorbed by I_1-pieces, the shaded region decreases; hence, r(SUM) decreases.

- $v = L - u$: The interval is symmetric with respect to the center line; hence, $r(SUM) \approx 0$. The global minimum of r(SUM) occurs when v = L − u.

- v > $L-u$ (corresponding to the case in Figure 9(c)): Most (u,M]-pieces are perfectly absorbed by I_1-pieces; hence, SOL ≈ LB and $r(LB) \approx 0$.

5.1.3 Intervals with Fixed Right Bound

In case 3, depicted in Figure 10(c), we consider intervals with a fixed right bound, v = L and $(1 - \frac{1}{k})L$, k = 2...8. The simulation results are shown in Figures 19-22.

Below we summarize the simulation results for various values of v:

1. v = $\frac{L}{2}$. When $u = 0$, the interval is symmetric with respect to the line $\frac{L}{4}$. An object i can be packed with an object j, $s_j = \frac{L}{2} - s_i$, to form a pseudo object with size $\frac{L}{2}$. Then two of these pseudo objects can be packed perfectly. As u increases, more $(\frac{L}{4}, \frac{L}{2}]$-pieces cannot be packed perfectly; hence, r(SUM) increases and reaches its maximum value at $u = \frac{L}{3}$, where $S^* = \lceil n/2 \rceil$. Passing this maxima, r(SUM) decreases because SUM increases while SOL remains unchanged (see Figure 19 for v = $\frac{L}{2}$ and $u \geq \frac{L}{3}$).
2. The curves are similar for v = $(1 - \frac{1}{k})L, k > 2$, except that the maximum r(SUM) and the minima are different. Hence, we consider only v = $\frac{2L}{3}$. r(SUM) remains small and reaches the minimum at u = L − v = $\frac{L}{3}$. Passing this minima, the empty space starts increasing because more I_1-pieces cannot be packed perfectly. It reaches the maximum at u = $\frac{L}{2}$. Again r(SUM) decreases as u increases because SUM increases while SOL = n remains unchanged.

From the simulation results, depending on the relative position of v and L − u, we draw the following conclusions:

- u < L − v (corresponding to the case in Figure 9(b)): Most [L − v, M)-pieces are absorbed perfectly by the I_1-pieces. The remaining (u, L − v)-pieces are small objects that are typically packed perfectly; hence, $r(SUM) \approx 0$.

- u = L − v: The interval is symmetric with respect to the center line; hence, $r(SUM) \approx 0$. The minimum r(SUM) occurs at u = L − v.

- u > L − v (corresponding to the case in Figure 9(c)): In this case, most (u,M]-pieces are perfectly absorbed by I_1-pieces; hence, SOL ≈ LB and $r(LB) \approx 0$.

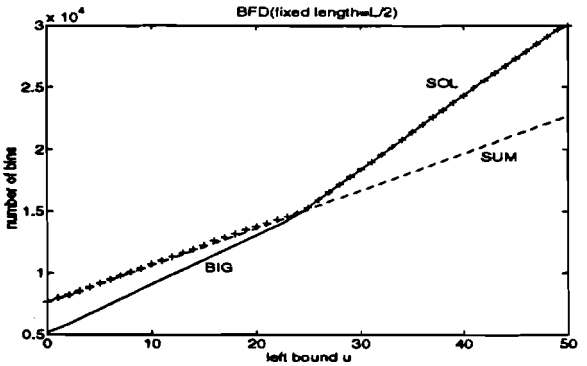
6 CONCLUSIONS

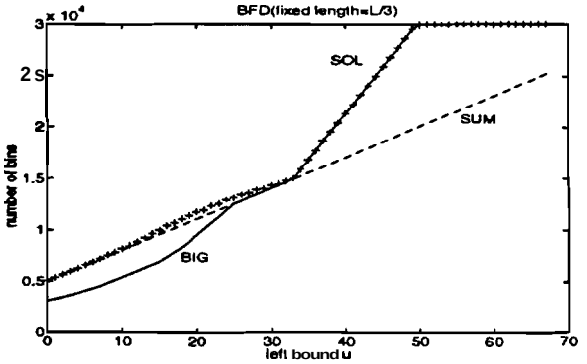
In this paper, we present an efficient technique for calculating a new lower bound, LB, on the minimum number of bins to pack a set of objects whose sizes range over (0,L]. It is tighter than the traditional lower bound, **SUM.** When more large and medium size objects exist, it gives a very good estimate on the optimal solution. As the approximate error rate becomes small, the uncertainty region of the decision problem becomes negligible. In many cases, the solution equals the lower bound, which means that the solution is optimal. It shows that a better estimate on the optimal solution of NP complete problems is important for assessing the quality of heuristic algorithms. In practical applications, we are satisfied with a solution which differs from the optimal solution within a small range of error.

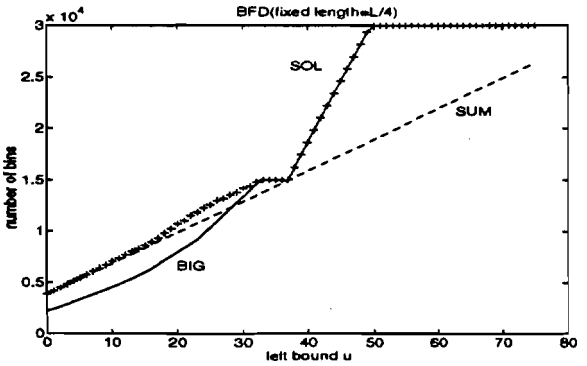
Further improvements can be achieved by considering $I_{4,5}$-pieces or determining an even tighter lower bound for packing $I_{1,2,3}$-pieces. It is also possible to use this lower bound technique to obtain a smaller worst case R(A) ratio.

