2

0-1 Knapsack problem

2.1 INTRODUCTION

The *0-1*, or *Binary*, *Knapsack Problem* (KP) is: given a set of n *items* and a *knapsack*, with

$$p_j = \textit{profit} \text{ of item } j,$$

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

$$c = \textit{capacity} \text{ of the knapsack},$$

select a subset of the items so as to

$$\text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c, \tag{2.2}$$

$$x_j = 0 \text{ or } 1, \qquad j \in N = \{1, \ldots, n\}, \tag{2.3}$$

where

$$x_j = \begin{cases} 1 & \text{if item } j \text{ is selected;} \\ 0 & \text{otherwise.} \end{cases}$$

KP is the most important knapsack problem and one of the most intensively studied discrete programming problems. The reason for such interest basically derives from three facts: (a) it can be viewed as the simplest *Integer Linear Programming* problem; (b) it appears as a subproblem in many more complex problems; (c) it may represent a great many practical situations. Recently, it has been used for generating minimal cover induced constraints (see, e.g., Crowder, Johnson and Padberg, (1983)) and in several coefficient reduction procedures for strengthening LP bounds in general integer programming (see, e.g., Dietrich and Escudero, (1989a, 1989b)). During the last few decades, KP has been studied through different approaches, according to the theoretical development of *Combinatorial Optimization.*

In the fifties, Bellman's *dynamic programming* theory produced the first algorithms to exactly solve the 0-1 knapsack problem. In 1957 Dantzig gave an elegant and efficient method to determine the solution to the continuous relaxation of the problem, and hence an *upper bound* on z which was used in the following twenty years in almost all studies on KP.

In the sixties, the dynamic programming approach to the KP and other knapsack-type problems was deeply investigated by Gilmore and Gomory. In 1967 Kolesar experimented with the first *branch-and-bound* algorithm for the problem.

In the seventies, the branch-and-bound approach was further developed, proving to be the only method capable of solving problems with a high number of variables. The most well-known algorithm of this period is due to Horowitz and Sahni. In 1973 Ingargiola and Korsh presented the first *reduction procedure*, a preprocessing algorithm which significantly reduces the number of variables. In 1974 Johnson gave the first *polynomial-time approximation scheme* for the subset-sum problem; the result was extended by Sahni to the 0-1 knapsack problem. The first *fully polynomial-time approximation scheme* was obtained by Ibarra and Kim in 1975. In 1977 Martello and Toth proposed the first upper bound dominating the value of the continuous relaxation.

The main results of the eighties concern the solution of large-size problems, for which sorting of the variables (required by all the most effective algorithms) takes a very high percentage of the running time. In 1980 Balas and Zemel presented a new approach to solve the problem by sorting, in many cases, only a small subset of the variables (the *core problem*).

In this chapter we describe the main results outlined above in logical (not necessarily chronological) sequence. Upper bounds are described in Sections 2.2 and 2.3, approximate algorithms in Sections 2.4 and 2.8, exact algorithms in Sections 2.5, 2.6 and 2.9, reduction procedures in Section 2.7. Computational experiments are reported in Section 2.10, while Section 2.11 contains an introduction to the facetial analysis of the problem. Section 2.12 deals with a generalization of KP (the multiple-choice knapsack problem).

We will assume, without any loss of generality, that

$$p_j, w_j \text{ and } c \text{ are positive integers,} \tag{2.4}$$

$$\sum_{j=1}^{n} w_j > c, \tag{2.5}$$

$$w_j \leq c \text{ for } j \in N. \tag{2.6}$$

If assumption (2.4) is violated, fractions can be handled by multiplying through by a proper factor, while nonpositive values can be handled as follows (Glover, 1965):

1. for each $j \in N^0 = \{j \in N : p_j \leq 0, w_j \geq 0\}$ **do** $x_j := 0$;
2. for each $j \in N^1 = \{j \in N : p_j \geq 0, w_j \leq 0\}$ **do** $x_j := 1$;

3. let $N^- = \{j \in N : p_j < 0, w_j < 0\}$, $N^+ = N \setminus (N^0 \cup N^1 \cup N^-)$, and

$$\begin{cases} y_j = 1 - x_j, \bar{p}_j = -p_j, \bar{w}_j = -w_j & \text{for } j \in N^-, \\ y_j = x_j, \bar{p}_j = p_j, \bar{w}_j = w_j & \text{for } j \in N^+; \end{cases}$$

