Discrete Applied Mathematics 28 (1990) 59–70
North-Holland

# LOWER BOUNDS AND REDUCTION PROCEDURES FOR THE BIN PACKING PROBLEM

Silvano MARTELLO and Paolo TOTH
*DEIS-University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy*

Received 27 October 1989

The bin packing problem, in which a set of items of various sizes has to be packed into a minimum number of identical bins, has been extensively studied during the past fifteen years, mainly with the aim of finding fast heuristic algorithms to provide good approximate solutions. We present lower bounds and a dominance criterion and derive a reduction algorithm. Lower bounds are evaluated through an extension of the concept of worst-case performance. For both lower bounds and reduction algorithm an experimental analysis is provided.

*Keywords.* Bin packing problem, lower bound, worst-case performance, dominance criterion, reduction algorithm.

## 1. Introduction

Given a positive integer $C$ and a set $I=\{1,2,\ldots,n\}$ of $n$ items, each having a positive integer size $s_i$ $(i=1,\ldots,n)$ satisfying $s_i \leq C$, the *bin packing problem* (*BPP*) is to find the smallest integer $m$ such that there is a partition $B=\{B_1, B_2, \ldots, B_m\}$ of set $I$ where the sum of the sizes in each $B_j$ does not exceed $C$. Informally we can think of the problem of packing the $n$ items into a minimum number of identical bins of capacity $C$; each subset $B_j$ is in this case the contents of a bin.

The bin packing problem is NP-hard in the strong sense (see Garey and Johnson [3]). In the past fifteen years an impressive amount of research on approximate algorithms for this problem has been published in the literature. We refer the interested reader to the excellent survey by Coffman, Garey and Johnson [1], which includes a bibliography of more than one hundred titles.

Heuristic algorithms for the bin packing problem are usually evaluated through their worst-case performance. Given any instance $P$ of the problem, let $m(P)$ be its optimal solution value and $h(P)$ the solution value given by a heuristic algorithm $h$. The *worst-case performance ratio* of $h$ is then defined as the minimum value $r(h)$ such that

$$r(h) \geq h(P)/m(P) \quad \text{for all } P.$$

Almost all the papers published on the bin packing problem concern heuristic algorithms. Very little has been done on lower bounds, dominance criteria and exact algorithms (to our knowledge, only Eilon and Christofides [2] and Hung and Brown [4] have presented exact procedures).

We present lower bounds and dominance criteria. These results are useful in two contexts. First, when a heuristic is used, the quality of the solution found can be evaluated through comparison with a lower bound. Second, lower bounds and dominance criteria can be used to obtain an enumerative algorithm for exact solution of the problem.

Section 2 analyzes lower bounds and evaluates them through an extension of the concept of worst-case performance. Section 3 analyzes dominances and shows how to use them to reduce the size of the problem and produce another lower bound. The average performance of lower bounds and reductions for randomly generated problems is experimentally analyzed in Section 4.

In what follows we will suppose, without loss of generality, that the items are sorted so that

$$s_1 \geq s_2 \geq \cdots \geq s_n. \tag{1}$$

## 2. Lower bounds

Let $L$ be a lower bound procedure. Given any instance $P$ of the problem, let $m(P)$ be its optimal solution value and $L(P)$ the solution value given by $L$. We define the worst-case performance ratio of $L$ as the maximum value $R(L)$ such that

$$R(L) \leq L(P)/m(P) \quad \text{for all } P.$$

For the sake of simplicity we will denote with $L$ both a lower bound procedure and the value it gives for a specific instance of the problem, with $m$ the value of the corresponding optimal solution.

### 2.1. *Bound $L_1$*

An obvious lower bound for the bin packing problem can be computed in $O(n)$ time as

$$L_1 = \left\lceil \sum_{i \in I} s_i / C \right\rceil.$$

It is easy to see that $R(L_1) = \frac{1}{2}$. In fact, in any optimal solution, at most one bin (say the $m$th) can have $\sum_{i \in B_m} s_i \leq \frac{1}{2}C$ (if two such bins existed, they could be replaced by a single bin). It follows that $\sum_{i \in I} s_i > \sum_{j=1}^{m-1} \sum_{i \in B_j} s_i > \frac{1}{2}(m-1)C$, from which $m \leq \lceil 2 \sum_{i \in I} s_i / C \rceil$. Hence $L_1/m \geq \frac{1}{2}$.

To prove that $\frac{1}{2}$ is tight consider the series of instances with $C = 2v$, $s_i = v + 1$ $(i \in I)$ for $v = 1, 2, \ldots$ and $n = 1, 2, \ldots$. For this series $m = n$ and $L_1 = \lceil n(v+1)/(2v) \rceil$, so $L_1/m$ can be arbitrarily close to $\frac{1}{2}$.

