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 n *knapsacks* (or *bins*), with

$$w_j = \textit{weight} \text{ of item } j,$$

$$c = \textit{capacity} \text{ of each bin,}$$

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

$$\text{minimize} \quad z = \sum_{i=1}^{n} y_i \tag{8.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_{ij} \le c y_i, \quad i \in N = \{1, \dots, n\}, \tag{8.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j \in N, \tag{8.3}$$

$$y_i = 0 \text{ or } 1, \quad i \in N, \tag{8.4}$$

$$x_{ij} = 0 \text{ or } 1, \quad i \in N, j \in N, \tag{8.5}$$

where

$$y_i = \begin{cases} 1 & \text{if bin } i \text{ is used;} \\ 0 & \text{otherwise,} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if item } j \text{ is assigned to bin } i; \\ 0 & \text{otherwise.} \end{cases}$$

We will suppose, as is usual, that the weights w_j are positive integers. Hence, without loss of generality, we will also assume that

$$c \text{ is a positive integer,} \tag{8.6}$$

$$w_j \le c \quad \text{for } j \in N. \tag{8.7}$$

If assumption (8.6) is violated, c can be replaced by $\lfloor c \rfloor$. If an item violates assumption (8.7), then the instance is trivially infeasible. There is no easy way, instead, of transforming an instance so as to handle negative weights.

For the sake of simplicity we will also assume that, in any feasible solution, the lowest indexed bins are used, i.e. $y_i \ge y_{i+1}$ for $i = 1, \dots, n-1$.

Almost the totality of the literature on BPP is concerned with approximate algorithms and their performance. A thorough analysis of such results would require a separate book (the brilliant survey by Coffman, Garey and Johnson (1984), to which the reader is referred, includes a bibliography of more than one hundred references, and new results continue to appear in the literature). In Section 8.2 we briefly summarize the classical results on approximate algorithms. The remainder of the chapter is devoted to lower bounds (Section 8.3), reduction procedures (Section 8.4) and exact algorithms (Section 8.5), on which very little can be found in the literature. Computational experiments are reported in Section 8.6.

8.2 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 $2, \dots, n$ 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

$$NF(I) \le 2\, z(I), \tag{8.8}$$

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 $r(\mathrm{NF}) = 2$. 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

$$\frac{A(I)}{z(I)} \le r(A) \text{ for all instances } I,$$

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 proved in Johnson, Demers, Ullman, Garey and Graham (1974) that

$$FF(I) \leq \frac{17}{10} z(I) + 2 \tag{8.9}$$

for all instances I of BPP, and that there exist instances I, with $z(I)$ arbitrarily large, for which

$$FF(I) > \frac{17}{10} z(I) - 8. \tag{8.10}$$

Because of the constant term in (8.9), as well as in analogous results for other algorithms, the worst-case performance ratio cannot give complete information on the worst-case behaviour. Instead, for the bin packing problem, the *asymptotic worst-case performance ratio* is commonly used. For an approximate algorithm A, this is defined as the minimum real number $r^{\infty}(A)$ such that, for some positive integer k,

$$\frac{A(I)}{z(I)} \leq r^{\infty}(A) \text{ for all instances } I \text{ satisfying } z(I) \geq k;$$

it is then clear, from (8.9)–(8.10), that $r^{\infty}(\text{FF}) = \frac{17}{10}$.

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 (breaking 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 $r^{\infty}(\text{BF}) = \frac{17}{10}$.

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 l; (c) labels at the nodes allow searching for a given leaf value, updating it, or inserting a new leaf in $O(l)$ 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 $O(\log n)$ time, since the number of leaves is bounded by n.

Assume now that the items are sorted so that

$$w_1 \geq w_2 \geq \ldots \geq w_n, \tag{8.11}$$

and then NF or FF, 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

$$FFD(I) \leq \frac{11}{9} z(I) + 4 \tag{8.12}$$

Table 8.1 Asymptotic worst-case performance ratios of bin-packing algorithms

Algorithm	Time complexity	r^{∞}	$r^{\infty}_{1/2}$	$r^{\infty}_{1/3}$	$r^{\infty}_{1/4}$
NF	$O(n)$	2.000	2.000	1.500	1.333...
FF	$O(n \log n)$	1.700	1.500	1.333...	1.250
BF	$O(n \log n)$	1.700	1.500	1.333...	1.250
NFD	$O(n \log n)$	1.691...	1.424...	1.302...	1.234...
FFD	$O(n \log n)$	1.222...	1.183...	1.183...	1.150
BFD	$O(n \log n)$	1.222...	1.183...	1.183...	1.150

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^{∞}_{α} of the asymptotic worst-case performance ratio of the algorithms when applied to instances satisfying $\min_{1 \le j \le n} \{w_j\} \le \alpha c$.