A LIST OF SYMBOLS

$\mathcal{N}$	natural numbers
$\mathcal{R}$	real numbers
	number of objects
	the set of objects
L	capacity of a bin
M	L/2
s_i	size of object i
S^*	number of bins used by an optimal packing
S^A	number of bins used by heuristic algorithm A
SUM	$\lceil \sum_{i \in I} s_i / L \rceil$
BIG(u)	a lower bound on $S^*\{i \in \mathbf{I} : \mathrm{s}; > u\}$
LB(u)	max{SUM,BIG(u)}
lb	lower bound on S^*
$r_A(lb)$	$\frac{S^A}{lb} - 1$, approximate error rate
B_k	bucket k
$\|B_k\|$	number of objects in bucket k
(i,j)	pseudo-object composed of objects i and j
Z	any sub-interval of (0,L]
Z-piece	an object i with s; $\in Z$
$\|Z\|$	number of 2-pieces
BP(Z)	bin-packing problems in which all objects are 2-pieces
I_k	$\{x \mid \frac{L}{k+1} < x \leq \frac{L}{k}, x \in \mathcal{N}\}$
$\|I_k\|$	number of I_k-pieces
I_K	$\bigcup_{k \in K} I_k, K \subseteq \mathcal{N}$
$\|I_K\|$	number of I_K-pieces
$N(i)$	set of I_1-pieces that are compatible with i
$\approx$	approximately







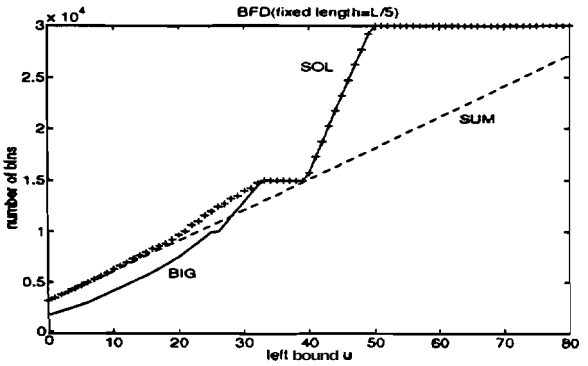
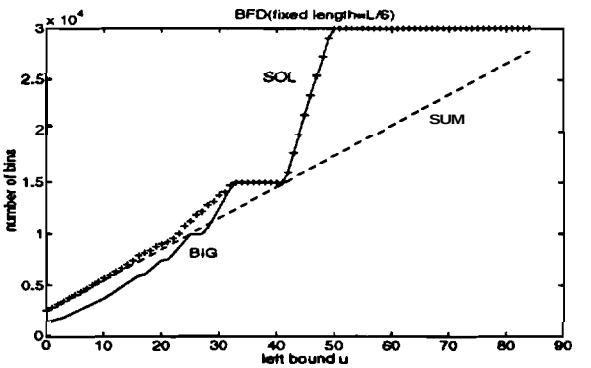
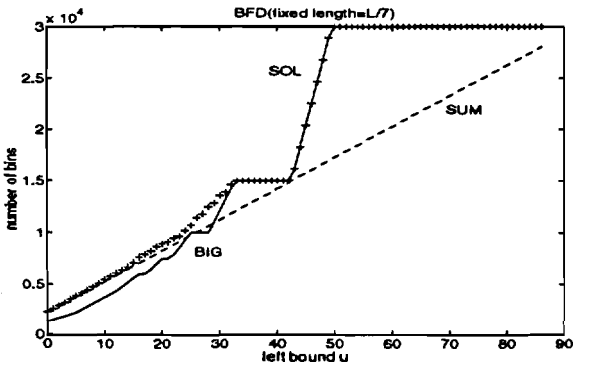
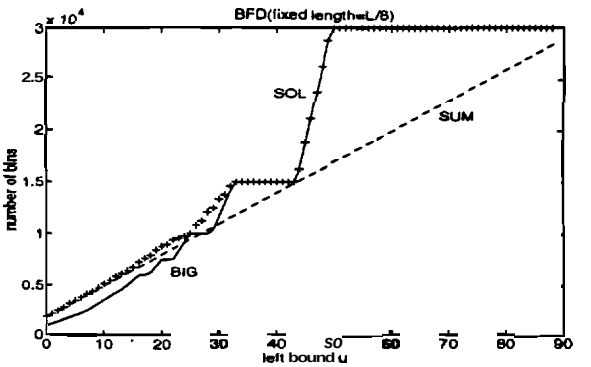


Figure 11: Solution vs lower bounds for fixed length $l = v - u = \frac{L}{2}, \frac{L}{3}, \frac{L}{4}, \frac{L}{5}, s_i \in (u, v]$