Despite its simplicity, $L_1$ can be expected to have "good" behaviour for problems where sizes $s_i$ are sufficiently small with respect to capacity $C$. $L_1$ in fact is obtained by supposing that the items can be split between different bins, so the error it introduces generally decreases when data tend to continuity. The computational results of Section 4 confirm this, since, for random problems with such characteristics, $L_1$ is generally very close to the optimal value $m$.

For different problems, i.e., problems where few items can be allocated to each bin, we propose the following better bound.

## 2.2. Bound $L_2$

**Theorem 2.1.** *Given any integer $K$, $0 \le K \le \frac{1}{2}C$, let:*

$$N_1 = \{i \in I: s_i > C - K\};$$
$$N_2 = \{i \in I: C - K \ge s_i > \tfrac{1}{2}C\};$$
$$N_3 = \{i \in I: \tfrac{1}{2}C \ge s_i \ge K\};$$

*then*

$$L(K) = |N_1| + |N_2| + \max\left(0, \left\lceil \frac{\sum_{i \in N_3} s_i - (|N_2|C - \sum_{i \in N_2} s_i)}{C} \right\rceil\right)$$

*is a valid lower bound on $m$.*

**Proof.** Since the items in $N_1 \cup N_2$ have size greater than $\frac{1}{2}C$, each of them requires a separate bin; hence $|N_1| + |N_2|$ bins are needed, in any feasible solution, for these items. No item in $N_3$ can be allocated to a bin containing an item of $N_1$, because of the capacity constraint. The total residual capacity of the $|N_2|$ bins needed for the items in $N_2$ is $\bar{C} = |N_2|C - \sum_{i \in N_2} s_i$. In the best case $\bar{C}$ will be completely filled by items in $N_3$ and remaining total size $S = \sum_{i \in N_3} s_i - \bar{C}$ will require $\lceil S/C \rceil$ new bins if $S > 0$. □

**Corollary 2.2.** $L_2 = \max_{0 \le K \le C/2} (L(K))$ *is a valid lower bound on $m$.*

**Proof.** Obvious. □

**Corollary 2.3.** $L_2 \ge L_1$.

**Proof.** Consider the case $K = 0$. We have:

$$L(0) = 0 + |N_2| + \max\left(0, \left\lceil \frac{\sum_{i \in I} s_i - |N_2|C}{C} \right\rceil\right)$$
$$= |N_2| + \max(0, L_1 - |N_2|).$$

It follows that $L_2 \ge L(0) = \max(|N_2|, L_1)$. □

Computation of $L_2$ as indicated by Corollary 2.2 is impractical. We will now show how to compute $L_2$ in $O(n)$ time.

**Theorem 2.4.** *Let $D$ be the set of distinct values $s_i \le \frac{1}{2}C$. If $D = \emptyset$, then $L_2 = |N_1| + |N_2|$; otherwise $L_2 = \max_{K \in D}(L(K))$.*

**Proof.** If $D = \emptyset$, the thesis is obvious. Otherwise we will prove that, given $K_1 < K_2$, if $K_1$ and $K_2$ produce the same set $N_3$, then $L(K_1) \le L(K_2)$. In fact: (a) the quantity $|N_1| + |N_2|$ is independent of $K$; (b) the quantity $(|N_2|C - \sum_{i \in N_2} s_i)$ produced by $K_1$ is not less than the corresponding quantity produced by $K_2$, since set $N_2$ produced by $K_2$ is a subset of set $N_2$ produced by $K_1$. Hence the theorem is true, since only distinct values $s_i \le \frac{1}{2}C$ produce different sets $N_3$ and each value $s_i$ dominates the values $s_i - 1, \dots, s_{i+1} + 1$. $\square$

**Corollary 2.5.** *If the items are sorted according to* (1), *$L_2$ can be computed in* $O(n)$ *time.*

**Proof.** Let $s_{\bar{v}}$ be the largest $s_v \le \frac{1}{2}C$. Following Theorem 2.4, we can compute $L(s_v)$ for $v = \bar{v}, \bar{v}+1, \dots, n$ by considering only distinct $s_v$ values. The computation of $L(s_{\bar{v}})$ clearly requires $O(n)$ time. Since $|N_1| + |N_2|$ is a constant, the computation of each new $L(s_v)$ simply requires the updating of $|N_2|$, $\sum_{i \in N_3} s_i$ and $\sum_{i \in N_2} s_i$. Hence all the updatings can be computed in $O(n)$ time, since they correspond to a constant time for each $v = \bar{v}+1, \bar{v}+2, \dots, n$. $\square$

The average efficiency of this computation can be improved on as follows. At any iteration, let $\bar{L}_2$ be the largest $L(s_v)$ computed so far. If $|N_1| + |N_2| + \lceil (\sum_{i=\bar{v}}^{n} s_i - (|N_2|C - \sum_{i \in N_2} s_i))/C \rceil \le \bar{L}_2$, then $L_2 = \bar{L}_2$ since no further iteration could produce a better bound.

