8

Bin-packing problem

8.1 INTRODUCTION

The Bin-Packing Problem (BPP) can be described, using the terminology of knapsack problems, as follows. Given $n$ items and $m$ knapsacks (or bins), with

- $w_j = \text{weight of item } j$,
- $c = \text{capacity of each bin},$

assign each item to one bin so that the total weight of the items in each bin does not exceed $c$ and the number of bins used is a minimum. A possible mathematical formulation of the problem is

\[
\text{minimize} \quad z = \sum_{i=1}^{n} y_i \\
\text{subject to} \quad \sum_{j=1}^{m} w_j x_{ij} \leq c_i, \quad i \in N = \{1, \ldots, n\}, \\
\sum_{i=1}^{n} x_{ij} = 1, \quad j \in N, \\
y_i = 0 \text{ or } 1, \quad i \in N, \\
x_{ij} = 0 \text{ or } 1, \quad i \in N, j \in N,
\]

where

- $y_i = 1$ if bin $i$ is used;
- $y_i = 0$ otherwise,
- $x_{ij} = 1$ if item $j$ is assigned to bin $i$;
- $x_{ij} = 0$ otherwise.

We will suppose, as is usual, that the weights $w_j$ are positive integers. Hence, without loss of generality, we will also assume that...
A brief outline of approximate algorithms

The simplest approximate approach to the bin packing problem is the Next-Fit (NF) algorithm. The first item is assigned to bin 1. Items $2,\ldots,n$ are then considered by increasing indices: each item is assigned to the current bin, if it fits; otherwise, it is assigned to a new bin, which becomes the current one. The time complexity of the algorithm is clearly $O(n)$. It is easy to prove that, for any instance $I$ of BPP, the solution value $NF(I)$ provided by the algorithm satisfies the bound

$$NF(I) \leq 2 \cdot z(I),$$

where $z(I)$ denotes the optimal solution value. Furthermore, there exist instances for which the ratio $NF(I)/z(I)$ is arbitrarily close to 2, i.e. the worst-case performance ratio of NF is $r(NF) = 2$. Note that, for a minimization problem, the worst-case performance ratio of an approximate algorithm $A$ is defined as the smallest real number $r(A)$ such that

$$\frac{A(I)}{z(I)} \leq r(A)$$

for all instances $I$,

where $A(I)$ denotes the solution value provided by $A$.

A better algorithm, First-Fit (FF), considers the items according to increasing indices and assigns each item to the lowest indexed initialized bin into which it fits; only when the current item cannot fit into any initialized bin, is a new bin introduced. It has been (1974) that

for all instances $I$ of BPP large, for which

Because of the constant algorithms, the worst-case the worst-case behaviour worst-case performance of this is defined as the number integer $k$,

$$A(I) \leq r$$

it is then clear, from (8.3)

The next algorithm, First-Fit Decreasing (FFD), is called $NF(k)$, and then FF or FF, and FFD, done by Baker and Coffman, Ullman, Garey and Graham, who proved that
8 Bin-packing problem

(8.6)

(8.7)

[...]

If an item violates the 2:1 rule, there is no easy way to assign weights to any feasible solution, the weight of the item is 1. The problem remains with approximate solutions, such results would require additional information. In Section 8.2 we consider the 2:1 rule and Johnson (1984), to prove that the problem is \\
more than one hundred years old. In Section 8.2 we present an algorithm. The remainder of the section procedures can be found in Section 8.6.

ALGORITHMS

A problem to solve is the Next-Fit (NF) algorithm. The time complexity of the problem is not considered by any algorithm. The 2:1 rule is such that for any instance $I$ of BPP, $z(I) + 1$ satisfies the bound $z(I) + 1$.

(8.8)

The next algorithm, Best-Fit (BF), is obtained from FF by assigning the current item to the feasible bin (if any) having the smallest residual capacity (to break ties in favour of the lowest indexed bin). Johnson, Demers, Ullman, Garey, and Graham (1974) have proved that BF satisfies the same worst-case bounds as FF (see (8.9)-(8.10)), hence $r^w(BF) = \frac{17}{10}$.

The time complexity of both FF and BF is $O(n \log n)$. This can be achieved by using a 2-3 tree whose leaves store the current residual capacities of the initialized bins. A 2-3 tree is a tree in which: (a) every non-leaf node has 2 or 3 sons; (b) every path from the root to a leaf has the same length; (c) labels at the nodes allow searching for a given leaf value, updating it, or inserting a new leaf in $O(1)$ time. We refer the reader to Aho, Hopcroft, and Ullman (1983) for details on this data structure. In this way each iteration of FF or BF requires $O(n \log n)$ time, since the number of leaves is bounded by $n$.

Assume now that the items are sorted so that

\[ u_1 \leq u_2 \leq \ldots \leq u_n, \]

and then FF or BF, or BF is applied. The resulting algorithms, of time complexity $O(n \log n)$, are called Next-Fit Decreasing (NFD), First-Fit Decreasing (FFD), and Best-Fit Decreasing (BFD), respectively. The worst-case analysis of NFD has been done by Baker and Coffman (1981); that of FFD and BFD by Johnson, Demers, Ullman, Garey, and Graham (1974), starting from an earlier result of Johnson (1973) who proved that

\[ FFD(I) \leq \frac{11}{10} z(I) + 4. \]
Table 8.1  Asymptotic worst-case performance ratios of bin-packing algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time complexity</th>
<th>$r^\infty$</th>
<th>$r^{i/2}$</th>
<th>$r^{i/3}$</th>
<th>$r^{i/4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF</td>
<td>$O(n)$</td>
<td>2.000</td>
<td>2.000</td>
<td>1.500</td>
<td>1.333</td>
</tr>
<tr>
<td>FF</td>
<td>$O(n\log n)$</td>
<td>1.700</td>
<td>1.500</td>
<td>1.333</td>
<td>1.250</td>
</tr>
<tr>
<td>BF</td>
<td>$O(n\log n)$</td>
<td>1.700</td>
<td>1.500</td>
<td>1.333</td>
<td>1.250</td>
</tr>
<tr>
<td>NFD</td>
<td>$O(n\log n)$</td>
<td>1.691</td>
<td>1.424</td>
<td>1.302</td>
<td>1.234</td>
</tr>
<tr>
<td>FFD</td>
<td>$O(n\log n)$</td>
<td>1.222</td>
<td>1.183</td>
<td>1.183</td>
<td>1.150</td>
</tr>
<tr>
<td>BFD</td>
<td>$O(n\log n)$</td>
<td>1.222</td>
<td>1.183</td>
<td>1.183</td>
<td>1.150</td>
</tr>
</tbody>
</table>

for all instances $I$. The results are summarized in Table 8.1 (taken from Coffman, Garey and Johnson (1984)), in which the last three columns give, for $\alpha = \frac{1}{2}, \frac{1}{3}, \frac{1}{4}$, the value $r^\infty$ of the asymptotic worst-case performance ratio of the algorithms when applied to instances satisfying $\min_{i \in N} (w_i) \leq \alpha \cdot c$.

8.3 LOWER BOUNDS

Given a lower bounding procedure $L$, for a minimization problem, let $L(I)$ and $z(I)$ denote, respectively, the value produced by $L$ and the optimal solution value for instance $I$. The worst-case performance ratio of $L$ is then defined as the largest real number $\rho(L)$ such that

$$\frac{L(I)}{z(I)} \geq \rho(L) \quad \text{for all instances } I.$$

8.3.1 Relaxations based lower bounds

For our model of BPP, the continuous relaxation $C(BPP)$ of the problem, given by (8.1)-(8.3) and

$$0 \leq y_i \leq 1, \quad i \in N,$$

$$0 \leq x_{ij} \leq 1, \quad i \in N, j \in N,$$

can be immediately solved by the values $x_{ij} = 1$, $x_{ij} = 0$ $(i \neq i)$ and $y_i = w_i/c$ for $i \in N$. Hence

$$z(C(BPP)) = \sum_{i=1}^{n} w_i/c,$$

(8.13)

so a lower bound for BPP is

Lower bound $L_3$ dominates $S(BPP, x)$ given, for a positive

$$\begin{array}{c}
\text{minimize} \\
\text{subject to}
\end{array}$$

First note that we do not allow $x_j = 0$ since this would immediately prove
then have the following.