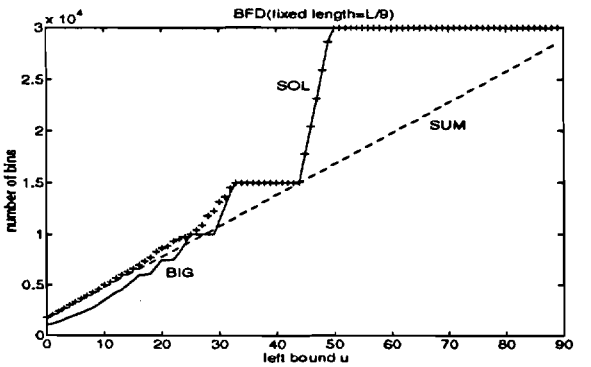
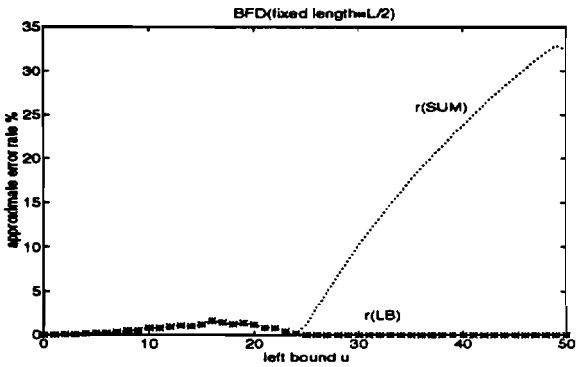
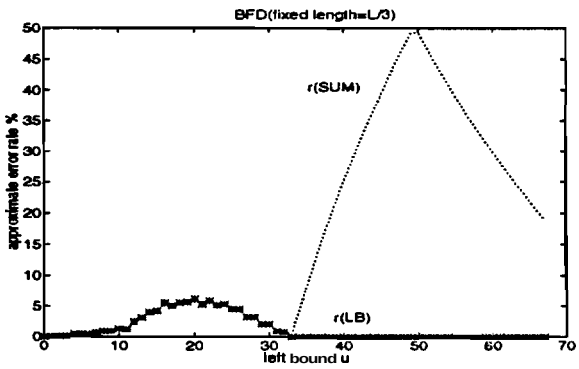
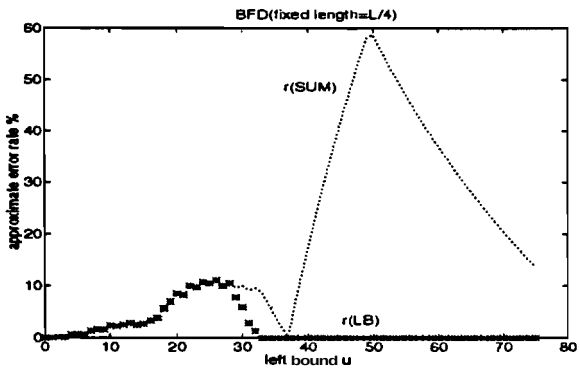


Figure 12: Solution vs lower bounds for fixed length $l = v - u = \frac{L}{6}, \frac{L}{7}, \frac{L}{8}, \frac{L}{9}, s_i \in (u, v]$







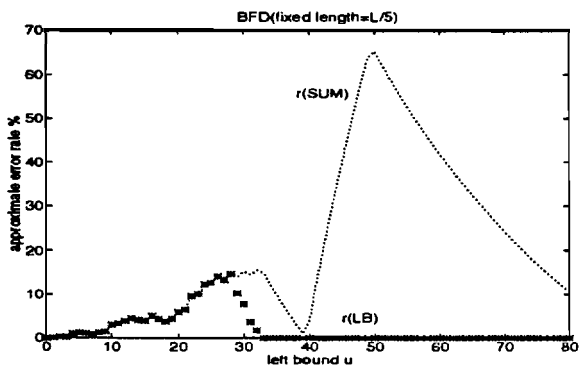
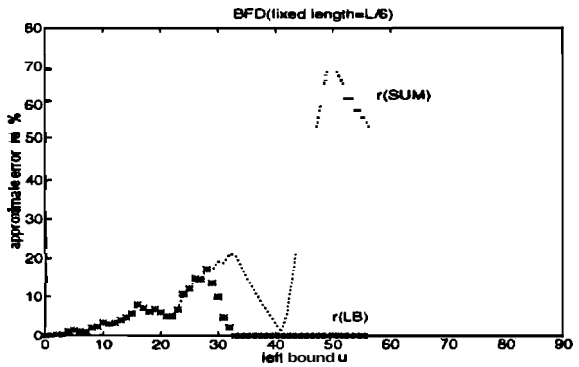
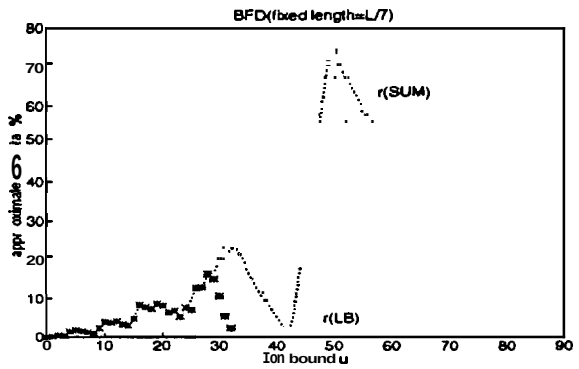
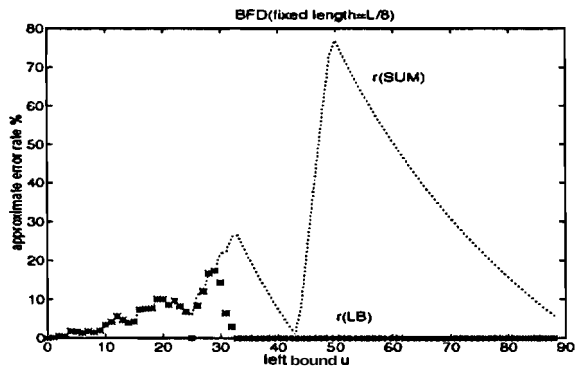


Figure 13: r(SUM) vs r(LB) for fixed length $l = v - u = \frac{L}{2}, \frac{L}{3}, \frac{L}{4}, \frac{L}{5}$, $s_i \in (u, v]$