We illustrate the above results with a numerical example. Let:

$$n = 9, \quad C = 100, \quad (s) = (70, 60, 50, 33, 33, 33, 11, 7, 3);$$

an optimal solution is $B_1 = \{1, 7, 8, 9\}$, $B_2 = \{2, 4\}$, $B_3 = \{3, 5\}$, $B_4 = \{6\}$. Bound $L_1$ gives $\lceil 300/100 \rceil = 3$. In order to determine $L_2$ we compute:

$$L(50) = 2 + 0 + \max(0, \lceil (50 - 0)/100 \rceil) = 3;$$

$$L(33) = 1 + 1 + \max(0, \lceil (149 - 40)/100 \rceil) = 4;$$

since at this point we have that $1 + 1 + \lceil (170 - 40)/100 \rceil = 4$, no further $L(s_v)$ needs to be computed and $L_2 = 4 = m$.

## 2.3. *Worst-case performance of bound $L_2$*

**Theorem 2.6.** $R(L_2) = \frac{2}{3}$.

**Proof.** Given any instance $P$ of BPP, consider sets $N_1$, $N_2$, $N_3$ as defined in Theorem 2.1. If $N_3 = \emptyset$, then $L_2 = |N_1| + |N_2| = m$. If $N_3 \ne \emptyset$, we will prove that

$L_2 \geq L(0) \geq \frac{2}{3}m$. Let $\bar{P}$ be the relaxed instance obtained by supposing that each item in $N_3$ can be split between different bins. Clearly $L(0)$ is the value of the optimal solution to $\bar{P}$. This solution can be obtained as follows. $|N_2|$ bins are first initialized for the items in $N_2$ (note that $N_1 = \emptyset$ since $K=0$). Then, for each item $k \in N_3$, let $B_{\bar{m}}$ be the first bin not completely loaded (if no such bin exists, initialize a new bin). If $s_k \leq C - \sum_{i \in B_{\bar{m}}} s_i$, then item $k$ is allocated to bin $B_{\bar{m}}$; otherwise item $k$ is replaced by two items $k_1$, $k_2$ with $s_{k_1} = C - \sum_{i \in B_{\bar{m}}} s_i$, $s_{k_2} = s_k - s_{k_1}$, item $k_1$ is allocated to bin $B_{\bar{m}}$ and execution continues with item $k_2$. In this solution at most $L(0)-1$ items have been split (no splitting can occur in the $L(0)$th bin). We can now obtain a feasible solution of value $\tilde{m} \geq m$ to instance $P$ by removing the split items from the previous solution and by allocating them to new bins. Because of the definition of $N_3$, at most $\lceil \frac{1}{2}(L(0)-1) \rceil$ new bins are needed, so $\tilde{m} \leq L(0) + \lfloor \frac{1}{2}L(0) \rfloor$. It follows that $\frac{3}{2}L(0) \geq m$.

We have proved that $R(L_2) \geq \frac{2}{3}$. To prove that $\frac{2}{3}$ is tight consider the series of instances with $n$ even, $C = 3v$, $s_i = v+1$ $(i \in I)$ for $v = 1, 2, \ldots$ and $n = 1, 2, \ldots$. For this series $m = \frac{1}{2}n$ and $L_2 = L(v+1) = \lceil (n(v+1))/(3v) \rceil$, so $L_2/m$ can be arbitrarily close to $\frac{2}{3}$. □

Worth is noting that a better worst-case performance can be obtained from approximate algorithms. Using for example Johnson's [5] best-fit decreasing algorithm to produce a solution of value $b$—for which it is known that $b \leq \min(\frac{3}{2}m, \frac{11}{9}m + 4)$—we obtain a lower bound

$$\text{LBF} = \max(\lceil \tfrac{2}{3}b \rceil, \lceil \tfrac{9}{11}(b-4) \rceil)$$

whose asymptotic worst-case performance ratio is $\frac{9}{11}$. Since however $b$ is known to be, in general, close to the optimum, the average performance of LBF can be expected to be quite poor (see Section 4).

## 3. Dominances and reductions

In this section we present a dominance criterion and show how it can be used to reduce the size of a bin packing problem and determine a lower bound.

### 3.1. Dominance criterion

**Definition 3.1.** A set $F$ is a *feasible set* iff $F \subseteq I$ and $\sum_{i \in F} s_i \leq C$.

**Definition 3.2.** Given two feasible sets $F_1$ and $F_2$, $F_1$ *dominates* $F_2$ iff the value of the optimal solution we can obtain by setting $B_1 = F_1$ is not greater than we can obtain by setting $B_1 = F_2$.