Theorem 8.1  For any function $S(BPP, x)$ is $x_i = k$ (for any positive $k$)

Proof. Let $i = \arg \min \{ x_i : i \in I \}$; an optimal solution to $S(BPP, x)$

$$\begin{array}{c}
\text{minimize} \\
\text{subject to}
\end{array}$$

i.e. to a special case of the 0-1 $V$ the optimal solution is trivially

$$x_i = 1$$

and setting $y_i = 1$ for $i \leq x = m^i$

$$i > x.$$ Hence the choice $z = \alpha \cdot c$

the maximum value of $z$, i.e. ab-
8 Bin-packing problem

8.2 Algorithm comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>n = 3</th>
<th>n = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.300</td>
<td>1.333</td>
</tr>
<tr>
<td>2</td>
<td>1.333</td>
<td>1.250</td>
</tr>
<tr>
<td>3</td>
<td>1.300</td>
<td>1.244</td>
</tr>
<tr>
<td>4</td>
<td>1.183</td>
<td>1.150</td>
</tr>
</tbody>
</table>

Table 8.1 (taken from Coffman, et al., 1976) gives for $\alpha = \frac{1}{4}, \frac{1}{2}, \frac{3}{4}$, the ratio of the algorithms 1:2:

8.3 Lower bounds

so a lower bound for BPP is

$$L_1 = \left( \sum_{j=1}^{n} \frac{w_j}{c} \right).$$

Lower bound $L_1$ dominates the bound provided by the surrogate relaxation $S(BPP, \gamma)$ given, for a positive vector $(\gamma_i)$ of multipliers, by

$$\text{minimize} \quad z = \sum_{j=1}^{n} \gamma_j$$

subject to

$$\sum_{j=1}^{n} \gamma_j \geq \sum_{j=1}^{n} w_j, \quad j = 1, 2, \ldots, n.$$  

First note that we do not allow any multiplier, say $\gamma_j$, to take the value zero, since this would immediately produce a useless solution $x_j = 1$ for all $j \in N$. We then have the following

Theorem 8.1 For any instance of BPP the optimal vector of multipliers for $S(BPP, \gamma)$ is $\gamma_i = k$ (k a positive constant) for all $i \in N$.

Proof. Let $z = \text{arg min} \{ y_i : i \in N \}, \alpha = y_i$, and suppose that $(x_i^\ast)$ and $(y_i^\ast)$ define an optimal solution to $S(BPP, \gamma)$. We can obtain a feasible solution of the same value by setting, for each $j \in N$, $x_j^\ast = 1$ and $y_j^\ast = 0$ for $i \neq j$. Hence $S(BPP, \gamma)$ is equivalent to the problem

$$\text{minimize} \quad \sum_{j=1}^{n} \gamma_j$$

subject to

$$\sum_{j=1}^{n} \gamma_j \geq \sum_{j=1}^{n} w_j, \quad y_i = 0 \text{ or } 1, \quad i \in N,$$

i.e., to a special case of the 0-1 knapsack problem in minimization form, for which the optimal solution is trivially obtained by re-indexing the bins so that

$$\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n = \pi$$

and setting $y_i = 1$ for $i \leq s = \pi_{\text{min}} \{ i : \sum_{j=1}^{i} \pi_j \geq \alpha \sum_{j=1}^{n} w_j \}, y_i = 0$ for $i > s$. Hence the choice $\pi_i = k$ (k, any positive constant) for all $i \in N$ produces the maximum value of $s$, i.e., also of $z(S(BPP, \gamma))$.\]
Corollary 8.1 When \( r_i = k > 0 \) for all \( i \in N \), \( z(S(BPP, r)) = z(C(BPP)) \).

Proof. With this choice of multipliers, \( S(BPP, r) \) becomes

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} y_i \\
\text{subject to} & \quad \sum_{i=1}^{n} w_i \leq c \sum_{i=1}^{n} x_i, \\
& \quad y_i = 0 \text{ or } 1, \quad i \in N,
\end{align*}
\]

where optimal solution value is \( \sum_{i=1}^{n} w_i/c \). \( \square \)

Lower bound \( L \) also dominates the bound provided by the Lagrangian relaxation \( L(BPP, \mu) \) defined, for a positive vector \( \langle \mu \rangle \) of multipliers, by

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{m} y_j + \sum_{i=1}^{n} \mu_i \left( \sum_{j=1}^{m} w_j x_j - c y_j \right) \\
\text{subject to} & \quad (8.3), (8.4), (8.5).
\end{align*}
\]

(Here again no multiplier of value zero can be accepted.)

Theorem 8.2 For any instance of \( BPP \) the optimal choice of multipliers for \( L(BPP, \mu) \) is \( \mu_i = 1/c \) for all \( i \in N \).

Proof. We first prove that, given any vector \( \langle \mu \rangle \), we can obtain a better (higher) objective function value by setting, for all \( i \in N \), \( \mu_i = \mu_i \), where \( \mu = \arg \min \{ \mu_i : i \in N \} \). In fact, by writing (8.16) as

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} (1 - c \mu_i) y_i + \sum_{j=1}^{m} w_j \sum_{i=1}^{n} \mu_i x_i \\
\text{subject to} & \quad y_i = 0 \text{ or } 1, \quad i \in N,
\end{align*}
\]

we see that the two terms can be optimized separately. The optimal \( (x_i) \) values are clearly \( x_i = 0 \) for \( i \notin \mu \) and \( x_i = 1 \), for all \( i \in N \). It follows that, setting \( \mu_i = \mu_i \) for all \( i \in N \), the first term is maximized, while the value of the second is unchanged.

Hence assume \( \mu_i = k \) for all \( i \in N \) (any positive constant) and let us determine the optimal value for \( k \). \( L(BPP, \mu) \) becomes

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} (1 - k c) y_i + \sum_{j=1}^{m} w_j \\
\text{subject to} & \quad y_i = 0 \text{ or } 1, \quad i \in N,
\end{align*}
\]

8.3 Lower bounds

and its optimal solution

(a) \( y_i = 0 \) for all \( i \in N \)

(b) \( y_i = 1 \) for all \( i \in N \)

In both cases the optimal solution value is \( k = 1/c \). \( \square \)

Corollary 8.2

Proof. Immediate.

A lower bound on the optimal value of \( L(BPP, \lambda) \), can be obtained by

\[
\text{minimize} \quad \sum_{i=1}^{n} \mu_i \left( \sum_{j=1}^{m} w_j x_j - c y_j \right)
\]

which immediately dominates the bound provided by the Lagrangian relaxation \( L(BPP, \mu) \) for each bin. By

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{m} w_j \sum_{i=1}^{n} \mu_i x_i \\
\text{subject to} & \quad y_i = 0 \text{ or } 1, \quad i \in N,
\end{align*}
\]

If \( z(\lambda) > 1 \) then \( \mu_i = 1/c \) for at least \( f \in N \) for \( f = 1 \).
and its optimal solution is

(a) $y_i = 0$ for all $i \in N$, hence $z(LBPP, \mu) = k \sum_{j=1}^{c} w_j$, if $k \leq 1/c$.
(b) $y_i = 1$ for all $i \in N$, hence $z(LBPP, \mu) = n - k(n - \sum_{j=1}^{c} w_j)$, if $k \geq 1/c$.

In both cases the highest value of the objective function $\sum_{j=1}^{c} w_j / c$ is provided by $k = 1/c$. □

**Corollary 8.2** When $\mu = k = 1/c$ for all $i \in N$, $z(LBPP, \mu) = z(C(BPP))$.

**Proof.** Immediate from (8.17) and (8.13). □

A lower bound dominating $L_1$ can be obtained by dualizing in a Lagrangian fashion constraints (8.3). Given a vector $(\lambda_j)$ of multipliers, the resulting relaxation, $L(BPP, \lambda)$, can be written as

$$\begin{align*}
\text{minimize} & \quad \sum_{i \in N} \left( y_i - \sum_{j \in S_i} \lambda_j x_{ij} \right) - \sum_{j \in J} \lambda_j \\
\text{subject to} & \quad (8.2), (8.4), (8.5),
\end{align*}$$

which immediately decomposes into $n$ independent and identical problems (one for each bin). By observing that for any $y, \lambda$, $y_i$ will take the value 1 if and only if $x_i = 1$ for at least one $i$, the optimal solution is obtained by defining

$$J^c = \{ j \in N : \lambda_j < 0 \}$$

and solving the 0-1 single knapsack problem

$$\begin{align*}
\text{maximize} & \quad z(\lambda) = \sum_{j \in J^c} (-\lambda_j) x_j \\
\text{subject to} & \quad \sum_{j \in J^c} w_j q_j \leq c, \\
& \quad q_j = 0 \text{ or } 1, j \in J^c.
\end{align*}$$

If $z(\lambda) > 1$ then, for all $i \in N$, we have $y_i = 1$ and $x_i = q_i$ (with $q_i = 0$ if $j \in N \setminus J^c$) for $j \in N$; otherwise we have $y_i = x_i = 0$ for all $i, j \in N$. Hence

$$z(LBPP, \lambda) = \min (0, n(1 - z(\lambda))) - \sum_{j \in J} \lambda_j.$$
8 Bin-packing problem

8.3 Lower bounds

Proof. Each item in \( J \cup \{ \alpha+1 \} \) can be placed in any feasible bin (\( J \cup \{ \alpha+1 \} \)). Because \( \alpha \) is a bin containing an item of weight \( w_{\alpha} \), it is possible to fill \( J \cup \{ \alpha+1 \} \) by placing items in \( J \cup \{ \alpha+1 \} \) without requiring \( \lceil |J| / c \rceil + 1 \) additional bins.