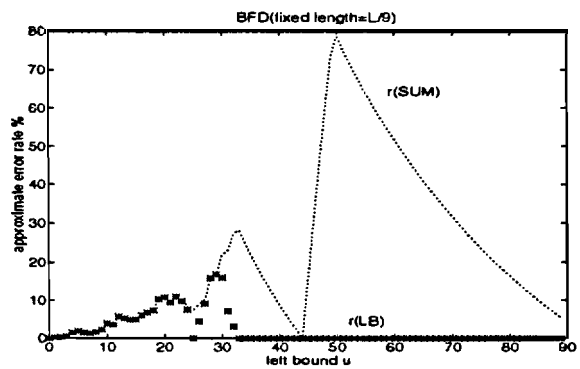
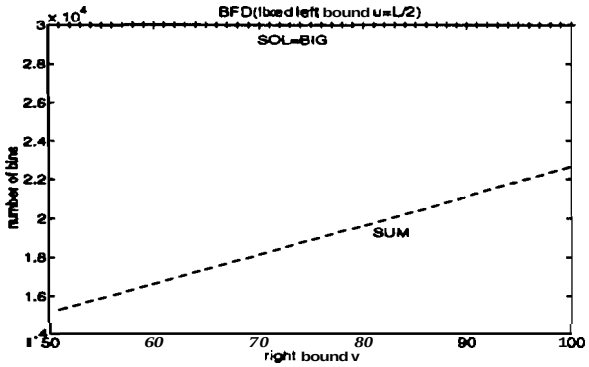
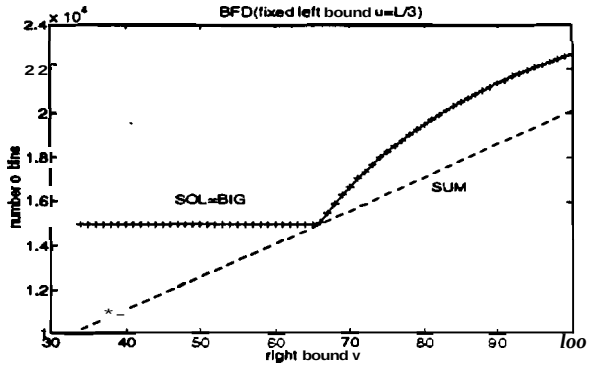
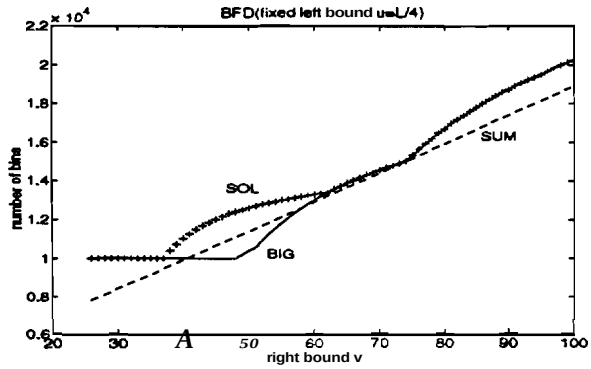


Figure 14: r(SUM) vs r(LB) for fixed length $l = v - u = \frac{L}{6}, \frac{L}{7}, \frac{L}{8}, \frac{L}{9}$, $s_i \in (u, v]$







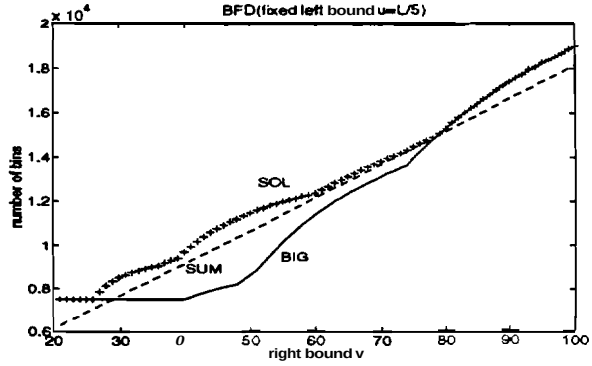
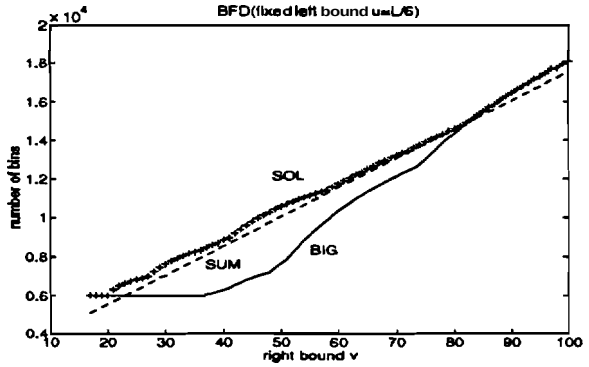
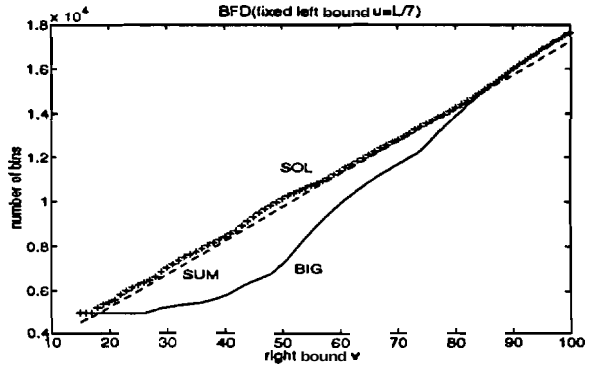
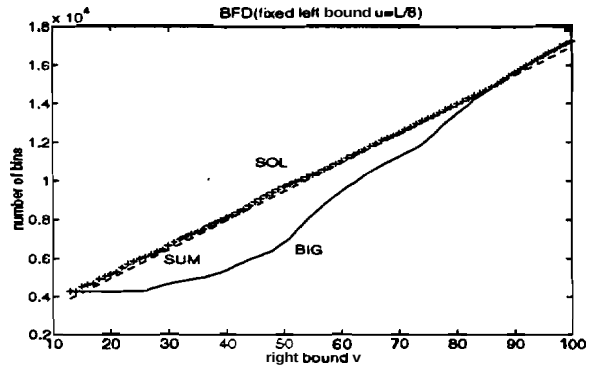