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

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

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

$$0 \le y_i \le 1, \qquad i \in N,$$

$$0 \le x_{ij} \le 1, \qquad i \in N, j \in N,$$

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

$$z(C(BPP)) = \sum_{i=1}^{n} w_i/c, \tag{8.13}$$

so a lower bound for BPP is

$$L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil . \tag{8.14}$$

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

$$\begin{aligned} &\text{minimize} && z = \sum_{i=1}^{n} y_i \\ &\text{subject to} && \sum_{i=1}^{n} \pi_i \sum_{j=1}^{n} w_j x_{ij} \le c \sum_{i=1}^{n} \pi_i y_i, \qquad (8.15) \\ &&& (8.3), (8.4), (8.5). \end{aligned}$$

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

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

Proof. Let $\bar{\imath} = \arg\min \{\pi_i : i \in N\}$, $\alpha = \pi_{\bar{\imath}}$, and suppose that (y_i^*) and (x_{ij}^*) define an optimal solution to $S(BPP, \pi)$. We can obtain a feasible solution of the same value by setting, for each $j \in N$, $x_{\bar{\imath}j}^* = 1$ and $x_{ij}^* = 0$ for $i \ne \bar{\imath}$. Hence $S(BPP, \pi)$ is equivalent to the problem

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{n} y_i \\ &\text{subject to} && \sum_{i=1}^{n} \pi_i y_i \ge \frac{\alpha}{c} \sum_{j=1}^{n} w_j, \\ &&& y_i = 0 \text{ or } 1,\ i \in N, \end{aligned}$$

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

$$\pi_1 \ge \pi_2 \ge \ldots \ge \pi_n \ (\equiv \alpha)$$

and setting $y_i = 1$ for $i \le s = \min \{l \in N : \sum_{r=1}^{l} \pi_r \ge (\alpha/c) \sum_{j=1}^{n} w_j\}$, $y_i = 0$ for $i > s$. Hence the choice $\pi_i = \alpha$ ($= k$, any positive constant) for all $i \in N$ produces the maximum value of s, i.e. also of $z(S(BPP, \pi))$. □

Corollary 8.1 *When $\pi_i = k > 0$ for all $i \in N$, $z(S(BPP, \pi)) = z(C(BPP))$.*

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

$$\text{minimize} \quad \sum_{i=1}^{n} y_i$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j \le c \sum_{i=1}^{n} y_i,$$

$$y_i = 0 \text{ or } 1, \quad i \in N,$$

whose optimal solution value is $\sum_{j=1}^{n} w_j / c$. □

Lower bound L_1 also dominates the bound provided by the Lagrangian relaxation $L(BPP, \mu)$ defined, for a positive vector (μ_i) of multipliers, by

$$\text{minimize} \quad \sum_{i=1}^{n} y_i + \sum_{i=1}^{n} \mu_i \left(\sum_{j=1}^{n} w_j x_{ij} - c y_i \right) \tag{8.16}$$

subject to (8.3), (8.4), (8.5).

(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 (μ_i), we can obtain a better (higher) objective function value by setting, for all $i \in N$, $\mu_i = \mu_{\bar{\imath}}$, where $\bar{\imath} = \arg\min \{ \mu_i : i \in N \}$. In fact, by writing (8.16) as

$$\text{minimize} \quad \sum_{i=1}^{n} (1 - c\mu_i) y_i + \sum_{j=1}^{n} w_j \sum_{i=1}^{n} \mu_i x_{ij},$$

we see that the two terms can be optimized separately. The optimal (x_{ij}) values are clearly $x_{ij} = 0$ for $i \ne \bar{\imath}$ and $x_{\bar{\imath} j} = 1$, for all $j \in N$. It follows that, setting $\mu_i = \mu_{\bar{\imath}}$ 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 $i \in N$ (k any positive constant) and let us determine the optimal value for k. $L(BPP, \mu)$ becomes

$$\text{minimize} \quad \sum_{i=1}^{n} (1 - ck) y_i + k \sum_{j=1}^{n} w_j \tag{8.17}$$

$$\text{subject to} \quad y_i = 0 \text{ or } 1, \quad i \in N,$$

and its optimal solution is

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

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

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

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 (λ_j) of multipliers, the resulting relaxation, $L(BPP, \lambda)$, can be written as

$$\text{minimize} \quad \sum_{i=1}^{n} \left(y_i + \sum_{j=1}^{n} \lambda_j x_{ij} \right) - \sum_{j=1}^{n} \lambda_j \tag{8.18}$$

$$\text{subject to} \quad (8.2), (8.4), (8.5),$$

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

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

and solving the 0-1 single knapsack problem

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

If $z(\lambda) > 1$ then, for all $i \in N$, we have $y_i = 1$ and $x_{ij} = q_j$ (with $q_j = 0$ if $j \in N \backslash J^<$) for $j \in N$; otherwise we have $y_i = x_{ij} = 0$ for all $i, j \in N$. Hence

$$z(L(BPP, \lambda)) = \min(0, n(1 - z(\lambda))) - \sum_{j=1}^{n} \lambda_j .$$

It is now easy to see that, with the choice $\overline{\lambda}_j = -w_j/c$ for all $j \in N$, the resulting bound coincides with L_1. The objective function of the knapsack problem is in fact $(\sum_{j \in J^<} w_j q_j)/c$, with $J^< \equiv N$, so $z(\bar{\lambda}) \leq 1$ and $z(L(BPP, \overline{\lambda})) = \sum_{j=1}^{n} w_j/c = z(C(BPP))$.

Better multipliers can be obtained by using subgradient optimization techniques. Computational experiments, however, gave results worse than those obtained with the bounds described in the following sections.