4. solve the residual problem

$$\text{maximize} \quad z = \sum_{j \in N^- \cup N^+} \bar{p}_j y_j + \sum_{j \in N^1 \cup N^-} p_j$$

$$\text{subject to} \quad \sum_{j \in N^- \cup N^+} \bar{w}_j y_j \leq c - \sum_{j \in N^1 \cup N^-} w_j,$$

$$y_j = 0 \text{ or } 1, \qquad j \in N^- \cup N^+.$$

If the input data violate assumption (2.5) then, trivially, $x_j = 1$ for all $j \in N$; if they violate assumption (2.6), then $x_j = 0$ for each j such that $w_j > c$.

Unless otherwise specified, we will always suppose that the items are ordered according to decreasing values of the profit per unit weight, i.e. so that

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}. \tag{2.7}$$

If this is not the case, profits and weights can be re-indexed in $O(n \log n)$ time through any efficient sorting procedure (see, for instance, Aho, Hopcroft and Ullman, (1983)).

Given any problem instance I, we denote the value of any optimal solution with $z(I)$, or, when no confusion arises, with z.

KP is always considered here in maximization form. The minimization version of the problem,

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

$$\text{subject to} \quad \sum_{j=1}^{n} w_j y_j \geq q,$$

$$y_j = 0 \text{ or } 1, \qquad j \in N$$

can easily be transformed into an equivalent maximization form by setting $y_j = 1 - x_j$ and solving (2.1), (2.2), (2.3) with $c = \sum_{j=1}^{n} w_j - q$. Let $z\max$ be the solution value of such a problem: the minimization problem then has solution value $z\min = \sum_{j=1}^{n} p_j - z\max$. (Intuitively, we maximize the total profit of the items *not inserted* in the knapsack.)

2.2 RELAXATIONS AND UPPER BOUNDS

2.2.1 Linear programming relaxation and Dantzig's bound

The most natural, and historically the first, relaxation of KP is the *linear programming relaxation*, i.e. the *continuous knapsack problem* $C(KP)$ obtained from (2.1), (2.2), (2.3) by removing the integrality constraint on x_j:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c,$$

$$0 \leq x_j \leq 1, \qquad j = 1, \ldots, n.$$

Suppose that the items, ordered according to (2.7), are consecutively inserted into the knapsack until the first item, s, is found which does not fit. We call it the *critical item*, i.e.

$$s = \min \left\{ j : \sum_{i=1}^{j} w_i > c \right\}, \tag{2.8}$$

and note that, because of assumptions (2.4)–(2.6), we have $1 < s \leq n$. Then $C(KP)$ can be solved through a property established by Dantzig (1957), which can be formally stated as follows.

Theorem 2.1 *The optimal solution* $\bar{x}$ *of* $C(KP)$ *is*

$$\bar{x}_j = 1 \qquad \text{for } j = 1, \ldots, s-1,$$

$$\bar{x}_j = 0 \qquad \text{for } j = s+1, \ldots, n,$$

$$\bar{x}_s = \frac{\bar{c}}{w_s},$$

where

$$\bar{c} = c - \sum_{j=1}^{s-1} w_j. \tag{2.9}$$

Proof. A graphical proof can be found in Dantzig (1957). More formally, observe that any optimal solution x of $C(KP)$ must be maximal, in the sense that $\sum_{j=1}^{n} w_j x_j = c$. Assume, without loss of generality, that $p_j/w_j > p_{j+1}/w_{j+1}$ for all j, and let x^* be the optimal solution of $C(KP)$. Suppose, by absurdity, that $x_k^* < 1$ for some $k < s$, then we must have $x_q^* > \bar{x}_q$ for at least one item $q \geq s$.