Figure 15: Solution vs lower bounds for fixed left bound $u = \frac{L}{2}, \frac{L}{3}, \frac{L}{4}, \frac{L}{5}$ $s_i \in (u, v]$







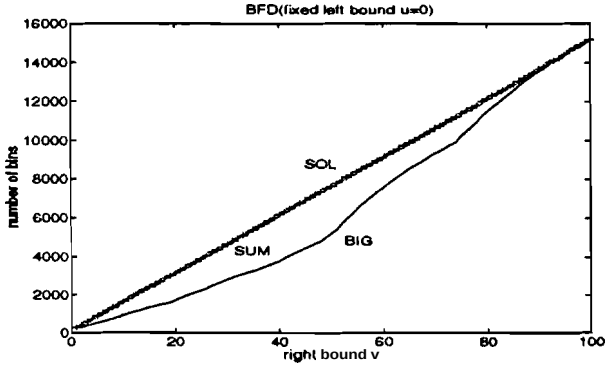
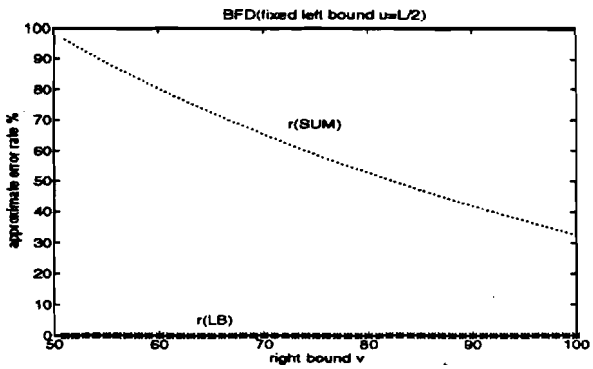
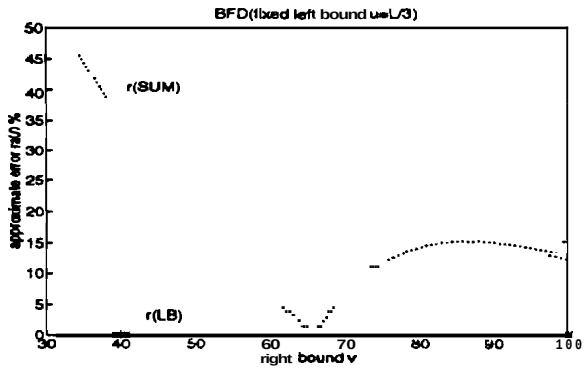
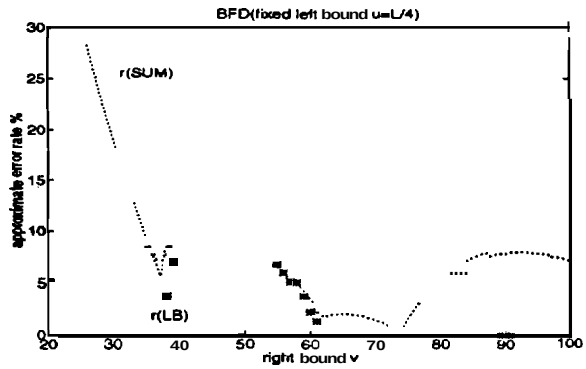


Figure 16: Solution vs lower bounds for fixed left bound $u = \frac{L}{6}, \frac{L}{7}, \frac{L}{8}, 0, s_i \in (u, v]$







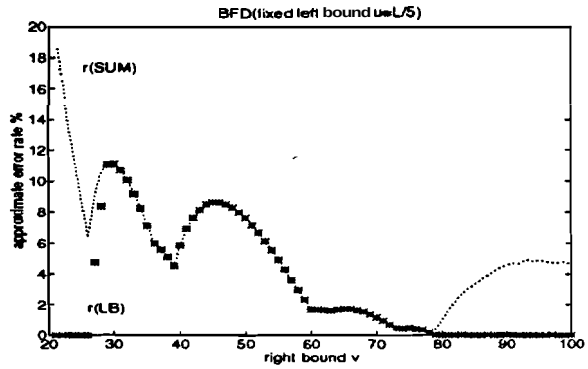
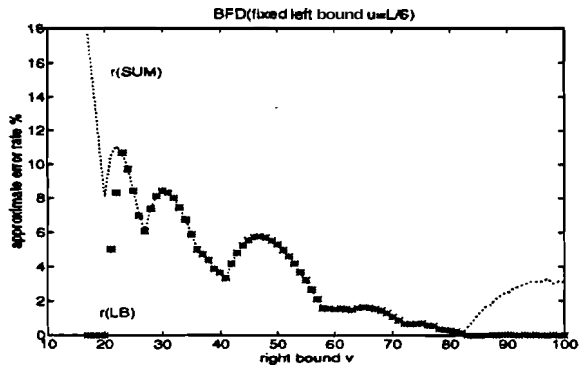
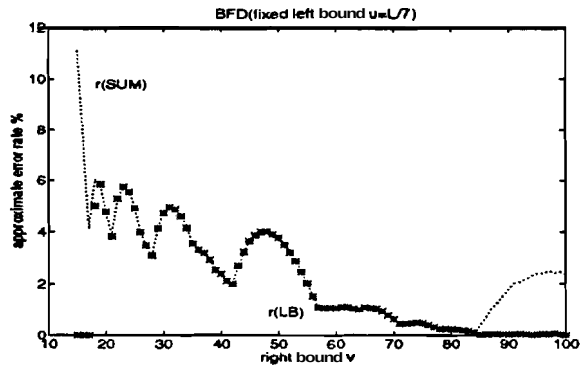
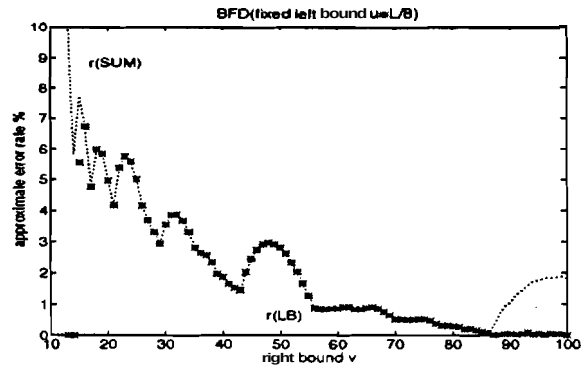