The following criterion gives a sufficient condition for a feasible set $F_1$ to dominate a feasible set $F_2$.

**Dominance criterion.** Given two distinct feasible sets $F_1$ and $F_2$, if there exist a partition $P=\{P_1,\dots,P_l\}$ of $F_2$ and a subset $\{i_1,\dots,i_l\}$ of $F_1$ such that $s_{i_h}\geq \sum_{k\in P_h} s_k$ for $h=1,\dots,l$, then $F_1$ dominates $F_2$.

**Proof.** Given any feasible solution with $B_1=F_2$, we can obtain a feasible solution of no greater value by interchanging each $P_h$ with the corresponding item $i_h$ and moving to $B_1$ the items of $F_1\setminus\{i_1,\dots,i_l\}$ (note that this transformation is in general impossible for the opposite case). For this solution $B_1=F_1$ and the criterion follows from Definition 3.2. □

## *3.2. Reduction procedures*

A general procedure to reduce the size of an instance of BPP could consider all feasible sets, find one (say $F$) dominating all the others, assign $F$ to a new bin and remove $F$ from $I$; this could be iterated until either no such $F$ exists or $I=\emptyset$. A possible implementation could be obtained through the following:

**Consideration 3.3.** Given any item $i$, if a feasible set $F$ containing $i$ dominates all feasible sets containing $i$, then there exists an optimal solution where $B_j=F$ for some $j$.

**Proof.** Immediate from Definition 3.2. □

The resulting procedure would be:

```
$j:=0$, $\bar{I}:=\emptyset$;
repeat
  let $i$ be the first item in $I\setminus\bar{I}$;
  if a feasible set $F$ containing $i$ dominates all feasible sets containing $i$
    then $j:=j+1$, $B_j:=F$, $I:=I\setminus F$
  else $\bar{I}:=\bar{I}\cup\{i\}$
until $I\setminus\bar{I}=\emptyset$.
```

After execution the first $j$ bins are fixed and we need to solve the BPP relative to the items currently in $I$. This procedure, however, could not be implemented efficiently. A simplified procedure can be obtained by limiting the cardinality of the feasible sets considered and by checking dominances only through our criterion.

```
    Procedure REDUCTION($I$);
1     $j:=0$, $\bar{I}:=\emptyset$;
2     repeat
3       find $i=\min\{h: h\in I\setminus\bar{I}\}$, let $I'=I\setminus\{i\}=\{i_1,i_2,\dots,i_l\}$ with $s_{i_1}\geq$
          $s_{i_2}\geq\cdots\geq s_{i_l}$ and set $F:=\emptyset$;
```

```
4            find the largest $k$ such that $s_i + \sum_{q=l-k+1}^{l} s_{i_q} \le C$;
5            if $k=0$ then $F := \{i\}$
6            else find $s_{\bar{i}} = \max\{s_h : h \in I',\ s_i + s_h \le C\}$;
7              if $k=1$ or $s_i + s_{\bar{i}} = C$ then $F := \{i, \bar{i}\}$
8              else
                 if $k=2$
9                then find $s_{\bar{i}_1} + s_{\bar{i}_2} = \max\{s_{h_1} + s_{h_2} : h_1, h_2 \in I', s_i + s_{h_1} + s_{h_2} \le C\}$;
10                 if $s_{\bar{i}} \ge s_{\bar{i}_1} + s_{\bar{i}_2}$ then $F := \{i, \bar{i}\}$
11                 else
                     if $s_{\bar{i}} = s_{\bar{i}_1}$ and $\{i, \bar{i}_1, \bar{i}_2\}$ dominates all feasible triplets of $I$
                       containing $i$
                     then $F := \{i, \bar{i}_1, \bar{i}_2\}$;
12           if $F \ne \emptyset$ then $j := j+1$, $B_j := F$, $I := I \setminus F$
13           else $\bar{I} := \bar{I} \cup \{i\}$
14         until $I \setminus \bar{I} = \emptyset$.
```

In order to prove the correctness of Procedure REDUCTION we have to prove that, whenever $F$ is defined, there exists an optimal solution where $B_j = F$ for some $j$. At step 5, if $k=0$, then the bin containing item $i$ can contain no further item, so we can immediately define $F = \{i\}$. At step 7: (a) if $k=1$, then the bin containing item $i$ can contain only one further item, so set $F = \{i, \bar{i}\}$, where $\bar{i}$ is the item of largest size which can be packed together with $i$, dominates all feasible sets containing $i$; (b) if $s_i + s_{\bar{i}} = C$, then $F = \{i, \bar{i}\}$ dominates all feasible sets containing $i$ plus one or more items. At steps 10 and 11, since $k=2$ (see step 8), we know that the bin containing item $i$ can contain at most two further items. At step 10, if $s_{\bar{i}} \ge s_{\bar{i}_1} + s_{\bar{i}_2}$, where $(\bar{i}_1, \bar{i}_2)$ is the pair of items of largest total size which can be packed together with $i$, then $F = \{i, \bar{i}\}$ dominates all feasible sets containing $i$. At step 11 we set $F = \{i, \bar{i}_1, \bar{i}_2\}$ if such $F$ dominates all feasible triplets and all feasible pairs (since $s_{\bar{i}} = s_{\bar{i}_1}$) containing item $i$.