8.3.2 A stronger lower bound

We first observe that the worst-case performance ratio of L_1 can easily be established as $r(L_1) = \frac{1}{2}$. Note, in fact, that in any optimal solution (x_{ij}) of value z, at most one bin (say the zth) can have $\sum_{j=1}^{n} w_j x_{zj} \leq c/2$ since, if two such bins existed, they could be replaced by a single bin. Hence $\sum_{j=1}^{n} w_j > \sum_{i=1}^{z-1} \sum_{j=1}^{n} w_j x_{ij} > (z-1)c/2$, from which $z \leq \lceil 2 \sum_{j=1}^{n} w_j/c \rceil$ and, from (8.14), $L_1/z \geq \frac{1}{2}$. To see that the ratio is tight, it is enough to consider the series of instances with $w_j = k+1$ for all $j \in N$ and $c = 2k$, for which $z = n$ and $L_1 = \lceil n(k+1)/2k \rceil$, so the ratio L_1/z can be arbitrarily close to $\frac{1}{2}$ for k sufficiently large.

Despite its simplicity, L_1 can be expected to have good average behaviour for problems where the weights are sufficiently small with respect to the capacity, since in such cases the evaluation is not greatly affected by the relaxation of the integrality constraints. For problems with larger weights, in which few items can be allocated, on average, to each bin, Martello and Toth (1990b) have proposed the following better bound.

Theorem 8.3 *Given any instance I of BPP, and any integer α, $0 \leq \alpha \leq c/2$, let*

$$J_1 = \{j \in N : w_j > c - \alpha\},$$

$$J_2 = \{j \in N : c - \alpha \geq w_j > c/2\},$$

$$J_3 = \{j \in N : c/2 \geq w_j \geq \alpha\};$$

then

$$L(\alpha) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j \in J_3} w_j - (|J_2|c - \sum_{j \in J_2} w_j)}{c} \right\rceil\right) \tag{8.19}$$

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

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 $|J_2|$ bins needed for the items in J_2 is $\overline{c} = |J_2|c - \sum_{j \in J_2} w_j$. In the best case $\overline{c}$ will be completely filled by items in J_3, so the remaining total weight $\overline{w} = \sum_{j \in J_3} w_j - \overline{c}$, if any, will require $\lceil \overline{w}/c \rceil$ additional bins. □

Corollary 8.3 *Given any instance I of BPP,*

$$L_2 = \max \{L(\alpha) : 0 \le \alpha \le c/2,\ \alpha \textit{ integer}\} \tag{8.20}$$

is a lower bound of z(I).

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),

$$\begin{aligned} L(0) &= 0 + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j \in N} w_j - |J_2|c}{c} \right\rceil\right) \\ &= |J_2| + \max(0, L_1 - |J_2|), \end{aligned}$$

hence $L_2 \ge L(0) = \max(|J_2|, L_1)$.

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 $w_j \le c/2$. Then*

$$L_2 = \begin{cases} n & \textit{if } V = \emptyset; \\ \max \{L(\alpha) : \alpha \in V\} & \textit{otherwise}. \end{cases}$$

Proof. If $V = \emptyset$ the thesis is obvious from (8.19). Assuming $V \neq \emptyset$, we prove that, given $\alpha_1 < \alpha_2$, if α_1 and α_2 produce the same set J_3, then $L(\alpha_1) \le L(\alpha_2)$. In fact: (a) the value $|J_1| + |J_2|$ is independent of α; (b) the value $(|J_2|c - \sum_{j \in J_2} w_j)$ produced by α_1 is no less than the corresponding value produced by α_2, since set J_2 produced by α_2 is a subset of set J_2 produced by α_1. Hence the thesis, since only distinct values $w_j \le c/2$ produce, when used as α, different sets J_3, and each value w_j dominates the values $w_j - 1, \dots, w_{j+1} + 1$ (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

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

from Theorem 8.4, L_2 can be determined by computing $L(w_j)$ for $j = j^*, j^* + 1, \dots, n$, 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 $|J_2|$, $\sum_{j \in J_3} w_j$ and $\sum_{j \in J_2} w_j$. Hence all the updatings can be computed in $O(n)$ time since they correspond to a constant time for each $j = j^* + 1, \dots, n$. □

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 $|J_1| + |J_2| + \lceil (\sum_{j=j^*}^{n} w_j - (|J_2|c - \sum_{j \in J_2} w_j))/c \rceil \le L_2^*$, 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),

$$L_1 = \lceil 300/100 \rceil = 3.$$

In order to determine L_2 we compute, using (8.19) and Corollary 8.4,

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

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

since at this point we have $1 + 1 + \lceil (170 - 40)/100 \rceil = 4$, 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 $w_n \le c/2$. (If $w_n > c/2$ then, trivially, $L_2 = n = z$.) Figure 8.1 illustrates the meaning of the main variables of the procedure.