Corollary 8.3 Given an integer \( \alpha \geq 0 \),

\[ L_1 = \min \{ \lfloor \alpha / c \rfloor \} \]

is a lower bound of \( z(\alpha) \).

Proof. Obvious. \[ \square \]

Lower bound \( L_2 \) defined by \( \alpha = 0 \), we have, from (8.3),

\[ L(0) = \frac{c}{2} \]

hence \( L_2 \geq L(0) \).

Computing \( L_2 \) through value, however, can be done. \( \sqrt{(4/3) + \alpha} \)

Theorem 8.4 Let \( V \) be an instance of BPP, and let \( \alpha_1, \alpha_2, \alpha_3 \) be such that \( \alpha_1 < \alpha_2 < \alpha_3 \), then

\[ L_2(\alpha) = \min \{ \alpha_1, \alpha_2, \alpha_3 \} \]

is a lower bound of \( z(\alpha) \).

Proof. If \( V = \emptyset \) the theorem is trivially true. If \( \alpha_1 \leq \alpha_2 \leq \alpha_3 \), let \( \alpha_1 \) be such that the value of \( J \) produced by \( \alpha_1 \) is less than the value of \( J \) produced by \( \alpha_2 \). Then \( \alpha_3 \) is such that the value of \( J \) produced by \( \alpha_3 \) is less than the value of \( J \) produced by \( \alpha_2 \). If \( \alpha_1 \leq \alpha_2 \leq \alpha_3 \), then \( \alpha_1 \) also dominates \( \alpha_3 \) and satisfy (8.11). \( \square \)
8 Bin-packing problem

If all $i \in N$, the resulting pack problem is in fact $\mathcal{P}(\lambda) = \sum_{i=1}^{n} w_i/c = \sum_{i=1}^{n} w_i/c$.

...and those obtained with optimization techniques...

8.3 Lower bounds

Proof: Each item in $J_1 \cup J_2$ requires a separate bin, so $|J_1| + |J_2|$ bins are needed for them in any feasible solution. Let us relax the instance by replacing $N$ with $(J_1 \cup J_2 \cup \{J_3\})$. Because of the capacity constraint, no item in $J_3$ can be assigned to a bin containing an item of $J_1$. The total residual capacity of the $|J_1|$ bins needed for the items in $J_2$ is $c - \sum_{i \in J_2} w_i$. In the best case $c$ will be completely filled by items in $J_1$, so the remaining total weight $w = \sum_{i \in J_2} w_i - c$, if any, will require $\lceil w/c \rceil$ additional bins. □

Corollary 8.3

Given any instance $I$ of BPP,

$$L_2 = \max \{ L_0 \alpha : 0 \leq \alpha \leq c/2, \alpha \text{ integer} \}$$

is a lower bound of $z(I)$. (8.20)

Proof. Obvious. □

Lower bound $L_2$ dominates $L_1$. In fact, for any instance of BPP, using the value $\alpha = 0$, we have, from (8.19),

$$L_2 = 0 + |J_2| + \max \left( 0, \left\lceil \frac{\sum_{i \in J_2} w_i - |J_2|c}{c} \right\rceil \right)$$

$$= |J_2| + \max (0, L_0 - |J_2|).$$

hence $L_2 \geq L_0 = \max \{ |J_1|, |J_2| \}$.

Computing $L_2$ through (8.20) would require a pseudo-polynomial time. The same value, however, can be determined efficiently as follows.

Theorem 8.4

Let $V$ be the set of all the distinct values $w_i \leq c/2$. Then

$$L_2 = \begin{cases} 
\alpha & \text{if } V = \emptyset, \\
\max \{ L_0 \alpha : \alpha \in V \} & \text{otherwise}.
\end{cases}$$

Proof. If $V = \emptyset$ the thesis is obvious from (8.19). Assuming $V \neq \emptyset$, we prove that, given $\alpha_1 < \alpha_2$, if $\alpha_1$ and $\alpha_2$ produce the same set $J_2$, then $L(\alpha_1) \leq L(\alpha_2)$. In fact (a) the value $|J_1| + |J_2|$ is independent of $\alpha$ (b) the value $c$ produced by $\alpha_1$ is no less than the corresponding value produced by $\alpha_2$, since set $J_2$ produced by $\alpha_2$ is a subset of set $J_2$ produced by $\alpha_1$. Hence the thesis, since only distinct values $w_i \leq c/2$ produce, when used in $\alpha$, different sets $J_2$, and each value $w_i$ dominates the values $w_1, \ldots, w_{i-1}, + 1$ (by assuming that the weights satisfy (8.11)). □
Corollary 8.4 If the items are sorted according to decreasing weights, $L_2$ can be computed in $O(n)$ time.

Proof. Let

$$j^* = \min \{ j \in N : w_j \leq c/2 \};$$

from Theorem 8.4, $L_2$ can be determined by computing $L(w_j)$ for $j = j^*, j^* + 1, \ldots, n$, by considering only distinct $w_j$ values. The computation of $L(w_j)$ clearly requires $O(n)$ time. Since $|J_1| + |J_2|$ is a constant, the computation of each new $L(w_j)$ simply requires to update $|J_1|, \sum_{j \in J_1} w_j$ and $\sum_{j \in J_2} w_j$. Hence all the updates can be computed in $O(n)$ time since they correspond to a constant time for each $j = j^* + 1, \ldots, n$. □

The average efficiency of the above computation can be improved as follows. At any iteration, let $L_2$ be the largest $L(w_j)$ value computed so far. If $|J_1| + |J_2| + \left| \sum_{j \in J_1} w_j - \left( |J_1| - \sum_{j \in J_2} w_j \right) \right| / c \leq L_2$, then (see point (b) in the proof of Theorem 8.4) no further iteration could produce a better bound, so $L_2 = L_2^*$.

Example 8.1

Consider the instance of BPP defined by

$$n = 9,$$

$$w_j = (70, 60, 50, 33, 33, 33, 11, 7, 3),$$

$$c = 100.$$ An optimal solution requires 4 bins for item sets \{1, 7, 8, 9\}, \{2, 4\}, \{3, 5\} and \{6\}, respectively.