Figure **17:** $\mathrm{r(SUM)}$ vs $\mathrm{r(LB)}$ for fixed left bound $u = \frac{L}{2}, \frac{L}{3}, \frac{L}{4}, \frac{L}{5}$, $s_i \in (u, v]$







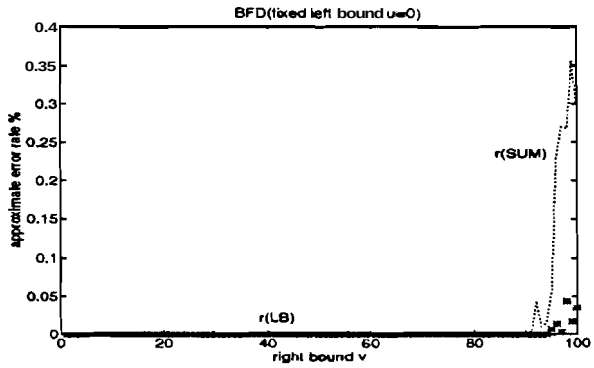
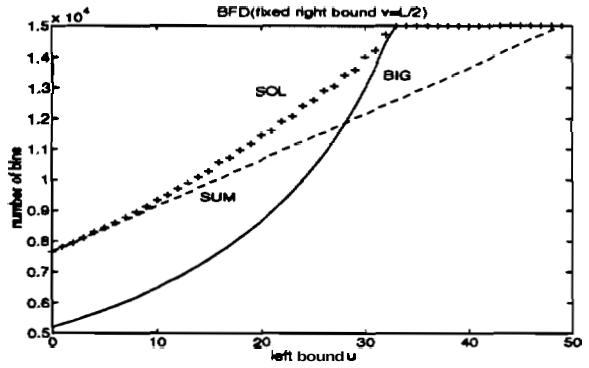
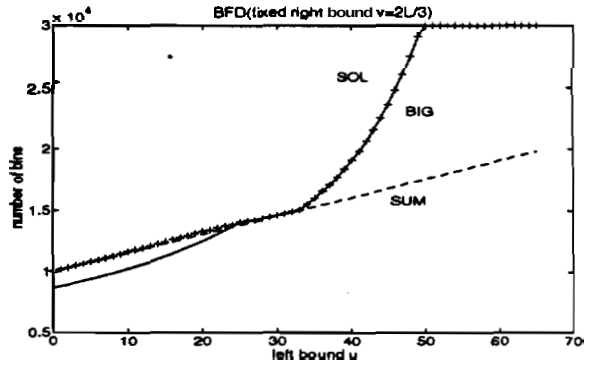
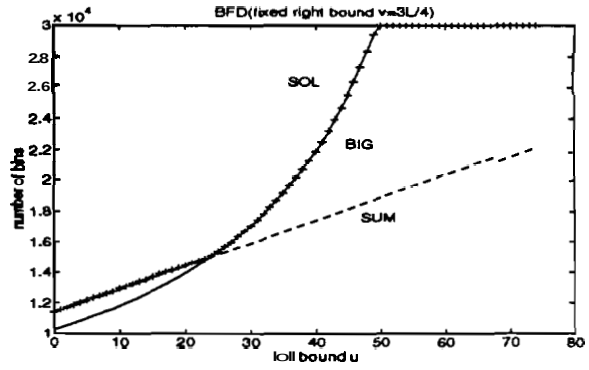


Figure **18:** $\mathrm{r(SUM)}$ vs $\mathrm{r(LB)}$ for fixed left bound $u = \frac{L}{6}, \frac{L}{7}, \frac{L}{8}, 0$, $s_i \in (u, v]$