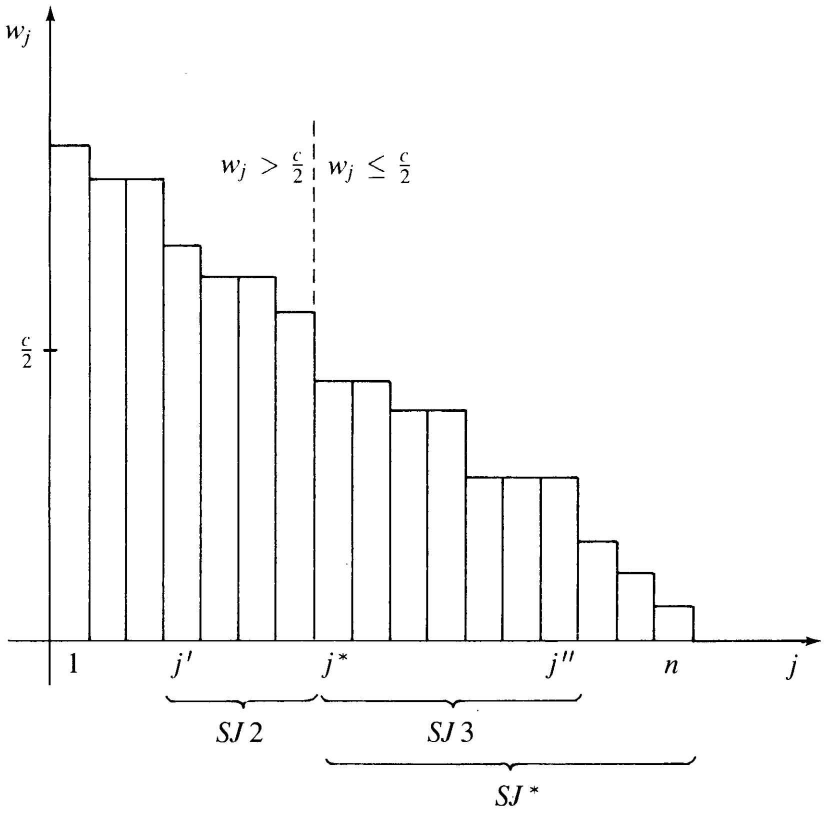

Figure 8.1 Main variables in procedure L2

```
procedure L2:
input: $n, (w_j), c$;
output: $L_2$;
begin
    $N := \{1, \dots, n\}$;
    $j^* :=$ min$\{j \in N : w_j \le c/2\}$;
    if $j^* = 1$ then $L_2 := \lceil \sum_{j=1}^{n} w_j / c \rceil$
    else
        begin
            $CJ12 := j^* - 1$ (comment : $CJ12 = |J_1| + |J_2|$);
            $SJ^* := \sum_{j=j^*}^{n} w_j$;
            $j' :=$ min$\{j \in N : j < j^*$ and $w_j \le c - w_{j^*}\}$($j' := j^*$ if no such $w_j$);
            $CJ2 := j^* - j'$ (comment : $CJ2 = |J_2|$) ;
            $SJ2 := \sum_{j=j'}^{j^*-1} w_j$ (comment : $SJ2 = \sum_{j \in J_2} w_j$);
            $j'' := j^*$;
            $SJ3 := w_{j''}$;
            $w_{n+1} := 0$;
            while $w_{j''+1} = w_{j''}$ do
```

```
                begin
                    $j'' := j'' + 1$;
                    $SJ3 := SJ3 + w_{j''}$
                end (comment : $SJ3 = \sum_{j \in J_3} w_j$);
            $L_2 := CJ12$;
            repeat
                $L_2 := \max(L_2, CJ12 + \lceil (SJ3 + SJ2)/c - CJ2 \rceil)$;
                $j'' := j'' + 1$;
                if $j'' \le n$ then
                    begin
                        $SJ3 := SJ3 + w_{j''}$;
                        while $w_{j''+1} = w_{j''}$ do
                            begin
                                $j'' := j'' + 1$;
                                $SJ3 := SJ3 + w_{j''}$
                            end;
                        while $j' > 1$ and $w_{j'-1} \le c - w_{j''}$ do
                            begin
                                $j' := j' - 1$;
                                $CJ2 := CJ2 + 1$;
                                $SJ2 := SJ2 + w_{j'}$
                            end
                    end
            until $j'' > n$ or $CJ12 + \lceil (SJ^* + SJ2)/c - CJ2 \rceil \le L_2$
        end
end.
```

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

Theorem 8.5 $r(L_2) = \frac{2}{3}$.

Proof. Let I be any instance of BPP and z its optimal solution value. We prove that $L_2 \ge L(0) \ge \frac{2}{3}z$. Hence, let $\alpha = 0$, i.e. $J_1 = \emptyset$, $J_2 = \{j \in N : w_j > c/2\}$, $J_3 = N \backslash J_2$. If $J_3 = \emptyset$, then, from (8.19), $L(0) = |J_2| = n = z$. Hence assume $J_3 \ne \emptyset$. Let $\overline{I}$ denote the instance we obtain by relaxing the integrality constraints on x_{ij} for all $j \in J_3$ and $i \in N$. It is clear that $L(0)$ is the value of the optimal solution to $\overline{I}$, which can be obtained as follows. $|J_2|$ bins are first initialized for the items in J_2. Then, for each item $j \in J_3$, let i^* denote the lowest indexed bin not completely filled (if no such bin, initialize a new one) and $c(i^*) \le c$ its residual capacity. If $w_j \le c(i^*)$ 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 $L(0) - 1$ items at most are split (no splitting can occur in the $L(0)$th bin). We can now obtain a feasible solution of value $\tilde{z} \ge z$ to I by removing the split items from the previous solution and assigning them to new bins. By the definition of J_3, at most $\lceil (L(0) - 1)/2 \rceil$ new bins are needed, so $\tilde{z} \le L(0) + \lfloor L(0)/2 \rfloor$, hence $\frac{3}{2} L(0) \ge z$.