From (8.14),

$$L_1 = \frac{100}{100} = 3.$$ In order to determine $L_2$ we compute, using (8.19) and Corollary 8.4,

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

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

since at this point we have $1 + 1 + l(170 - 40/100) = 4$, the computation can be terminated with $L_2 = 4$. □

The following procedure efficiently computes $L_2$. It is assumed that, on input, the items are sorted according to (8.11) and $w_n \leq c/2$. (If $w_n > c/2$ then, trivially, $L_2 = n$.) Figure 8.1 illustrates the meaning of the main variables of the procedure.

```python
# L2 procedure
input n, (w_j); c;
output L_2;
begin
N := \{1, \ldots, n\};
j^* := \min \{ j \in N : w_j \leq c/2 \};
if j^* = 1 then L_2 := \sum_{j \in J_2} w_j/c; 
else begin
    CF_2 := j^* - 1 (comment : CF_2 := |J_1| + |J_2|);
    j^* := \sum_{j \in J_1} w_j;
    j^* := \min \{ j \in N : j < j^* \text{ and } w_j \leq c - w_j \};
    CF_2 := j^* - j^* (comment : CF_2 = |J_2|);
    SJ_2 := \sum_{j \in J_2} w_j (comment : SJ_2 = \sum_{j \in J_2} w_j);
    j^* := j^*;
    SJ_3 := w_j;
    w_{k+1} := 0;
    while w_{j+1} = w_j do
end.
```

Figure 8.1 Main variables in procedure L2.
8 Bin-packing problem

Given weights, \( L_2 \) can be

\[ U(\omega_i) \quad \text{for } j = j^*, j^* = \text{an integer}\]

- A new L(w) clearly shows that all the updatings can be done
  in \( O(n) \) time for each

\[ (a) \quad \text{improved as follows, } \]

- So far if \( |J| + |J'| \leq n \) then \( \text{in } \Phi \) is the proof of
  \( \Phi \text{ in } \Phi \), so \( L_2 = L_2 \).

\[ (b) \quad \text{is the proof of } \]

8.3 Lower bounds

\[ w_j > \frac{c}{2} \quad w_j \leq \frac{c}{2} \]

\[ \text{Figure 8.1 Main variables in procedure L2} \]

Procedure \( L_2 \):

- Input: \( \omega_i, c \);
- Output: \( L_2 \);

\[ \begin{align*}
N & \leftarrow \{1, \ldots, n\}; \\
J & \leftarrow \min \{ j \in N : w_j \leq c/2 \}; \\
\text{If } j^* = 1 \text{ then } L_2 & \leftarrow \left\lceil \frac{\sum w_j}{c} \right\rceil \\
\text{else} & \\
\text{begin} \\
CJ 1 & \leftarrow j^* - 1 \quad \text{(comment: } CJ 1 = \{J_1\} + \{J_2\}); \\
S' & \leftarrow \sum_{j=1}^{j^*-1} w_j; \\
j^* & \leftarrow \min \{ j \in N : j < j^* \quad \text{and } w_j \leq c/2 \}; \\
\text{If } j^* = j^* \text{ then } & \\
CJ 2 & \leftarrow j^* - j' \quad \text{(comment: } CJ 2 = \{J_2\}); \\
S' & \leftarrow \sum_{j=j^*}^{j'} w_j \quad \text{(comment: } SJ_2 = \sum_{j=1}^{j^*} w_j); \\
j' & \leftarrow j' + 1; \\
S & \leftarrow w_{j'}; \\
w_{j'} & \leftarrow 0; \\
\text{while } w_{j'+1} = w_{j'} \text{ do} \\
\text{end} \end{align*} \]
begin
  $j_2 = j_2 + 1$
  $Sj_3 = Sj_3 + w_{j_2}$
end (comment $Sj_3 = Sj_2$)
$L_2 = CJ/12$
repeat
  $L_2 = \max(L_2, CJ/12 + (SJ_3 + SJ_2)/c - CJ/2)$
  $j_2 = j_2 + 1$
  if $j_2 < n$ then
  begin
    $Sj_3 = Sj_3 + w_{j_2}$
    while $w_{j_2 - 1} = w_{j_2}$ do
      begin
        $j_2 = j_2 + 1$
        $Sj_3 = Sj_3 + w_{j_2}$
      end
    end
    while $j_2 > 1$ and $w_{j_2 - 1} \leq c - w_{j_2}$ do
      begin
        $j_2 = j_2 - 1$
        $CJ_2 = CJ_2 + 1$
        $SJ_2 = SJ_2 + w_{j_2}$
      end
  end
until $j_2 > n$ or $CJ/12 + (SJ_3 + SJ_2)/c - CJ/2 \leq L_2$
end.

The worst-case performance ratio of $L_2$ is established by the following

Theorem 8.5  $\rho(L_2) = \frac{3}{4}$.

Proof. Let $I$ be any instance of BPP and $z$ its optimal solution value. We prove that $L_2 \geq L(I) \geq \frac{3}{4} z$. Hence, let $n = 0$, i.e., $J_0 = \emptyset$, $J_1 = \{ j \in N : w_j > c/2 \}$, $J_2 = N \setminus J_0$. If $J_1 = \emptyset$, then, from (8.10), $L(I) = |J_2| = n = z$. Hence $J_1 \neq \emptyset$. Let $I$ denote the instance we obtain by relaxing the integrality constraints on $x_j$ for all $j \in J_1$ and $i \in N$. It is clear that $L(I)$ is the value of the optimal solution to $I$, which can be obtained as follows. ($J_0$) bins are first initialized for the items in $J_1$. Then, for each item $j \in J_1$, let $i^*$ denote the lowest indexed bin not completely filled (if no such bin, initialize a new one) and $c(i^*) \leq c$ its residual capacity. If $w_j \leq c(i^*)$ then item $j$ is assigned to bin $i^*$; otherwise item $j$ is replaced by two items $j_1, j_2$ with $w_{j_1} = c(i^*)$ and $w_{j_2} = w_j - w_{j_1}$, item $j_1$ is assigned to bin $i^*$ and the process is continued with item $j_2$. In this solution $L(I) - 1$ items at most are split (no splitting can occur in the $L(I)th$ bin). We can now obtain a feasible solution of value $z \geq z$ to $I$ by removing the split items from the previous solution and assigning them to new bins. By the definition of $J_2$, at most $|L(I) - 1/2|$ new bins are needed, so $z \leq L(I) + (|L(I)| - 2)$, hence $\rho(L(I)) \geq \frac{3}{4}$.

To prove that the ratio is tight, consider the series of instances with $n$ even.

8.4 Reduction algorithm

$w_j = k + 1$ ($k \geq 2$)
$L_2 = LIC + 1$ = sufficiently large.

It is worthy of note that we can easily obtain an algorithm BFD of $S[X, BFD(N)]$. This solution satisfies the same worst-case performance ratio given in Section 8.6.

8.4 Reduction

The reduction technique is based on the following dominance criterion:

We define a feasible set $F_0$ and $F_I$ whose existence can be checked such situations:

Dominance Criterion

of $F_I$ into subsets $F_0$, $F_1$, $F_2$, etc., $\sum_{j \in I} w_j$ for $h = 1, 2, \ldots$.

Proof. Completing it easier than through $\sum_{j \in I} w_j$ for $h = 1, 2, \ldots$.

If a feasible set $F$ is to a bin and removing impractical. The following greater than 3 and $av$ according to decreasing
8.4 Reduction algorithms

The reduction techniques described in the present section are based on the following dominance criterion (Martello and Toth, 1990b).

We define a feasible set as a subset \( F \subseteq N \) such that \( \sum_{j \in F} w_j \leq c \). Given two feasible sets \( F_1 \) and \( F_2 \), we say that \( F_1 \) dominates \( F_2 \) if the value of the optimal solution which can be obtained by imposing a bin, say \( i^* \), the values \( x_{i^*} = 1 \) if \( i \in F_1 \) and \( x_{i^*} = 0 \) if \( i \notin F_1 \), is no greater than the value that can be obtained by forcing the values \( x_{i^*} = 1 \) if \( i \in F_1 \) and \( x_{i^*} = 0 \) if \( i \notin F_1 \). A possible way to check such situations is the following

**Dominance Criterion**

Given two distinct feasible sets \( F_1 \) and \( F_2 \), if a partition of \( F_1 \) into subsets \( P_1, \ldots, P_l \) and a subset \( \{ j_1, \ldots, j_k \} \) of \( F_2 \) exist such that \( w_j \geq \sum_{i \in P_h} w_i \) for \( h = 1, \ldots, l \), then \( F_1 \) dominates \( F_2 \).