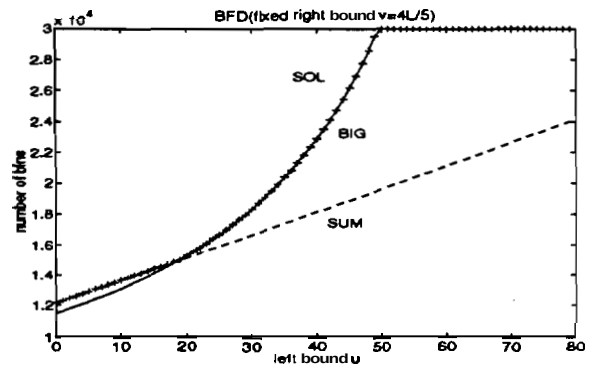
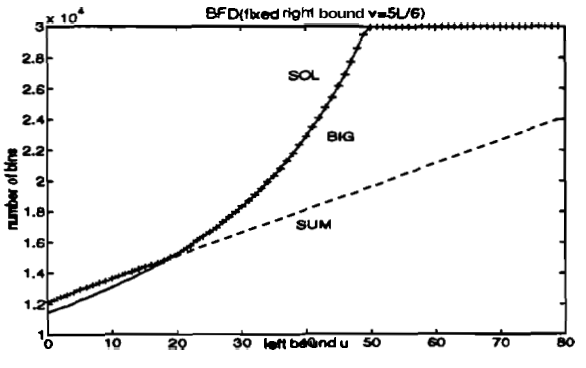
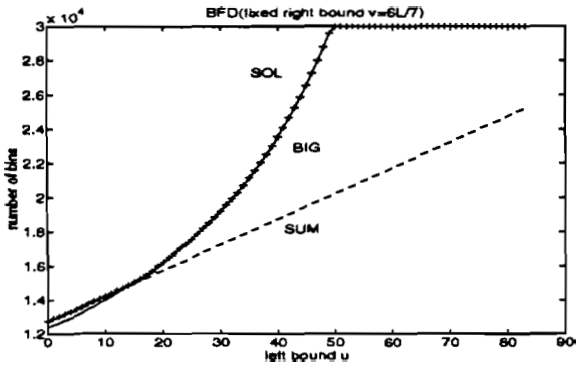
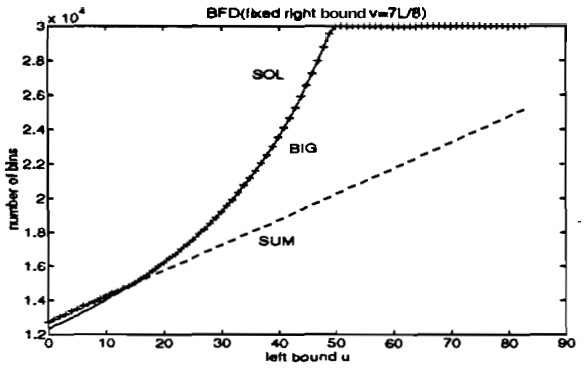


Figure 19: Solution vs lower bounds for fixed right bound $v = \frac{L}{2}, \frac{2L}{3}, \frac{3L}{4}, \frac{4L}{5}, s_i \in (u, v]$







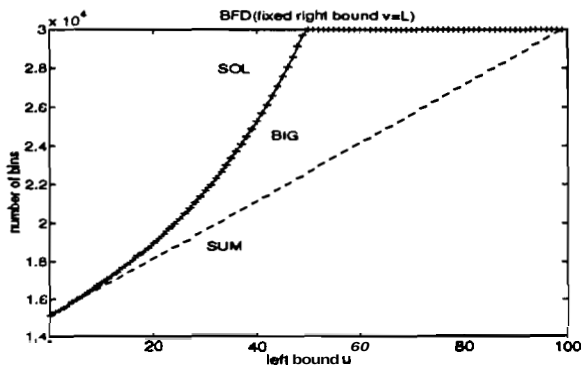
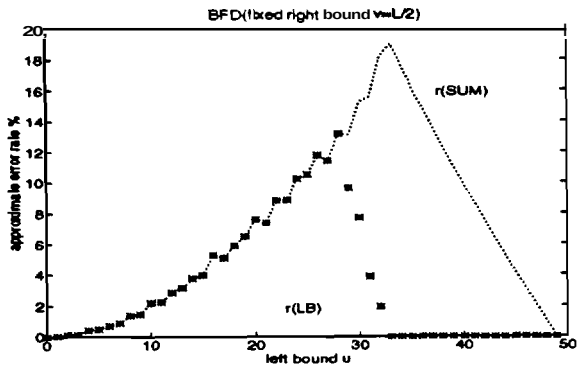
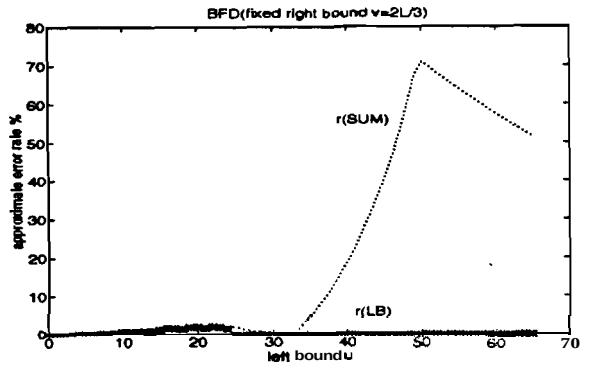
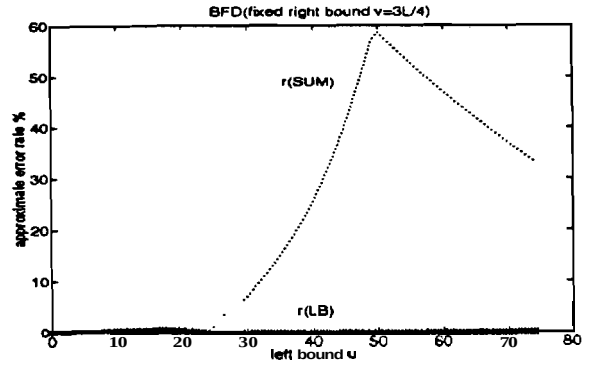