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

$w_j = k+1$ $(k \geq 2)$ for $j = 1,\dots,n$ and $c = 3k$. We have $z = n/2$ and $L_2 = L(k+1) = \lceil n(k+1)/(3k) \rceil$, so ratio L_2/z can be arbitrarily close to $\frac{2}{3}$ for k sufficiently large. □

It is worthy of note that lower bounds with better worst-case performance can easily be obtained from approximate algorithms. We can use, for example, algorithm BFD of Section 8.2 to produce, for any instance I, a solution of value *BFD(I)*. This solution (see Johnson, Demers, Ullman, Garey and Graham (1974)) satisfies the same worst-case bound as *FFD(I)*, so we trivially obtain a lower bound (see (8.12))

$$LBFD(I) = \frac{9}{11}(BFD(I) - 4), \tag{8.21}$$

whose worst-case performance is smaller than that of L_2 for $z(I)$ sufficiently large, and asymptotically tends to $\frac{9}{11}$. Since however *BFD(I)* is known to be, in general, close to $z(I)$, the average performance of *LBFD* is quite poor (as will be seen in Section 8.6).

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 for a bin, say i^*, the values $x_{i^*j} = 1$ if $j \in F_1$ and $x_{i^*j} = 0$ if $j \notin F_1$, is no greater than the value that can be obtained by forcing the values $x_{i^*j} = 1$ if $j \in F_2$ and $x_{i^*j} = 0$ if $j \notin F_2$. 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_2 into subsets $P_1,\dots,P_l$ and a subset $\{j_1,\dots,j_l\}$ of F_1 exist such that $w_{j_h} \geq \sum_{k \in P_h} w_k$ for $h = 1,\dots,l$, then F_1 dominates F_2.*

Proof. Completing the solution through assignment of the items in $N \backslash F_1$ is easier than through assignment of the items in $N \backslash F_2$. In fact: (a) $\sum_{j \in N \backslash F_1} w_j \leq \sum_{j \in N \backslash F_2} w_j$; (b) for any feasible assignment of an item $j_h \in \{j_1,\dots,j_l\} \subseteq F_1$ there exists a feasible assignment of the items in $P_h \subseteq F_2$ (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 not 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^r gives the number of optimally filled bins, and, for each $j \in N$,

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

```
procedure MTRP:
input: $n, (w_j), c$;
output: $z^r, (b_j)$;
begin
    $N := \{1, \dots, n\}$;
    $\overline{N} := \emptyset$;
    $z^r := 0$;
    for $j := 1$ to $n$ do $b_j := 0$;
    repeat
        find $j = \min\{h : h \in N \setminus \overline{N}\}$;
        let $N' = N \setminus \{j\} = \{j_1, \dots, j_l\}$ with $w_{j_1} \geq \dots \geq w_{j_l}$;
        $F := \emptyset$;
        find the largest $k$ such that $w_j + \sum_{q=l-k+1}^{l} w_{j_q} \leq c$;
        if $k = 0$ then $F := \{j\}$
        else
            begin
                $j^* := \min\{h \in N' : w_j + w_h \leq c\}$;
                if $k = 1$ or $w_j + w_{j^*} = c$ then $F := \{j, j^*\}$
                else if $k = 2$ then
                    begin
                        find $j_a, j_b \in N'$, with $a < b$, such that
                          $w_{j_a} + w_{j_b} = \max\{w_{j_r} + w_{j_s}$ :
                          $j_r, j_s \in N', w_j + w_{j_r} + w_{j_s} \leq c\}$;
                        if $w_{j^*} \geq w_{j_a} + w_{j_b}$ then $F := \{j, j^*\}$
                        else if $w_{j^*} = w_{j_a}$ and $(b - a \leq 2$
                                  or $w_j + w_{j_b-1} + w_{j_b-2} > c)$
                              then $F := \{j, j_a, j_b\}$
                    end
            end;
        if $F = \emptyset$ then $\overline{N} := \overline{N} \cup \{j\}$
        else
            begin
                $z^r := z^r + 1$;
                for each $h \in F$ do $b_h = z^r$;
                $N := N \setminus F$
            end
    until $N \setminus \overline{N} = \emptyset$
end.
```

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 $w_{j^*} \geq w_{j_a} + w_{j_b}$ then set $\{j^*\}$ dominates all pairs of items (and, by definition of j^*, all singletons) which can be packed together with j, so $\{j, j^*\}$ dominates all feasible sets containing j; (b) if $w_{j^*} = w_{j_a}$ and either $b - a \leq 2$ or $w_j + w_{j_b - 1} + w_{j_b - 2} > c$ then set $\{j_a, j_b\}$ 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_a and j_b, 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_1^r denote the output value of z^r, and $I(z_1^r)$ the corresponding residual instance, defined by item set $\{j \in N : b_j = 0\}$. It is obvious that a lower bound for I is given by $z_1^r + L(I(z_1^r))$, where $L(I(z_1^r))$ denotes the value of any lower bound for $I(z_1^r)$. (Note that $z_1^r + L(I(z_1^r)) \geq L(I)$.) Suppose now that $I(z_1^r)$ is relaxed in some way (see below) and MTRP is applied to the relaxed instance, producing the output value z_2^r and a residual relaxed instance $I(z_1^r, z_2^r)$. A lower bound for I is then $z_1^r + z_2^r + L(I(z_1^r, z_2^r))$. Iterating the process we obtain a series of lower bounds of the form