**Proof**

Completing the solution through assignment of the items in \( N \setminus F_1 \) is easier than through assignment of the items in \( N \setminus F_2 \). In fact: (a) \( \sum_{i \in N \setminus F_1} w_i \leq \sum_{i \in N \setminus F_2} w_i \) for any feasible assignment of an item \( j \in \{ j_1, \ldots, j_k \} \subseteq F_2 \), there exists a feasible assignment of the items in \( F_2 \subseteq F_2 \) (while the opposite does not hold). □

If a feasible set \( F \) dominates all the others, then the items of \( F \) can be assigned to a bin and removed from \( N \). Checking all such situations, however, is clearly impractical. The following algorithm limits the search to sets of cardinality no greater than 3 and avoids the enumeration of useless sets. It considers the items according to decreasing weights and, for each item \( j \), it checks for the existence of a
feasible set \( F \) such that \( j \in F \), with \( |F| \leq 3 \), dominating all feasible sets containing item \( j \). Whenever such a set is found, the corresponding items are assigned to a new bin and the search continues with the remaining items. It is assumed that, on input, the items are sorted according to (8.11). On output, \( z^* \) gives the number of optimally filled bins, and, for each \( j \in N \),

\[
b_j = \begin{cases} 0 & \text{if item } j \text{ has not been assigned;} \\ \text{bin to which it has been assigned.} & \end{cases}
\]

**procedure MTRP:**

**input:** \( \{w_i\}, c \);

**output:** \( z^*, \{b_j\} \);

begin

\( N = \{1, \ldots, n\}; \)

\( N^* = \emptyset; \)

\( z^* = 0; \)

for \( j = 1 \) to \( n \) do \( b_j = 0; \)

repeat

find \( j = \min(\{k : h \in N^* \}) \);

let \( N^* = N^* \setminus \{j\} \) with \( w_j \geq \cdots \geq w_1; \)

\( F := \emptyset; \)

find the largest \( k \) such that \( w_j + \sum_{k \in F} w_k \leq c; \)

if \( k = 0 \) then \( F := \{j\} \)

else begin

\( j^* := \min(\{k : w_j + w_k \leq c\}); \)

if \( k = 1 \) or \( w_j + w_{j^*} = c \) then \( F := \{j^*, j\} \)

else if \( k = 2 \) then begin

find \( j, j_1 \in N^* \), with \( a < b \), such that

\( w_{j_1} + w_{j_2} = \max(\{w_{j_1} + w_{j_2} : j, j_1 \in N^* \}, w_{j_1} + w_{j_2} \leq c); \)

if \( w_{j_1} + w_{j_2} \leq c \) then \( F := \{j, j_1, j_2\} \)

else if \( w_{j_1} + w_{j_2} = c \) then \( F := \{j, j_1\} \)

or \( w_{j_1} + w_{j_2} \leq c \) then \( F := \{j, j_1, j_2\} \)

end;

end;

if \( F = \emptyset \) then \( N = N \cup \{j\} \)

else begin

\( z^* := z^* + 1; \)

for each \( h \in F \) do \( b_h = z^*; \)

\( N := N \setminus F \)

end

until \( N = \emptyset \)

end.

### 8.4 Reduction algorithm

At each iteration \( k \) of item \( j \). Hence it begins when \( k = 0 \) and \( F = \emptyset \).

\( w_j \geq w_1 \geq \ldots \geq w_n; \)

the \( w_1 \) sets containing \( j \).

**set \( \{j_k, j_h\} \) dominates with \( j \).**

The time complexity \( O(n^3) \) times. At each \( h \) which can easily be \( x = 0 \) and \( y \) (assuming \( y < x \), respectively.

The reduction phase \( h_2 \).

After executes \( K \) the output value of \( z^* \). item set \( \{j \in N : h \}

\( z^* + rd(u) \}

(see below) and MTRP-value \( z^* \) and \( u \) need the form

\( [z^* + z^* + \ldots \}

The following proof for \( L_3 \).

At each iteration \( k \) of the smallest item \( I \)

\( (8.11). \)

**procedure L3:**

**input:** \( \{w_i\}, c \);

**output:** \( L_3; \)

begin

\( L_3 := \emptyset; \)

\( z := 0; \)

\( \bar{N} := n; \)

for \( j = 1 \) to \( n \) do

begin

\( \bar{N} := \emptyset; \)

\( b_h := 0; \)

if \( b_h = 0 \) then

end.

end.
8 Bit-packing problem

At each iteration, \( k+1 \) gives the maximum cardinality of a feasible set containing item \( j \). Hence it immediately follows from the dominance criterion that \( F = \{ j \} \) when \( k = 0 \), and \( F = \{ j, j' \} \) when \( k = 1 \) or \( w_j + w_{j'} = c \). When \( k = 2 \), (a) if \( w_j + w_{j'} \geq w_0 \), then set \( \{ j \} \) dominates all pairs of items (and, by definition of \( j' \), all singletons) which can be packed together with \( j \), so \( \{ j \} \) dominates all feasible sets containing \( j \); (b) if \( w_j + w_0 \) and either \( b - a \leq 2 \) or \( w_j + w_{j-1} + w_{j-2} > c \) then set \( \{ j, b \} \) dominates all pairs and all singletons which can be packed together with \( j \).

The time complexity of MTRP is \( O(n^2) \). In fact, the repeat-until loop is executed \( O(n) \) times. At each iteration, the heaviest step is the determination of \( j_0 \) and \( j_2 \), which can easily be implemented so as to require \( O(n) \) time, since the pointers \( r \) and \( s \) (assuming \( r < s \)) must be moved only from left to right and from right to left, respectively.

The reduction procedure above can also be used to determine a new lower bound \( L_3 \). After execution of procedure MTRP for an instance \( I \) of BPP, let \( z' \) denote the output value of \( z' \), and \( f(z') \) the corresponding residual instance, defined by item set \( \{ j \in N : h_j = 0 \} \). It is obvious that a lower bound for \( I \) is given by \( z' + \text{LH}(z') \), where \( \text{LH}(z') \) denotes the value of any lower bound for \( h(z') \). (Note that \( z' + \text{LH}(z') \) is relaxed in some way (see below) and MTRP is applied to the relaxed instance, producing the output value \( z' \) and a residual instance \( I(1, z') \).) A lower bound for \( I \) is then \( z' + \text{LH}(z') \). Iterating the process we obtain a series of lower bounds of the form

\[
L_3 = z'_1 + z'_2 + \ldots + \text{LH}(z'_1, z'_2, \ldots, z')
\]

The following procedure computes the maximum of the above bounds, using \( L_3 \) for \( L_3 \). At each iteration, the current residual instance is relaxed through removal of the smallest item. It is assumed that on input the items are sorted according to (8.11).

**procedure L3:**

**input:** \( n, (w_j), c \);
**output:** \( L_3 \);

**begin**

\( L_3 := 0; \)
\( z := 0; \)
\( n := n; \)

**for** \( j := 1 \) **to** \( n \) **do** \( w_j := w_j; \)

**while** \( n \geq 1 \) **do**

**begin**

**call** MTRP giving \( n, (w_j) \) and \( c, \) yielding \( z' \) and \( (j_0, b) \);

\( z := z' + z'; \)
\( k := 0; \)

**for** \( j := 1 \) **to** \( k \) **do**

**if** \( b_j = 0 \) **then**

\}
begin
  \( k := k + 1; \)
  \( \overline{w}_k := \overline{w} \)
end.

\( \overline{w} := k; \)
If \( \overline{w} = 0 \) then \( L_2 := 0 \)
else call L2 giving \( n, (\overline{w}_j) \) and \( c \), yielding \( L_2; \)
\( L_3 := \max(L_2, z + L_2); \)
\( \overline{w} := \overline{w} - 1 \) (comment: removal of the smallest item)
end.

Since MTRP runs in \( O(n^2) \) time, the overall time complexity of L3 is \( O(n^3) \). It is clear that \( L_3 \geq L_2 \).

Note that only the reduction determined in the first iteration of MTRP is valid for the original instance, since the other reductions are obtained after one or more relaxations. If however, after the execution of L3, all the removed items can be assigned to the bins filled up by the executions of MTRP, then we obtain a feasible solution of value \( L_3 \), i.e., optimal.

Example 8.2
Consider the instance of BPP defined by
\[
\begin{align*}
  n &= 14, \\
  (w_j) &= (99, 94, 79, 64, 50, 46, 43, 37, 32, 19, 18, 7, 6, 3), \\
  c &= 100.
\end{align*}
\]
The first execution of MTRP gives
\[
\begin{align*}
  j &= 1: k = 0, F = \{1\}; \\
  j &= 2: k = 1, j^* = 13, F = \{2, 13\},
\end{align*}
\]
and \( F = \emptyset \) for \( j \geq 3 \). Hence
\[
\begin{align*}
  z &= 2; (\overline{b}_j) = (1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0); \\
  \text{executing L2 for the residual instance we get } L_2 &= 4, \text{ so } L_3 = 6.
\end{align*}
\]
Item 14 is now removed and MTRP is applied to item set \{3, 4, ..., 12\}, producing (indices refer to the original instance)

8.5 Exact Algorithms

\[
\begin{align*}
  j &= 3: k = 1, j^* = 10, F = \{3, 10\}; \\
  j &= 4: k = 2, j^* = 9, j_9 = 11, j_{10} = 12, \\
  j &= 5: k = 2, j^* = 6, j_6 = 7, j_{10} = 12, \\
  j &= 6: k = 2, j^* = 5, j_5 = 7, j_{10} = 12, \\
  j &= 7: k = 2, j^* = 8, j_8 = 8, j_{10} = 11, \\
  j &= 12: k = 0, F = \{12\};
\end{align*}
\]

to number the new bins with 3, 4, ..., 7.

\[
\begin{align*}
  z &= 7; (\overline{b}_j) = (1, 2, 3, 4, 5, 6, 7, 8); \\
  \text{hence } L_2 &= 0 \text{ (since } \overline{w} = 0 \text{) and the execution of MTRP yields the optimum for } \overline{w} = 0.
\end{align*}
\]

8.5 EXACT ALGORITHMS

As already mentioned, very little can be said for BPP.