The procedure requires $O(n^2)$ time. In fact the repeat-until loop is executed $O(n)$ times and the heaviest steps (9 and 11) can be implemented by exploiting the sorting of the items so as to require $O(n)$ time at each iteration. A simplified version of the procedure can be obtained by suppressing steps 8–11. This version requires $O(n)$ time since steps 4 and 6 can be easily implemented so as to globally require $O(n)$ time. The procedure can also be generalized by considering the feasible quadruplets containing item $i$ (if $k=3$), the feasible quintuplets containing item $i$ (if $k=4$), and so on.

### *3.3. Lower bound LR*

The reduction procedure of the previous section can be used to compute a new lower bound. In order to do this, we need to define some possible relaxations of an instance of BPP:

*Relaxation* 1: Substitute one item (say $i \in I$) with two new items $i_1$, $i_2$ such that $s_{i_1} + s_{i_2} \le s_i$.
*Relaxation* 2: Substitute one item (say $i \in I$) with a new item $i_1$ of size $s_{i_1} < s_i$.
*Relaxation* 3: Eliminate one item from $I$.

A lower bound LR can then be computed as follows:

```
    Procedure LR
1     LR:= 0, $\bar{j}$:= 0;
2     while $I \neq \emptyset$ do
3       apply REDUCTION($I$), let $j$ be the number of new bins fixed and
          set $\bar{j} := \bar{j} + j$;
4       compute $L_2$ for the items currently in $I$;
5       if $\bar{j} + L_2 >$ LR then LR:= $\bar{j} + L_2$;
6       relax the problem corresponding to $I$.
```

The complexity of this procedure depends on the relaxation used at step 6. We implemented it with Relaxation 3 by eliminating at each iteration the item of smallest size currently in $I$. In this way the procedure clearly requires $O(n^3)$ time.

Procedure LR can also be used to produce a better reduction through the following:

**Consideration 3.4.** At any iteration of Procedure LR, if all the items relaxed in the previous iterations can be inserted, through any heuristic, into bins $B_1, \ldots, B_{\bar{j}}$, then bins $B_1, \ldots, B_{\bar{j}}$ can be fixed for the original problem.

**Proof.** Suppose that relaxed item $i$ can be inserted in bin $B_k = \{i_1, \ldots, i_p\}$ $(1 \le k \le \bar{j})$. Since, with respect to the relaxed problem, the feasible set $B_k$ dominates all feasible sets containing item $i_1$, the same holds for the new feasible set $B_k \cup \{i\}$ with respect to the problem where item $i$ is not relaxed. □

## 3.4. Example

Let $n = 14$, $C = 100$, $(s) = (99, 94, 79, 64, 50, 46, 43, 37, 32, 19, 18, 7, 6, 3)$ and apply Procedure LR.

The first execution of REDUCTION gives: $B_1 = \{1\}$ (from step 5), $B_2 = \{2, 13\}$ (from step 7). Hence $\bar{j} = 2$, $L_2 = 4$, LR $= 6$. We now eliminate item 14. The current vector $s$ is

$$(s) = (-, -, 79, 64, 50, 46, 43, 37, 32, 19, 18, 7, -, -).$$

The second execution of REDUCTION gives: $B_3 = \{3, 10\}$ (from step 7), $B_4 = \{4, 9\}$ (from step 10), $B_5 = \{6, 5\}$ (from step 10), $B_6 = \{7, 8, 11\}$ (from step 11), $B_7 = \{12\}$ (from step 5). Hence $\bar{j} = 7$, $L_2 = 0$ (since $I = \emptyset$), LR $= 7$ and the execution terminates.

The eliminated item 14 can now be inserted into bin $B_4$ which becomes $B_4 = \{4, 9, 14\}$ and, because of Consideration 3.4, bins $B_1$ to $B_7$ can be fixed for the original problem. In particular this solution is optimal since all the items have been packed into LR bins.