$$L_3 = z_1^r + z_2^r + \ldots + L(I(z_1^r, z_2^r, \ldots)).$$

The following procedure computes the maximum of the above bounds, using L_2 for L. 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
    $L_3 := 0$;
    $z := 0$;
    $\overline{n} := n$;
    for $j := 1$ to $n$ do $\overline{w}_j := w_j$;
    while $\overline{n} \geq 1$ do
        begin
            call MTRP giving $\overline{n}$, $(\overline{w}_j)$ and $c$, yielding $z^r$ and $(\overline{b}_j)$;
            $z := z + z^r$;
            $k := 0$;
            for $j := 1$ to $\overline{n}$ do
                if $\overline{b}_j = 0$ then
                begin
                    $k := k + 1$;
                    $\overline{w}_k := \overline{w}_j$
                end;
        $\overline{n} := k$;
        if $\overline{n} = 0$ then $L_2 := 0$
        else call $L2$ giving $\overline{n}, (\overline{w}_j)$ and $c$, yielding $L_2$;
        $L_3 := \max(L_3, z + L_2)$;
        $\overline{n} := \overline{n} - 1$ (comment: removal of the smallest item)
    end
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

$$n = 14,$$

$$(w_j) = (99, 94, 79, 64, 50, 46, 43, 37, 32, 19, 18, 7, 6, 3),$$

$$c = 100.$$

The first execution of MTRP gives

$$j = 1 : k = 0, F = \{1\};$$

$$j = 2 : k = 1, j^* = 13, F = \{2, 13\},$$

and $F = \emptyset$ for $j \geq 3$. Hence

$$z = 2; (\overline{b}_j) = (1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0);$$

executing L2 for the residual instance we get $L_2 = 4$, so

$$L_3 = 6.$$

Item 14 is now removed and MTRP is applied to item set $\{3, 4, \ldots, 12\}$, producing (indices refer to the original instance)

$j = 3 : k = 1, j^* = 10, F = \{3, 10\};$

$j = 4 : k = 2, j^* = 9, j_a = 11, j_b = 12, F = \{4, 9\};$

$j = 5 : k = 2, j^* = 6, j_a = 7, j_b = 12, F = \emptyset;$

$j = 6 : k = 2, j^* = 5, j_a = 7, j_b = 12, F = \{6, 5\};$

$j = 7 : k = 2, j^* = 8, j_a = 8, j_b = 11, F = \{7, 8, 11\};$

$j = 12 : k = 0, F = \{12\};$

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

$$z = 7;\ (\overline{b}_j) = (1, 2, 3, 4, 5, 5, 6, 6, 4, 3, 6, 7, 2, -);$$

hence $L_2 = 0$ (since $\overline{n} = 0$) and the execution terminates with $L_3 = 7$.

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 $\overline{b}_{14} = 4$) is also optimal, since all items are assigned. □

8.5 EXACT ALGORITHMS

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

Eilon 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 $(\overline{c}_{i_1}, \dots, \overline{c}_{i_b})$ denote their current residual capacities sorted by increasing value, and $\overline{c}_{i_{b+1}} \equiv c_{b+1} = c$ the capacity of the next (not yet initialized) bin: the branching phase assigns the free item j^* of largest weight, in turn, to bins $i_s, \dots, i_b, i_{b+1}$, where $s = \min \{h : 1 \leq h \leq b + 1,\ \ \overline{c}_{i_h} + w_{j^*} \leq c\}$. Lower bound L_1 (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_1.

We do not give further details on these algorithms, since the computational results reported in Eilon 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 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 $\overline{c}_{i^*}$ is less than $w_{j^*} + w_n$, 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 $j \in \{k > j^* : w_k + w_n > \overline{c}_{i^*}\}$ is assigned to i^*; the bin is "re-opened" when the first item $j > j^*$ for which $w_j + w_n \leq \overline{c}_{i^*}$ is considered or, if no such item exists, when the first backtracking on an item $l < j^*$ is performed.

Since at any decision node the current residual capacities $\overline{c}_i$ 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 $c - \overline{c}_i$ 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

$$n = 10;$$

$$(w_j) = (49, 41, 34, 33, 29, 26, 26, 22, 20, 19);$$

$$c = 100.$$

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

$b_j =$ bin to which item j is assigned $(j = 1, \dots, n)$;