Eilon and Christofides (1971) have presented an algorithm based on the following “best-fit” decision rule, assuming that bins have a fixed capacity of the next (not yet initialized) item \( j^* \) of largest weight, in turn, to bin if \( \overline{w}_j \leq b + 1 \). The lower-bound decision node.

Hung and Brown (1978) have presented a generalization of BPP to the case where the bins have variable capacities. Their branching strategy is based on assignments, which reduces the number of bins employed is again \( L_3 \).

We do not give further details on these algorithms reported in Eilon and Christofides (1971) since they can solve only small-size instances. Martello and Toth (1989) have proposed a decreasing branch strategy. The decreasing weights. The algorithm is described when they are initialized. At each decision...
8.5 Exact algorithms

\[ j = 3 : k = 1, j^* = 10, F = \{3, 10\}; \]
\[ j = 4 : k = 2, j^* = 9, j_0 = 11, j_1 = 12, F = \{4, 9\}; \]
\[ j = 5 : k = 2, j^* = 6, j_0 = 7, j_1 = 12, F = \emptyset; \]
\[ j = 6 : k = 2, j^* = 5, j_0 = 7, j_1 = 12, F = \{6, 5\}; \]
\[ j = 7 : k = 2, j^* = 8, j_0 = 8, j_1 = 11, F = \{7, 8, 11\}; \]
\[ j = 12 : k = 0, F = \{12\}; \]

numbering the new bins with 3, 4, ..., 7 we thus obtain

\[ z = 7; (b_j) = (1, 2, 3, 4, 5, 6, 7, 2, -); \]

\[ z = 7; (b_j) = (1, 2, 3, 4, 5, 6, 7, 2, -); \]

hence \( L_0 = 0 \) (since \( \Pi = 0 \)) and the execution terminates with \( L_0 = 7 \).

Noting now that the eliminated item 14 can be assigned, for example to bin 4, we conclude that all reductions are valid for the original instance. The solution obtained (with \( h_{14} = 4 \)) is also optimal, since all items are assigned. \( \square \)

8.5 EXACT ALGORITHMS

As already mentioned, very little can be found in the literature on the exact solution of BPP.

Elion and Christofides (1971) have presented a simple depth-first enumerative algorithm based on the following "best-fit decreasing" branching strategy. At any decision node, assuming that \( b \) bins have been initialized, let \((c_1, \ldots, c_b)\) denote their current residual capacities sorted by increasing value, and \( c_{\text{min}} = c_{\text{max}} = c \) the capacity of the next (not yet initialized) bin: the branching phase assigns the free item \( j^* \) of largest weight, in turn, to bins \( i_1, \ldots, i_b, i_{b+1} \), where \( j = \min \{ h : i \leq h \leq b + 1, c_{i, i + 1} = w_j \} \). Lower bound \( L_i \) (see Section 8.3.1) is used to fathom decision nodes.

Hung and Brown (1978) have presented a branch-and-bound algorithm for a generalization of BPP to the case in which the bins are allowed to have different capacities. Their branching strategy is based on a characterization of equivalent assignments, which reduces the number of explored decision nodes. The lower bound employed is again \( L_i \).

We do not give further details on these algorithms, since the computational results reported in Elion and Christofides (1971) and Hung and Brown (1978) indicate that they can solve only small-size instances.

Martello and Toth (1989) have proposed an algorithm, MTP, based on a "first-fit decreasing" branching strategy. The items are initially sorted according to decreasing weights. The algorithm indexes the bins according to the order in which they are initialized. At each decision node, the first (i.e., largest) free item is
assigned, in turn, to the feasible initialized bins (by increasing index) and to a new bin. At any forward step, (a) procedures \( L_2 \) and then \( L_3 \) are called to attempt to fathom the node and reduce the current problem; (b) when no fathoming occurs, approximate algorithms FFD, BFD (see Section 8.2) and WFD are applied to the current problem, to try and improve the best solution so far. (A Worst-Fit Decreasing (WFD) approximate algorithm for BPP sorts the items by decreasing weights and assigns each item to the feasible initialized bin (if any) of largest residual capacity.) A backtracking step implies the removal of the current item \( j' \) from its current bin \( i' \), and its assignment to the next feasible bin (but backtracking occurs if \( i' \) had been initialized by \( j' \), since initializing \( i' + 1 \) with \( j' \) would produce an identical situation). If \( z \) is the value of the current optimal solution, whenever backtracking must occur, it is performed on the last item assigned to a bin of index not greater than \( z - 2 \) (since backtracking on any item assigned to bin \( z \) or \( z - 1 \) would produce solutions requiring at least \( z + 1 \) bins).

In addition, the following dominance criterion between decision nodes is used. When the current item \( j' \) is assigned to a bin \( i' \) whose residual capacity \( c_{i'} \) is less than \( w_j + \alpha \), this assignment dominates all the assignments to \( i' \) of items \( j > j' \) which do not allow the insertion of at least one further item. Hence such assignment "closes" bin \( i' \), in the sense that, after backtracking on \( j' \), no item \( j \in \{ k > j' : w_k + \alpha > c_{i'} \} \) is assigned to \( i' \); the bin is "re-opened" when the first item \( j > j' \) for which \( w_j + \alpha \leq c_{i'} \) is considered or, if no such item exists, when the first backtracking on an item \( j' < j' \) is performed.

Since at any decision node the current residual capacities \( c_i \) of the bins are different, the computation of lower bounds \( L_2 \) and \( L_3 \) must take into account this situation. An easy way is to relax the current instance by adding one extra item of weight \( c - c_i \) to the free items for each initialized bin \( i \), and by supposing that all the bins have capacity \( c \).

Example 8.3
Consider the instance of MTP defined by

\[
\begin{align*}
n &= 10; \\
w_j &= \{49, 41, 34, 33, 29, 26, 26, 22, 20, 19\}; \\
c &= 100.
\end{align*}
\]

We define a feasible solution through vector \((b_j)\), with

\[
b_j = \text{bin to which item } j \text{ is assigned (} j = 1, \ldots, n); 
\]

Figure 8.2 gives the decision-tree produced by algorithm MTP. Initially, all lower bound computations give the value 3, while approximate algorithm FFD gives the first feasible solution.

\[
\begin{align*}
z &= 4; \\
(b_j) &= \{1, 1, 2, 2, 3, 3, 3, 3, 4\}. 
\end{align*}
\]
8 Bin-packing problem

Figure 8.3 Decision-tree for Example 8.3
corresponding to decision-nodes 1-10. No second son is generated by nodes 5-9, since this would produce a solution of value 4 or more. Nodes 11 and 12 are fathomed by lower bound $L_2$. The first son of node 2 initializes bin 2, so no further son is generated. The first son of node 13 is dominated by node 2, since in both situations no further item can be assigned to bin 1; for the same reason node 2 dominates the first son of node 15. Node 14 is fathomed by lower bound $L_2$. At node 16, procedure MTRP (called by L3) is applied to problem

$\pi = 9,$

$(w_i) = (74, 49, 34, 29, 26, 26, 22, 20, 19),$

$\bar{c} = 100,$

and optimally assigns to bin 2 the first and fifth of these items (corresponding to items 2, 4 and 6 of the original instance). Then, by executing the approximate algorithm FFD for the reduced instance

$(w_i) = (74, 49, 34, 29, 26, 22, 20, 19),$

$(\bar{c}_i) = (51, 0, 66, 100, 100, \ldots),$

where $\bar{c}_i$ denotes the residual capacity of bin $i$, we obtain

$(b_i) = (1, 1, 0, 0, 0, 0, 0, 0),$

hence an overall solution of value 3, i.e. optimal. □

The Fortran implementation of algorithm MTP is included in the present volume.