## 4. Computational experiments

We have coded in FORTRAN IV the procedures to compute $L_1$, $L_2$ and LBF, and Procedures REDUCTION and LR. The programs have been run on a VAX 11/780 on three classes of randomly generated item sizes:

(1) $s_i$ uniformly random between 1 and 100;
(2) $s_i$ uniformly random between 20 and 100;
(3) $s_i$ uniformly random between 50 and 100.

For each class, three values of $C$ have been considered: $C=100$, $C=120$, $C=150$. For each pair (class, value of $C$) and for different values of $n$ ($n=50, 100, 200, 500, 1000$), 20 problems have been generated. The entries in Tables 1, 2 and 3 give, for each bound, the average computing time, the average percentage error and the number of times the bound obtained was optimal (the values of the optimal solution have been obtained through an enumerative algorithm). LBF requires times almost independent from the data generation and, obviously, produces high errors, tending to $\frac{9}{11}$ when $n$ grows. $L_1$ requires very small times, practically independent from the data generation; the approximation it produces improves with the ratio $C/\min_i\{s_i\}$ (as could be expected, since the bound is based on the continuous relaxation of the

Table 1. VAX 11/780 seconds. Values computed over 20 instances. $C=100$.

| | | LBF | | $L_1$ | | $L_2$ | | LR | |
|---|---|---|---|---|---|---|---|---|---|
| Class | $n$ | Time | % err. (opt.) | Time | % err. (opt.) | Time | % err. (opt.) | Time | % err. (opt.) |
| | 50 | 0.005 | 28.046 (0) | 0.001 | 6.319 (3) | 0.008 | 0.530 (17) | 0.007 | 0.185 (19) |
| | 100 | 0.013 | 23.434 (0) | 0.001 | 2.893 (3) | 0.007 | 0.675 (13) | 0.012 | 0.404 (16) |
| 1 | 200 | 0.029 | 20.873 (0) | 0.001 | 2.647 (1) | 0.008 | 0.294 (15) | 0.024 | 0.096 (18) |
| | 500 | 0.087 | 19.205 (0) | 0.003 | 2.408 (0) | 0.012 | 0.310 ( 9) | 0.055 | 0.078 (16) |
| | 1000 | 0.197 | 18.734 (0) | 0.007 | 1.785 (0) | 0.015 | 0.146 (10) | 0.149 | 0.030 (17) |
| | 50 | 0.006 | 26.410 (0) | 0.002 | 9.135 (0) | 0.006 | 0.294 (18) | 0.004 | 0.143 (19) |
| | 100 | 0.014 | 22.585 (0) | 0.003 | 6.655 (0) | 0.007 | 0.476 (14) | 0.008 | 0.000 (20) |
| 2 | 200 | 0.031 | 20.336 (0) | 0.002 | 6.069 (0) | 0.010 | 0.316 (12) | 0.014 | 0.000 (20) |
| | 500 | 0.092 | 19.012 (0) | 0.003 | 5.587 (0) | 0.009 | 0.206 (10) | 0.039 | 0.016 (19) |
| | 1000 | 0.202 | 18.613 (0) | 0.005 | 5.099 (0) | 0.012 | 0.103 (13) | 0.076 | 0.024 (17) |
| | 50 | 0.006 | 24.044 (0) | 0.001 | 23.119 (0) | 0.007 | 0.000 (20) | 0.006 | 0.000 (20) |
| | 100 | 0.015 | 21.160 (0) | 0.002 | 24.228 (0) | 0.005 | 0.000 (20) | 0.008 | 0.000 (20) |
| 3 | 200 | 0.034 | 19.608 (0) | 0.006 | 24.440 (0) | 0.004 | 0.000 (20) | 0.010 | 0.000 (20) |
| | 500 | 0.095 | 18.746 (0) | 0.004 | 23.876 (0) | 0.004 | 0.000 (20) | 0.031 | 0.000 (20) |
| | 1000 | 0.218 | 18.469 (0) | 0.004 | 24.110 (0) | 0.006 | 0.000 (20) | 0.063 | 0.000 (20) |

Table 2. VAX 11/780 seconds. Values computed over 20 instances. $C = 120$.