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

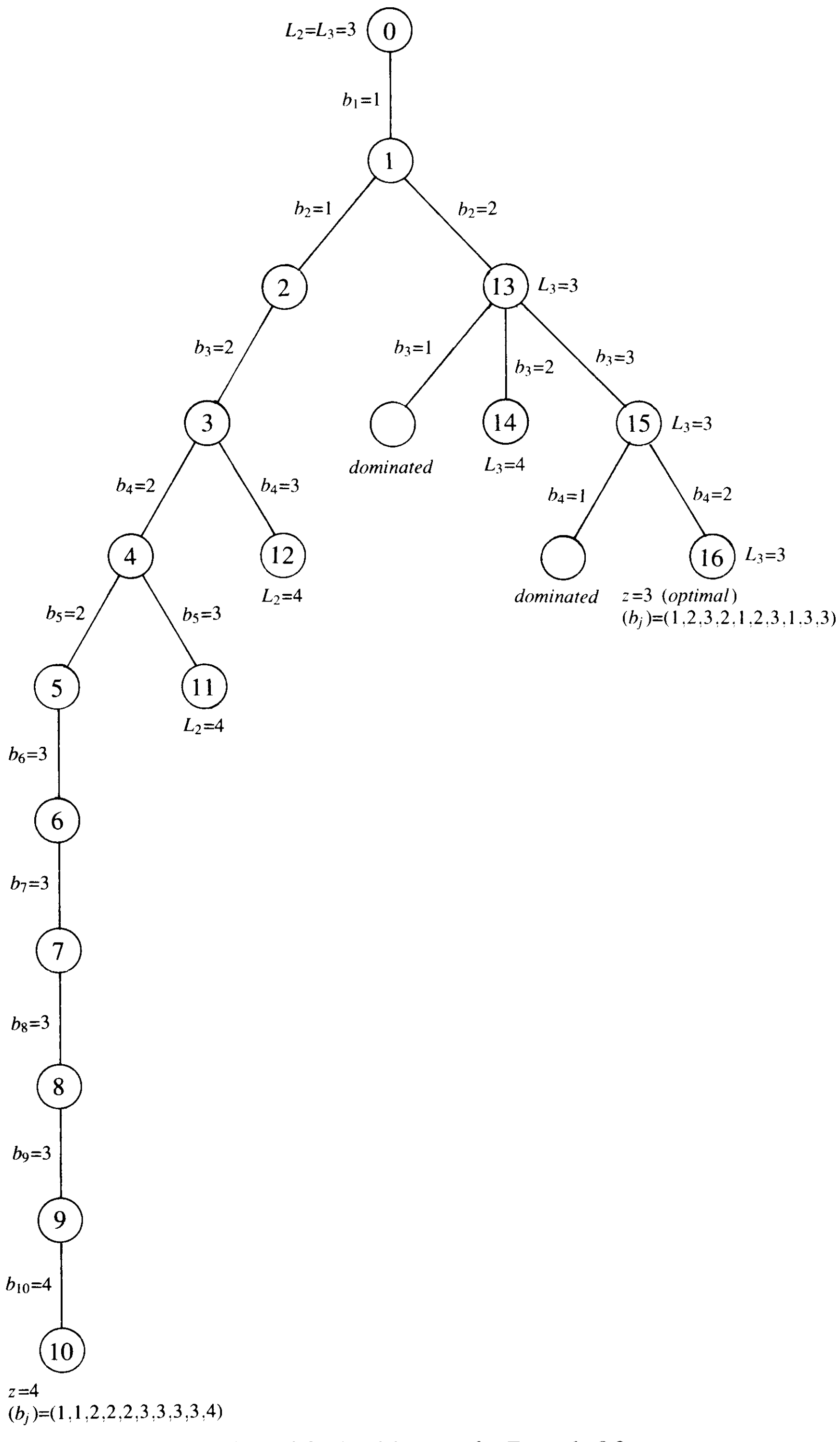

Figure 8.2 Decision-tree for Example 8.3

$z = 4,$

$(b_j) = (1, 1, 2, 2, 2, 3, 3, 3, 3, 4),$

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_3. At node 16, procedure MTRP (called by L3) is applied to problem

$\overline{n} = 9,$

$(\overline{w}_j) = (74, 49, 34, 29, 26, 26, 22, 20, 19),$

$\overline{c} = 100,$

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

$(w_j) = (-, -, -, -, 29, -, 26, 22, 20, 19),$

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

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

$(\overline{b}_j) = (-, -, -, -, 1, -, 3, 1, 3, 3),$

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 9000/840 (using option "-o" 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

Class	n	*LBFD*		L_1		L_2		L_3	
		Time	% err(opt)	Time	% err(opt)	Time	% err(opt)	Time	% err(opt)
	50	0.003	28.075(0)	0.001	6.413(2)	0.001	0.519(17)	0.001	0.000(20)
	100	0.009	24.530(0)	0.001	3.549(0)	0.001	0.877(11)	0.006	0.196(18)
1	200	0.020	20.934(0)	0.002	2.539(1)	0.001	0.291(14)	0.013	0.149(17)
	500	0.061	19.175(0)	0.002	2.487(0)	0.003	0.194(12)	0.033	0.078(16)
	1 000	0.136	18.452(0)	0.003	1.127(0)	0.007	0.159(7)	0.091	0.060(15)
	50	0.004	27.814(0)	0.001	7.938(0)	0.001	0.308(18)	0.002	0.000(20)
	100	0.011	23.641(0)	0.001	6.846(0)	0.002	0.306(16)	0.003	0.000(20)
2	200	0.022	20.192(0)	0.001	6.084(0)	0.001	0.352(13)	0.009	0.039(19)
	500	0.059	19.506(0)	0.001	5.558(0)	0.003	0.189(10)	0.019	0.032(18)
	1 000	0.139	18.441(0)	0.002	4.805(0)	0.005	0.168(5)	0.058	0.032(16)
	50	0.004	24.281(0)	0.001	23.300(0)	0.001	0.000(20)	0.001	0.000(20)
	100	0.010	21.612(0)	0.001	24.012(0)	0.001	0.000(20)	0.003	0.000(20)
3	200	0.023	19.483(0)	0.001	24.003(0)	0.001	0.000(20)	0.005	0.000(20)
	500	0.062	18.841(0)	0.001	24.044(0)	0.001	0.000(20)	0.016	0.000(20)
	1 000	0.146	18.153(0)	0.002	24.285(0)	0.001	0.000(20)	0.038	0.000(20)