8.6 COMPUTATIONAL EXPERIMENTS

In this section we examine the average computational performance of the lower bounds (Sections 8.3-8.4) and of the exact algorithm MTP (Section 8.5). The procedures have been coded in Fortran IV and run on an HP 9000/8400 using option "-oa" for the compiler on three classes of randomly generated item sizes:

Class 1: $w_j$ uniformly random in [1, 100];
Class 2: $w_j$ uniformly random in [20, 100];
Class 3: $w_j$ uniformly random in [50, 100].
Table 8.2  \( C = 100 \), HP 9000/840 in seconds. Average times / Average percentage errors (exact solution values found) over 20 problems

<table>
<thead>
<tr>
<th>Class</th>
<th>n</th>
<th>Time</th>
<th>% err(opt)</th>
<th>Time</th>
<th>% err(opt)</th>
<th>Time</th>
<th>% err(opt)</th>
<th>Time</th>
<th>% err(opt)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50</td>
<td>0.003</td>
<td>28.075(0)</td>
<td>0.001</td>
<td>6.413(2)</td>
<td>0.001</td>
<td>0.519(1)</td>
<td>0.001</td>
<td>0.000(20)</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>0.009</td>
<td>24.530(0)</td>
<td>0.001</td>
<td>3.549(0)</td>
<td>0.001</td>
<td>0.877(11)</td>
<td>0.006</td>
<td>0.196(18)</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.020</td>
<td>20.594(0)</td>
<td>0.002</td>
<td>2.593(1)</td>
<td>0.001</td>
<td>0.291(14)</td>
<td>0.013</td>
<td>0.149(17)</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.061</td>
<td>19.175(0)</td>
<td>0.002</td>
<td>2.487(0)</td>
<td>0.003</td>
<td>0.394(12)</td>
<td>0.033</td>
<td>0.078(16)</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.136</td>
<td>18.452(0)</td>
<td>0.003</td>
<td>1.127(0)</td>
<td>0.007</td>
<td>0.159(7)</td>
<td>0.091</td>
<td>0.000(15)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.004</td>
<td>27.814(0)</td>
<td>0.001</td>
<td>7.938(0)</td>
<td>0.001</td>
<td>0.308(18)</td>
<td>0.002</td>
<td>0.000(20)</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>0.011</td>
<td>23.641(0)</td>
<td>0.001</td>
<td>6.548(0)</td>
<td>0.002</td>
<td>0.306(16)</td>
<td>0.003</td>
<td>0.000(20)</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.022</td>
<td>20.192(0)</td>
<td>0.001</td>
<td>6.084(0)</td>
<td>0.001</td>
<td>0.352(13)</td>
<td>0.009</td>
<td>0.059(19)</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.059</td>
<td>19.506(0)</td>
<td>0.001</td>
<td>5.558(0)</td>
<td>0.003</td>
<td>0.189(10)</td>
<td>0.019</td>
<td>0.032(18)</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.139</td>
<td>18.443(0)</td>
<td>0.002</td>
<td>4.805(0)</td>
<td>0.005</td>
<td>0.168(5)</td>
<td>0.058</td>
<td>0.032(16)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.004</td>
<td>24.281(0)</td>
<td>0.001</td>
<td>23.300(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.001</td>
<td>0.000(20)</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>0.010</td>
<td>21.612(0)</td>
<td>0.001</td>
<td>24.012(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.003</td>
<td>0.000(20)</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.023</td>
<td>19.485(0)</td>
<td>0.001</td>
<td>24.003(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.005</td>
<td>0.000(20)</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.062</td>
<td>18.841(0)</td>
<td>0.001</td>
<td>24.044(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.016</td>
<td>0.000(20)</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.146</td>
<td>18.153(0)</td>
<td>0.002</td>
<td>24.285(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.038</td>
<td>0.000(20)</td>
</tr>
</tbody>
</table>
Table 8.3  \( C = 120 \). HP 9000/840 in seconds. Average times / Average percentage errors (exact solution values found) over 20 problems

<table>
<thead>
<tr>
<th>Class</th>
<th>( n )</th>
<th>( LBD )</th>
<th>Time</th>
<th>% err(opt)</th>
<th>( L_1 )</th>
<th>Time</th>
<th>% err(opt)</th>
<th>( L_2 )</th>
<th>Time</th>
<th>% err(opt)</th>
<th>( L_3 )</th>
<th>Time</th>
<th>% err(opt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>0.004</td>
<td>29.8320(0)</td>
<td>0.001</td>
<td>3.788(10)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.005</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.008</td>
<td>24.5620(0)</td>
<td>0.001</td>
<td>1.990(10)</td>
<td>0.002</td>
<td>0.474(16)</td>
<td>0.014</td>
<td>0.357(17)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.020</td>
<td>21.1800(0)</td>
<td>0.001</td>
<td>0.732(13)</td>
<td>0.003</td>
<td>0.165(17)</td>
<td>0.042</td>
<td>0.090(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.055</td>
<td>19.7350(0)</td>
<td>0.001</td>
<td>0.558(13)</td>
<td>0.006</td>
<td>0.116(15)</td>
<td>0.232</td>
<td>0.047(18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.126</td>
<td>18.5370(0)</td>
<td>0.002</td>
<td>0.105(17)</td>
<td>0.008</td>
<td>0.035(17)</td>
<td>0.983</td>
<td>0.024(18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>0.005</td>
<td>28.5140(0)</td>
<td>0.001</td>
<td>7.924(0)</td>
<td>0.001</td>
<td>0.976(15)</td>
<td>0.003</td>
<td>0.408(18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.009</td>
<td>23.8810(0)</td>
<td>0.001</td>
<td>3.714(2)</td>
<td>0.001</td>
<td>0.476(15)</td>
<td>0.005</td>
<td>0.091(19)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.019</td>
<td>20.2120(0)</td>
<td>0.001</td>
<td>3.326(0)</td>
<td>0.001</td>
<td>0.580(10)</td>
<td>0.009</td>
<td>0.050(19)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.060</td>
<td>19.3040(0)</td>
<td>0.001</td>
<td>3.106(0)</td>
<td>0.004</td>
<td>0.216(11)</td>
<td>0.023</td>
<td>0.039(17)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.126</td>
<td>18.6070(0)</td>
<td>0.003</td>
<td>1.386(0)</td>
<td>0.007</td>
<td>0.239(8)</td>
<td>0.062</td>
<td>0.080(13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>0.004</td>
<td>25.2080(0)</td>
<td>0.001</td>
<td>24.145(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.002</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.010</td>
<td>21.5050(0)</td>
<td>0.001</td>
<td>22.344(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.004</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.023</td>
<td>19.7520(0)</td>
<td>0.001</td>
<td>22.155(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.007</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.064</td>
<td>18.6140(0)</td>
<td>0.002</td>
<td>22.035(0)</td>
<td>0.002</td>
<td>0.012(19)</td>
<td>0.017</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.147</td>
<td>18.3230(0)</td>
<td>0.003</td>
<td>21.775(0)</td>
<td>0.003</td>
<td>0.006(12)</td>
<td>0.036</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.4  \( C = 150 \). HP 9000/840 in seconds. Average times / Average percentage errors (exact solution values found) over 20 problems