| | | LBF | | $L_1$ | | $L_2$ | | LR | |
|---|---|---|---|---|---|---|---|---|---|
| Class | $n$ | Time | % err. (opt.) | Time | % err. (opt.) | Time | % err. (opt.) | Time | % err. (opt.) |
| | 50 | 0.006 | 30.073 (0) | 0.002 | 5.422 ( 7) | 0.006 | 0.445 (18) | 0.010 | 0.217 (19) |
| | 100 | 0.012 | 24.220 (0) | 0.002 | 1.481 (13) | 0.009 | 0.442 (13) | 0.023 | 0.114 (19) |
| 1 | 200 | 0.029 | 21.285 (0) | 0.002 | 0.972 (11) | 0.011 | 0.291 (15) | 0.072 | 0.119 (18) |
| | 500 | 0.082 | 19.445 (0) | 0.002 | 0.804 (11) | 0.012 | 0.045 (18) | 0.352 | 0.023 (19) |
| | 1000 | 0.183 | 18.806 (0) | 0.007 | 0.182 (14) | 0.017 | 0.057 (16) | 1.310 | 0.011 (19) |
| | 50 | 0.005 | 28.419 (0) | 0.002 | 8.040 ( 1) | 0.007 | 0.554 (17) | 0.006 | 0.000 (20) |
| | 100 | 0.013 | 23.476 (0) | 0.001 | 3.836 ( 1) | 0.007 | 0.678 (13) | 0.010 | 0.000 (20) |
| 2 | 200 | 0.028 | 20.979 (0) | 0.003 | 2.848 ( 1) | 0.013 | 0.540 (10) | 0.018 | 0.048 (19) |
| | 500 | 0.088 | 19.280 (0) | 0.006 | 3.076 ( 0) | 0.010 | 0.353 (10) | 0.043 | 0.040 (18) |
| | 1000 | 0.187 | 18.730 (0) | 0.007 | 2.253 ( 0) | 0.017 | 0.198 (10) | 0.110 | 0.090 (15) |
| | 50 | 0.006 | 24.966 (0) | 0.003 | 23.856 ( 0) | 0.006 | 0.000 (20) | 0.006 | 0.000 (20) |
| | 100 | 0.014 | 21.634 (0) | 0.001 | 23.064 ( 0) | 0.008 | 0.000 (20) | 0.008 | 0.000 (20) |
| 3 | 200 | 0.033 | 19.916 (0) | 0.002 | 22.790 ( 0) | 0.007 | 0.030 (19) | 0.014 | 0.000 (20) |
| | 500 | 0.095 | 18.900 (0) | 0.003 | 22.159 ( 0) | 0.010 | 0.075 (15) | 0.032 | 0.000 (20) |
| | 1000 | 0.216 | 18.534 (0) | 0.004 | 21.913 ( 0) | 0.012 | 0.100 ( 9) | 0.060 | 0.000 (20) |

problem). $L_2$ requires slightly higher times, but produces better approximation; for class (1) it improves when $C$ grows, for classes (2) and (3) it gets worse when $C$ grows. The times required by LR are in general comparatively high, and clearly growing both with $n$ and $C$; the approximation is generally very good, with few exceptions.

Table 3. VAX 11/780 seconds. Values computed over 20 instances. $C = 150$.

| | | LBF | | $L_1$ | | $L_2$ | | LR | |
|---|---|---|---|---|---|---|---|---|---|
| Class | $n$ | Time | % err. (opt.) | Time | % err. (opt.) | Time | % err. (opt.) | Time | % err. (opt.) |
| | 50 | 0.004 | 31.490 (0) | 0.003 | 0.455 (19) | 0.007 | 0.000 (20) | 0.019 | 0.000 (20) |
| | 100 | 0.011 | 25.923 (0) | 0.002 | 0.000 (20) | 0.006 | 0.000 (20) | 0.059 | 0.000 (20) |
| 1 | 200 | 0.025 | 22.339 (0) | 0.001 | 0.074 (19) | 0.013 | 0.074 (19) | 0.220 | 0.074 (19) |
| | 500 | 0.084 | 19.802 (0) | 0.004 | 0.029 (19) | 0.015 | 0.029 (19) | 1.327 | 0.029 (19) |
| | 1000 | 0.176 | 19.001 (0) | 0.004 | 0.015 (19) | 0.019 | 0.015 (19) | 5.890 | 0.015 (19) |
| | 50 | 0.005 | 31.580 (0) | 0.004 | 1.053 (17) | 0.008 | 0.476 (18) | 0.018 | 0.476 (18) |
| | 100 | 0.013 | 23.853 (0) | 0.002 | 0.488 (16) | 0.009 | 0.488 (16) | 0.066 | 0.488 (16) |
| 2 | 200 | 0.028 | 21.135 (0) | 0.003 | 0.984 ( 5) | 0.009 | 0.984 ( 5) | 0.248 | 0.984 ( 5) |
| | 500 | 0.088 | 19.383 (0) | 0.005 | 1.009 ( 0) | 0.016 | 1.009 ( 0) | 1.660 | 1.009 ( 0) |
| | 1000 | 0.192 | 18.760 (0) | 0.006 | 1.067 ( 0) | 0.016 | 1.067 ( 0) | 7.147 | 1.067 ( 0) |
| | 50 | 0.005 | 28.272 (0) | 0.002 | 8.824 ( 2) | 0.006 | 0.538 (17) | 0.005 | 0.000 (20) |
| | 100 | 0.012 | 23.323 (0) | 0.001 | 4.369 ( 2) | 0.008 | 0.673 (14) | 0.008 | 0.000 (20) |
| 3 | 200 | 0.029 | 20.937 (0) | 0.001 | 3.143 ( 0) | 0.011 | 0.541 (10) | 0.013 | 0.000 (20) |
| | 500 | 0.089 | 19.271 (0) | 0.004 | 3.398 ( 0) | 0.012 | 0.294 (11) | 0.031 | 0.000 (20) |
| | 1000 | 0.197 | 18.761 (0) | 0.005 | 2.714 ( 0) | 0.015 | 0.277 (11) | 0.063 | 0.000 (20) |