Table 8.3 $C = 120$. HP 9000/840 in seconds. Average times / Average percentage errors (exact solution values found) over 20 problems

		LBFD		L_1		L_2		L_3	
Class	n	Time	% err(opt)	Time	% err(opt)	Time	% err(opt)	Time	% err(opt)
	50	0.004	29.832(0)	0.001	3.788(10)	0.001	0.000(20)	0.005	0.000(20)
	100	0.008	24.562(0)	0.001	1.990(10)	0.002	0.474(16)	0.014	0.357(17)
1	200	0.020	21.180(0)	0.001	0.732(13)	0.003	0.165(17)	0.042	0.000(20)
	500	0.055	19.731(0)	0.001	0.538(13)	0.006	0.116(15)	0.232	0.047(18)
	1 000	0.126	18.573(0)	0.002	0.105(17)	0.008	0.035(17)	0.983	0.024(18)
	50	0.003	28.514(0)	0.001	7.924(0)	0.001	0.976(15)	0.003	0.408(18)
	100	0.009	23.881(0)	0.001	3.714(2)	0.001	0.478(15)	0.005	0.091(19)
2	200	0.019	20.212(0)	0.001	3.326(0)	0.001	0.580(10)	0.009	0.050(19)
	500	0.060	19.304(0)	0.001	3.106(0)	0.004	0.216(11)	0.023	0.059(17)
	1 000	0.126	18.607(0)	0.003	1.386(0)	0.007	0.239(8)	0.062	0.080(13)
	50	0.004	25.208(0)	0.001	24.145(0)	0.001	0.000(20)	0.002	0.000(20)
	100	0.010	21.503(0)	0.001	22.344(0)	0.001	0.000(20)	0.004	0.000(20)
3	200	0.023	19.752(0)	0.001	22.155(0)	0.001	0.063(18)	0.007	0.000(20)
	500	0.064	18.614(0)	0.002	22.035(0)	0.002	0.012(19)	0.017	0.000(20)
	1 000	0.147	18.323(0)	0.003	21.775(0)	0.003	0.056(12)	0.036	0.000(20)

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

		LBFD		L_1		L_2		L_3	
Class	n	Time	% err(opt)	Time	% err(opt)	Time	% err(opt)	Time	% err(opt)
	50	0.003	31.832(0)	0.001	0.488(18)	0.001	0.250(19)	0.011	0.250(19)
	100	0.009	25.634(0)	0.001	0.000(20)	0.003	0.000(20)	0.039	0.000(20)
1	200	0.016	21.984(0)	0.001	0.000(20)	0.002	0.000(20)	0.137	0.000(20)
	500	0.056	20.007(0)	0.002	0.000(20)	0.004	0.000(20)	0.857	0.000(20)
	1 000	0.119	19.189(0)	0.003	0.000(20)	0.009	0.000(20)	4.011	0.000(20)
	50	0.004	31.408(0)	0.001	0.617(18)	0.001	0.217(19)	0.011	0.217(19)
	100	0.009	23.652(0)	0.001	0.727(14)	0.001	0.727(14)	0.043	0.727(14)
2	200	0.020	20.984(0)	0.001	0.978(0)	0.004	0.978(5)	0.169	0.978(5)
	500	0.059	19.837(0)	0.002	1.153(0)	0.004	1.153(0)	1.139	1.153(0)
	1 000	0.128	18.416(0)	0.003	1.224(0)	0.005	1.224(0)	9.806	1.224(0)
	50	0.003	27.941(0)	0.001	8.851(1)	0.001	0.579(17)	0.002	0.000(20)
	100	0.009	23.247(0)	0.001	4.302(4)	0.002	0.665(13)	0.003	0.000(20)
3	200	0.021	20.671(0)	0.001	3.856(0)	0.002	0.625(9)	0.007	0.000(20)
	500	0.061	19.459(0)	0.001	3.480(0)	0.003	0.234(11)	0.019	0.000(20)
	1 000	0.132	18.527(0)	0.003	1.687(0)	0.005	0.328(8)	0.036	0.000(20)

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 behaviour 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 $\frac{2}{11}$ when n grows. L_1 obviously requires very small times, practically independent of the data generation; the tightness improves when the ratio $c/\min_j\{w_j\}$ 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 get 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 8.5 Algorithm MTP. HP 9000/840 in seconds. Average times/Average numbers of nodes over 20 problems

		$c = 100$		$c = 120$		$c = 150$	
Class	n	Time	Nodes	Time	Nodes	Time	Nodes
	50	0.006	0	0.005	0	0.096	11
	100	0.012	1	15.022(17)	3561	0.156	29
1	200	5.391	1114	0.062	6	0.140	10
	500	10.236	2805	10.340	887	2.124	28
	1000	20.206(16)	2686	6.596	244	8.958	44
	50	0.005	0	0.008	1	0.183	61
	100	0.012	1	0.030	9	26.599(15)	4275
2	200	0.047	11	0.073	18	69.438(7)	8685
	500	0.127	28	10.062	1663	—	—
	1000	15.524(17)	3896	30.148(14)	4774	—	—
	50	0.005	0	0.005	0	0.005	0
	100	0.010	0	0.010	0	0.010	0
3	200	0.019	0	0.020	0	0.018	0
	500	0.049	0	0.050	0	0.051	0
	1000	0.102	0	0.104	0	0.105	0

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.