Given a sufficiently small $\varepsilon > 0$, we could increase the value of x_k^* by ε and decrease that of x_q^* by $\varepsilon w_k / w_q$, thus augmenting the value of the objective function of $\varepsilon(p_k - p_q w_k / w_q)$ (> 0, since $p_k / w_k > p_q / w_q$), which is a contradiction. In the same way we can prove that $x_k^* > 0$ for $k > s$ is impossible. Hence $\bar{x}_s = \bar{c} / w_s$ from maximality. □

The optimal solution value of $C(KP)$ follows:

$$z(C(KP)) = \sum_{j=1}^{s-1} p_j + \bar{c} \frac{p_s}{w_s}.$$

Because of the integrality of p_j and x_j, a valid upper bound on $z(KP)$ is thus

$$U_1 = \lfloor z(C(KP)) \rfloor = \sum_{j=1}^{s-1} p_j + \left\lfloor \bar{c} \frac{p_s}{w_s} \right\rfloor, \tag{2.10}$$

where $\lfloor a \rfloor$ denotes the largest integer not greater than a.

The worst-case performance ratio of U_1 is $\rho(U_1) = 2$. This can easily be proved by observing that, from (2.10), $U_1 < \sum_{j=1}^{s-1} p_j + p_s$. Both $\sum_{j=1}^{s-1} p_j$ and p_s are feasible solution values for KP, hence no greater than the optimal solution value z, thus, for any instance, $U_1 / z < 2$. To see that $\rho(U_1)$ is tight, consider the series of problems with $n = 2$. $p_1 = w_1 = p_2 = w_2 = k$ and $c = 2k - 1$, for which $U_1 = 2k - 1$ and $z = k$, so U_1 / z can be arbitrarily close to 2 for k sufficiently large.

The computation of $z(C(KP))$, hence that of the Dantzig bound U_1, clearly requires $O(n)$ time if the items are already sorted as assumed. If this is not the case, the computation can still be performed in $O(n)$ time by using the following procedure to determine the critical item.

2.2.2 Finding the critical item in $O(n)$ time

For each $j \in N$, define $r_j = p_j / w_j$. The *critical ratio* r_s can then be identified by determining a partition of N into $J1 \cup JC \cup J0$ such that

$$r_j > r_s \qquad \text{for } j \in J1,$$

$$r_j = r_s \qquad \text{for } j \in JC,$$

$$r_j < r_s \qquad \text{for } j \in J0,$$

and

$$\sum_{j \in J1} w_j \leq c < \sum_{j \in J1 \cup JC} w_j.$$

The procedure, proposed by Balas and Zemel (1980), progressively determines $J1$

and $J0$ using, at each iteration, a tentative value λ to partition the set of currently "free" items $N \setminus (J1 \cup J0)$. Once the final partition is known, the critical item s is identified by filling the residual capacity $c - \sum_{j \in J1} w_j$ with items in JC, in any order.

procedure CRITICAL_ITEM:
input: $n, c, (p_j), (w_j)$;
output: s;
begin
$J1 := \emptyset$;
$J0 := \emptyset$;
$JC := \{1, \ldots, n\}$;
$\overline{c} := c$;
partition := "no";
while *partition* = "no" **do**
begin
determine the median λ of the values in $R = \{p_j/w_j : j \in JC\}$;
$G := \{j \in JC : p_j/w_j > \lambda\}$;
$L := \{j \in JC : p_j/w_j < \lambda\}$;
$E := \{j \in JC : p_j/w_j = \lambda\}$;
$c' := \sum_{j \in G} w_j$;
$c'' := c' + \sum_{j \in E} w_j$;
if $c' \leq \overline{c} < c''$ **then** *partition* := "yes"
else if $c' > \overline{c}$ **then** (**comment**: λ is too small)
begin
$J0 := J0 \cup L \cup E$;
$JC := G$
end
else (**comment**: λ is too large)
begin
$J1 := J1 \cup G \cup E$;
$JC := L$;
$\overline{c} := \overline{c} - c''$
end
end;
$J1 := J1 \cup G$;
$J0 := J0 \cup L$;
$JC := E\ (= \{e_1, e_2, \ldots, e_q\})$;
$\overline{c} := \overline{c} - c'$;
$\sigma := \min \{j : \sum_{i=1}^{j} w_{e_i} > \overline{c}\}$;
$s := e_\sigma$
end.

Finding the median of m elements requires $O(m)$ time (see Aho, Hopcroft and Ullman, (1983)), so each iteration of the "while" loop requires $O(|JC|)$ time. Since at least half the elements of JC are eliminated at each iteration, the overall time complexity of the procedure is $O(n)$.