Figure 20: Solution vs lower bounds for fixed right bound $v = \frac{5L}{6}, \frac{6L}{7}, \frac{7L}{8}, L, s_i \in (u, v]$







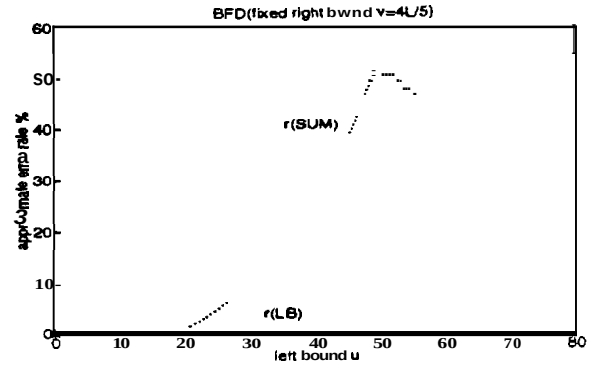
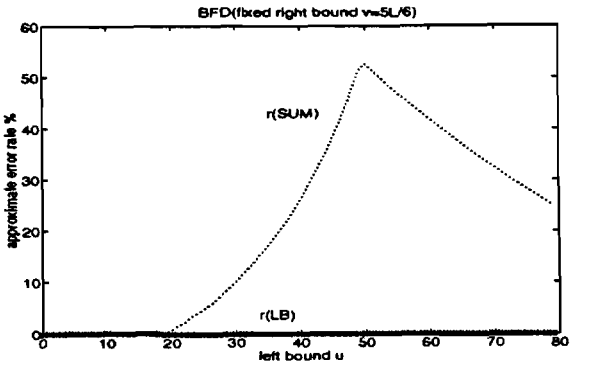
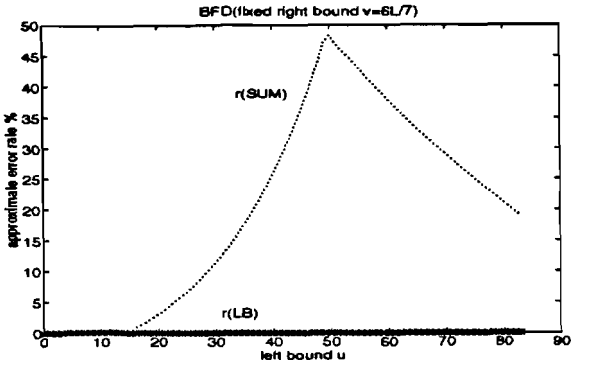
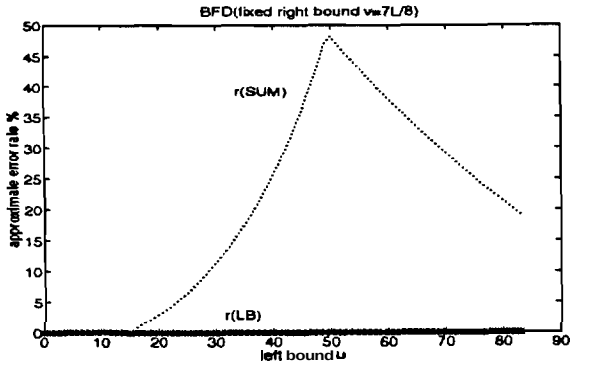


Figure 21: $\mathbf{r(SUM)}$ vs $\mathbf{r(LB)}$ for fixed right bound $v = \frac{\mathrm{L}}{2}, \frac{2L}{3}, \frac{3\mathrm{L}}{4}, \frac{4L}{5}, s_i \in (u, v]$







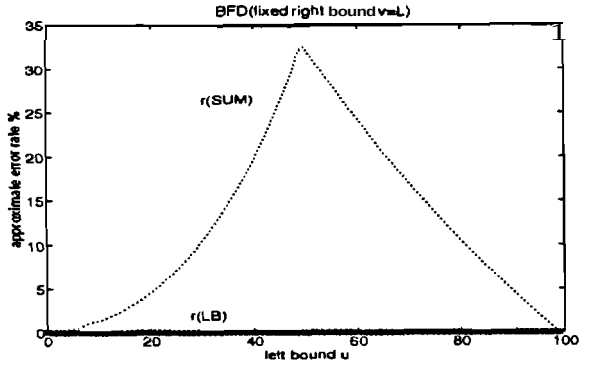


Figure 22: $\mathbf{r(SUM)}$ vs $\mathbf{r(LB)}$ for fixed right bound $v = \frac{5L}{6}, \frac{6\mathrm{L}}{7}, \frac{7L}{8}, L, s_i \in (u, v]$

References

[Baa88] Sara Baase. Computer Algorithms. Addison-Wesley Publishing Company, San Diego, CA, 1988.

[BJ83] Jon Louis Bentley and David S. Johnson. An experimental study of bin packing. In Proceedings of the *21st* Annual Allerton Conference on Communication Control and Computing, 1983.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. McGraw-Hill Book Company, New York, NY, 1990.

[GJ79] Michael R. Garey and David S. Johnson. Computers and Intractability. W.H. freeman and Company, San Francisco, CA, 1979.

[Joh73] D.S. Johnson. Near-Optimal Bin Packing Algorithms. PhD thesis, MIT, 1973.

[Joh74] David S. Johnson. Fast algorithms for bin packing. Journal of Computers and System Sciences, 8:272–314, 1974.

[LL85] C.C. Lee and D.T. Lee. A simple on-line bin packing algorithm. Journal of the Association for Computing Machiney, 32:562–572, 1985.

[Yao80] Andrew Chi-Chih Yao. New algorithms for bin packing. Journal of the Association for Computing Machiney, 27:207–227, 1980.