Table 4. Results over 20 instances.

| | | $C=100$ | | | | | $C=150$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | $n$ | $L_1=b$ | $L_2=b$ | $\text{LR}=b$ | Compl. red. | Aver. left | $L_1=b$ | $L_2=b$ | $\text{LR}=b$ | Compl. red. | Aver. left |
| | 50 | 0 | 14 | 16 | 17 | 2.7 | 17 | 18 | 18 | 4 | 34.5 |
| | 100 | 2 | 11 | 14 | 14 | 9.9 | 15 | 15 | 15 | 0 | 76.1 |
| 1 | 200 | 0 | 12 | 15 | 15 | 5.7 | 19 | 19 | 19 | 0 | 135.8 |
| | 500 | 0 | 9 | 14 | 14 | 10.1 | 16 | 16 | 16 | 0 | 306.7 |
| | 1000 | 0 | 10 | 17 | 15 | 16.3 | 19 | 19 | 19 | 0 | 577.0 |
| | 50 | 0 | 18 | 19 | 19 | 0.7 | 14 | 15 | 15 | 3 | 31.2 |
| | 100 | 0 | 14 | 20 | 20 | 0.0 | 6 | 6 | 6 | 0 | 66.5 |
| 2 | 200 | 0 | 11 | 19 | 17 | 3.5 | 1 | 1 | 1 | 0 | 118.5 |
| | 500 | 0 | 10 | 16 | 15 | 7.3 | 0 | 0 | 0 | 0 | 256.8 |
| | 1000 | 0 | 13 | 16 | 15 | 9.5 | 0 | 0 | 0 | 0 | 462.6 |
| | 50 | 0 | 20 | 20 | 20 | 0.0 | 2 | 17 | 20 | 20 | 0.0 |
| | 100 | 0 | 20 | 20 | 20 | 0.0 | 2 | 14 | 20 | 20 | 0.0 |
| 3 | 200 | 0 | 20 | 20 | 20 | 0.0 | 0 | 10 | 20 | 20 | 0.0 |
| | 500 | 0 | 20 | 20 | 20 | 0.0 | 0 | 11 | 20 | 20 | 0.0 |
| | 1000 | 0 | 20 | 20 | 20 | 0.0 | 0 | 11 | 20 | 20 | 0.0 |

Table 4 gives, for $L_1$, $L_2$ and LR, the number of times the lower bound produced was equal to the value $b$ of the approximate solution given by Johnson's [5] best-fit decreasing algorithm (hence proving optimality of $b$). In addition it gives, for Procedure REDUCTION, the number of problems completely reduced (i.e., optimally solved) and the average number of items not fixed. We do not present the results for $C=120$, as they were very close to those for $C=100$. The table shows a poor behaviour of $L_1$ (except for class (1) with $C=150$) and a satisfactory behaviour of $L_2$ and LR (except for class (2) with $C=150$). As for Procedure REDUCTION, it works very well for $C=100$ and for class (3); for the remaining cases, it reduces the size of the instances to about one half.

It should be noted that the problems generated are comparatively "hard" (few items are packed in each bin). For the same problems with $C=1000$, $L_1$ requires the same times and almost always gives the same value as the best-fit decreasing algorithm.

## Acknowledgement

This research was supported by Ministero della Pubblica Istruzione, Italy. We thank Dr. Matteo Fischetti and an anonymous referee for helpful comments.

## References

[1] E.G. Coffman Jr, M.R. Garey and D.S. Johnson, Approximation algorithms for bin-packing—An updated survey, in: G. Ausiello, M. Lucertini and P. Serafini, eds., Algorithm Design for Computer System Design (Springer, Vienna, 1984).

[2] S. Eilon and N. Christofides, The loading problem, Management Sci. 17 (1971) 259-267.

[3] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, San Francisco, CA, 1979).

[4] M.S. Hung and J.R. Brown, An algorithm for a class of loading problems, Naval Res. Logist. Quart. 25 (1978) 289-297.

[5] D.S. Johnson, Near-optimal bin packing algorithms, Doctoral Thesis, Department of Mathematics, Massachussets Institute of Technology, Cambridge, MA (1973).