The solution of $C(KP)$ can then be determined as

$$\bar{x}_j = 1 \qquad \text{for } j \in J1 \cup \{e_1, e_2, \ldots, e_{\sigma-1}\};$$

$$\bar{x}_j = 0 \qquad \text{for } j \in J0 \cup \{e_{\sigma+1}, \ldots, e_q\};$$

$$\bar{x}_s = \left(c - \sum_{j \in N \setminus \{s\}} w_j \bar{x}_j \right) / w_s .$$

2.2.3 Lagrangian relaxation

An alternative way to relax KP is through the Lagrangian approach. Given a non-negative multiplier λ, the *Lagrangian relaxation* of KP ($L(KP, \lambda)$) is

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j + \lambda \left(c - \sum_{j=1}^{n} w_j x_j \right)$$

$$\text{subject to} \quad x_j = 0 \text{ or } 1, \qquad j = 1, \ldots, n.$$

The objective function can be restated as

$$z(L(KP, \lambda)) = \sum_{j=1}^{n} \tilde{p}_j x_j + \lambda c, \tag{2.11}$$

where $\tilde{p}_j = p_j - \lambda w_j$ for $j = 1, \ldots, n$, and the optimal solution of $L(KP, \lambda)$ is easily determined, in $O(n)$ time, as

$$\tilde{x}_j = \begin{cases} 1 & \text{if } \tilde{p}_j > 0, \\ 0 & \text{if } \tilde{p}_j < 0. \end{cases} \tag{2.12}$$

(When $\tilde{p}_j = 0$, the value of $\tilde{x}_j$ is immaterial.) Hence, by defining $J(\lambda) = \{j : p_j / w_j > \lambda\}$, the solution value of $L(KP, \lambda)$ is

$$z(L(KP, \lambda)) = \sum_{j \in J(\lambda)} \tilde{p}_j + \lambda c.$$

For any $\lambda \geq 0$, this is an upper bound on $z(KP)$ which, however, can never be better than the Dantzig bound U_1. In fact (2.12) also gives the solution of the continuous relaxation of $L(KP, \lambda)$, so

$$z(L(KP, \lambda)) = z(C(L(KP, \lambda))) \geq z(C(KP)).$$

The value of λ producing the minimum value of $z(L(KP, \lambda))$ is $\lambda^* = p_s/w_s$. With this value, in fact, we have $\tilde{p}_j \geq 0$ for $j = 1, \ldots, s-1$ and $\tilde{p}_j \leq 0$ for $j = s, \ldots, n$, so $J(\lambda^*) \subseteq \{1, \ldots, s-1\}$. Hence $\tilde{x}_j = \bar{x}_j$ for $j \in N \setminus \{s\}$ (where $(\bar{x}_j)$ is defined by Theorem 2.1) and, from (2.11)–(2.12), $z(L(KP, \lambda^*)) = \sum_{j=1}^{s-1}(p_j - \lambda^* w_j) + \lambda^* c = z(C(KP))$. Also notice that, for $\lambda = \lambda^*$, $\tilde{p}_j$ becomes

$$p_j^* = p_j - w_j \frac{p_s}{w_s}; \tag{2.13}$$

$| p_j^* |$ is the decrease which we obtain in $z(L(KP, \lambda^*))$ by setting $\tilde{x}_j = 1 - \tilde{x}_j$, and hence a *lower bound* on the corresponding decrease in the continuous solution value (since the optimal λ generally changes by imposing the above conditions). The value of $| p_j^* |$ will be very useful in the next sections.

Other properties of the Lagrangian relaxation for KP have been investigated by Maculan (1983). See also Fisher (1981) for a general survey on Lagrangian relaxations.

2.3 IMPROVED BOUNDS

In the present section we consider upper bounds dominating the Dantzig one, useful to improve on the average efficiency of algorithms for KP. Because of this dominance property, the worst-case performance ratio of these bounds is at most 2. Indeed, it is exactly 2, as can easily be verified through series of examples similar to that introduced for U_1, i.e. having $p_j = w_j$ for all j (so that the bounds take the trivial value c).

2.3.1 Bounds from additional constraints

Martello and Toth obtained the first upper bound dominating the Dantzig one, by imposing the integrality of the *critical variable* x_s.