<table>
<thead>
<tr>
<th>Class</th>
<th>( n )</th>
<th>( LBD )</th>
<th>Time</th>
<th>% err(opt)</th>
<th>( L_1 )</th>
<th>Time</th>
<th>% err(opt)</th>
<th>( L_2 )</th>
<th>Time</th>
<th>% err(opt)</th>
<th>( L_3 )</th>
<th>Time</th>
<th>% err(opt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>0.004</td>
<td>29.8320(0)</td>
<td>0.001</td>
<td>3.788(10)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.005</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.008</td>
<td>24.5620(0)</td>
<td>0.001</td>
<td>1.990(10)</td>
<td>0.002</td>
<td>0.474(16)</td>
<td>0.014</td>
<td>0.357(17)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.020</td>
<td>21.1800(0)</td>
<td>0.001</td>
<td>0.732(13)</td>
<td>0.003</td>
<td>0.165(17)</td>
<td>0.042</td>
<td>0.090(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.055</td>
<td>19.7350(0)</td>
<td>0.001</td>
<td>0.558(13)</td>
<td>0.006</td>
<td>0.116(15)</td>
<td>0.232</td>
<td>0.047(18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.126</td>
<td>18.5370(0)</td>
<td>0.002</td>
<td>0.105(17)</td>
<td>0.008</td>
<td>0.035(17)</td>
<td>0.983</td>
<td>0.024(18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>0.005</td>
<td>28.5140(0)</td>
<td>0.001</td>
<td>7.924(0)</td>
<td>0.001</td>
<td>0.976(15)</td>
<td>0.003</td>
<td>0.408(18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.009</td>
<td>23.8810(0)</td>
<td>0.001</td>
<td>3.714(2)</td>
<td>0.001</td>
<td>0.476(15)</td>
<td>0.005</td>
<td>0.091(19)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.019</td>
<td>20.2120(0)</td>
<td>0.001</td>
<td>3.326(0)</td>
<td>0.001</td>
<td>0.580(10)</td>
<td>0.009</td>
<td>0.050(19)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.060</td>
<td>19.3040(0)</td>
<td>0.001</td>
<td>3.106(0)</td>
<td>0.004</td>
<td>0.216(11)</td>
<td>0.023</td>
<td>0.039(17)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.126</td>
<td>18.6070(0)</td>
<td>0.003</td>
<td>1.386(0)</td>
<td>0.007</td>
<td>0.239(8)</td>
<td>0.062</td>
<td>0.080(13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>0.004</td>
<td>25.2080(0)</td>
<td>0.001</td>
<td>24.145(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.002</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.010</td>
<td>21.5050(0)</td>
<td>0.001</td>
<td>22.344(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.004</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.023</td>
<td>19.7520(0)</td>
<td>0.001</td>
<td>22.155(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
<td>0.007</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.064</td>
<td>18.6140(0)</td>
<td>0.002</td>
<td>22.035(0)</td>
<td>0.002</td>
<td>0.012(19)</td>
<td>0.017</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.147</td>
<td>18.3230(0)</td>
<td>0.003</td>
<td>21.775(0)</td>
<td>0.003</td>
<td>0.006(12)</td>
<td>0.036</td>
<td>0.000(20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 8.4  \( C = 150 \), HP 9000/840 in seconds, Average times / Average percentage errors (exact solution values found) over 20 problems

<table>
<thead>
<tr>
<th>Class</th>
<th>( n )</th>
<th>( LRBD )</th>
<th>( L_1 )</th>
<th>( L_2 )</th>
<th>( L_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>% err(opt)</td>
<td>Time</td>
<td>% err(opt)</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>0.003</td>
<td>31.832(0)</td>
<td>0.001</td>
<td>0.488(18)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.009</td>
<td>25.634(0)</td>
<td>0.001</td>
<td>0.000(20)</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.016</td>
<td>20.984(0)</td>
<td>0.001</td>
<td>0.727(14)</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.056</td>
<td>20.007(0)</td>
<td>0.002</td>
<td>0.000(20)</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.119</td>
<td>19.189(0)</td>
<td>0.003</td>
<td>0.000(20)</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>0.004</td>
<td>31.408(0)</td>
<td>0.001</td>
<td>0.617(18)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.009</td>
<td>23.652(0)</td>
<td>0.001</td>
<td>0.727(14)</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.020</td>
<td>20.984(0)</td>
<td>0.001</td>
<td>0.978(0)</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.059</td>
<td>19.837(0)</td>
<td>0.002</td>
<td>1.155(0)</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.128</td>
<td>18.416(0)</td>
<td>0.003</td>
<td>1.224(0)</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>0.003</td>
<td>27.941(0)</td>
<td>0.001</td>
<td>8.851(1)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.009</td>
<td>23.247(0)</td>
<td>0.001</td>
<td>4.302(4)</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.021</td>
<td>20.671(0)</td>
<td>0.001</td>
<td>3.856(0)</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.061</td>
<td>19.459(0)</td>
<td>0.001</td>
<td>3.480(0)</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.132</td>
<td>18.527(0)</td>
<td>0.003</td>
<td>1.687(0)</td>
</tr>
</tbody>
</table>
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 instances have been generated.

In Tables 8.2–8.4 we examine the behavior of lower bounds LBFD, L1, L2 and L3. The entries give, for each bound, the average computing time (expressed in seconds and not comprehensive of the sorting time), the average percentage error and, in brackets, the number of times the value of the lower bound coincided with that of the optimal solution. LBFD requires times almost independent of the data generation and, because of the good approximation produced by the best-fit decreasing algorithm, gives high errors, tending to \( \frac{1}{\log n} \) when \( n \) grows. \( L_1 \) obviously requires very small times, practically independent of the data generation; the tightness improves when the ratio \( c / \min \{w_i \} \) grows, since the computation is based on continuous relaxation of the problem. \( L_2 \) requires slightly higher times, but produces tighter values; for class 1 it improves when \( c \) grows, for classes 2 and 3 it gets worse when \( c \) grows. The times required by \( L_3 \) are in general comparatively very high (because of the iterated execution of reduction procedure MTRP), and clearly grow both with \( n \) and \( c \); the approximation produced is generally very good, with few exceptions.

Note that the problems generated can be considered "hard", since few items are packed in each bin. Using the value \( c = 1000 \), \( L_1 \) requires the same times and almost always produces the optimal solution value.

Table 8.5 gives the results obtained by the exact algorithm MTP for the instances used for the previous tables. The entries give average running time (expressed in seconds and comprehensive of the sorting time) and average number of nodes.

<table>
<thead>
<tr>
<th>Class</th>
<th>50 nodes</th>
<th>100 nodes</th>
<th>200 nodes</th>
<th>500 nodes</th>
<th>1000 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td>2</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td>3</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
</tr>
</tbody>
</table>

8.6 Computational cost

explored in the beginning of the algorithm for the corresponding entry (the average value).
When less than \( n = 10 \)

All the instances produced the optimal solution except for Class 1 for \( c = 10 \).
Worth noting is the good performance of the approximate algorithm.
considered: \( c = 100 \), \( c = 120 \),
for different values of \( n \) (\( n \) = generated.
If lower bounds LBF, L1, L2
range computing time (expressed
1 time), the average percentage
value of the lower bound coincided
in times almost independent of
approximation produced by the
ending \( \frac{1}{2} \) when \( n \) grows, \( L_1 \)
dependent of the data generation;
grows, since the computation is
requires slightly higher times,
when \( c \) grows, for classes 2 and
\( L_3 \) are in general comparatively
reduction procedure MTRP), and
produced is generally very good,
named “hard”, since few items are
1 requires the same times and
algorithm MTP for the instances
range running time (expressed in
and average number of nodes

<table>
<thead>
<tr>
<th>( c = 150 )</th>
<th>( c = 100 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>Time</td>
</tr>
<tr>
<td>0</td>
<td>0.996</td>
</tr>
<tr>
<td>358</td>
<td>0.136</td>
</tr>
<tr>
<td>6</td>
<td>0.140</td>
</tr>
<tr>
<td>885</td>
<td>2.124</td>
</tr>
<tr>
<td>244</td>
<td>8.958</td>
</tr>
<tr>
<td>1</td>
<td>0.183</td>
</tr>
<tr>
<td>9</td>
<td>26.599 (15)</td>
</tr>
<tr>
<td>18</td>
<td>89.451 (7)</td>
</tr>
<tr>
<td>1060</td>
<td>16</td>
</tr>
<tr>
<td>4774</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0.005</td>
</tr>
<tr>
<td>0</td>
<td>0.010</td>
</tr>
<tr>
<td>0</td>
<td>0.018</td>
</tr>
<tr>
<td>0</td>
<td>0.051</td>
</tr>
<tr>
<td>0</td>
<td>0.105</td>
</tr>
</tbody>
</table>

explored in the branch-decision tree. A time limit of 100 seconds was assigned
to the algorithm for each problem instance. When the time limit occurred, the
corresponding entry gives, in brackets, the number of instances solved to optimality
(the average values are computed by also considering the interrupted instances).
When less than half of the 20 instances generated for an entry was completed,
larger values of \( n \) were not considered.

All the instances of Class 3 were solved very quickly, since procedure L3 always
produced the optimal solution. For Class 1 the results are very satisfactory, with
few exceptions. On Class 2, the behaviour of the algorithm was better than on
Class 1 for \( c = 100 \), about the same for \( c = 120 \), and clearly worse for \( c = 150 \).
Worth noting is that in only a few cases the optimal solution was found by the
approximate algorithms used.