Theorem 2.2 (Martello and Toth, 1977a) *Let*

$$U^0 = \sum_{j=1}^{s-1} p_j + \left\lfloor \bar{c} \frac{p_{s+1}}{w_{s+1}} \right\rfloor, \tag{2.14}$$

$$U^1 = \sum_{j=1}^{s-1} p_j + \left\lfloor p_s - (w_s - \bar{c}) \frac{p_{s-1}}{w_{s-1}} \right\rfloor, \tag{2.15}$$

where s and $\bar{c}$ are the values defined by (2.8) and (2.9). Then

(i) an upper bound on $z(KP)$ is

$$U_2 = \max(U^0, U^1); \tag{2.16}$$

(ii) for any instance of KP, we have $U_2 \leq U_1$.

Proof. (i) Since x_s cannot take a fractional value, the optimal solution of KP can be obtained from the continuous solution $\bar{x}$ of $C(KP)$ either without inserting item s (i.e. by imposing $\bar{x}_s = 0$), or by inserting it (i.e. by imposing $\bar{x}_s = 1$) and hence removing at least one of the first $s-1$ items. In the former case, the solution value cannot exceed U^0, which corresponds to the case of filling the residual capacity $\bar{c}$ with items having the best possible value of p_j/w_j (i.e. p_{s+1}/w_{s+1}). In the latter it cannot exceed U^1, where it is supposed that the item to be removed has exactly the minimum necessary value of w_j (i.e. $w_s - \bar{c}$) and the worst possible value of p_j/w_j (i.e. p_{s-1}/w_{s-1}).

(ii) $U^0 \leq U_1$ directly follows from (2.10), (2.14) and (2.7). To prove that $U^1 \leq U_1$ also holds, notice that $p_s/w_s \leq p_{s-1}/w_{s-1}$ (from (2.7)), and $\bar{c} < w_s$ (from (2.8), (2.9)). Hence

$$(\bar{c} - w_s)\left(\frac{p_s}{w_s} - \frac{p_{s-1}}{w_{s-1}}\right) \geq 0,$$

and, by algebraic manipulation,

$$\bar{c}\frac{p_s}{w_s} \geq p_s - (w_s - \bar{c})\frac{p_{s-1}}{w_{s-1}},$$

from which one has the thesis. □

The time complexity for the computation of U_2 is trivially $O(n)$, once the critical item is known.

Example 2.1

Consider the instance of KP defined by

$$\begin{aligned} n &= 8, \\ (p_j) &= (15, 100, 90, 60, 40, 15, 10, 1), \\ (w_j) &= (2, 20, 20, 30, 40, 30, 60, 10), \\ c &= 102. \end{aligned}$$

The optimal solution is $x = (1, 1, 1, 1, 0, 1, 0, 0)$, of value $z = 280$. From (2.8) we have $s = 5$. Hence

$$U_1 = 265 + \left\lfloor 30 \frac{40}{40} \right\rfloor = 295.$$

$$U^0 = 265 + \left\lfloor 30 \frac{15}{30} \right\rfloor = 280;$$

$$U^1 = 265 + \left\lfloor 40 - 10 \frac{60}{30} \right\rfloor = 285;$$

$$U_2 = 285. \square$$

The consideration on which the Martello and Toth bound is based can be further exploited to compute more restrictive upper bounds than U_2. This can be achieved by replacing the values U^0 and U^1 with tighter values, say $\overline{U}^0$ and $\overline{U}^1$, which take the exclusion and inclusion of item s more carefully into account. Hudson (1977) has proposed computing $\overline{U}^1$ as the solution value of the continuous relaxation of KP with the additional constraint $x_s = 1$. Fayard and Plateau (1982) and, independently, Villela and Bornstein (1983), proposed computing $\overline{U}^0$ as the solution value of $C(KP)$ with the additional constraint $x_s = 0$.

By defining $\sigma^1(j)$ and $\sigma^0(j)$ as the critical item when we impose, respectively, $x_j = 1$ $(j \geq s)$ and $x_j = 0$ $(j \leq s)$, that is

$$\sigma^1(j) = \min \left\{ k : \sum_{i=1}^{k} w_i > c - w_j \right\}, \tag{2.17}$$

$$\sigma^0(j) = \min \left\{ k : \sum_{\substack{i=1 \\ i \neq j}}^{k} w_i > c \right\}, \tag{2.18}$$

we obtain

$$\overline{U}^0 = \sum_{\substack{j=1 \\ j \neq s}}^{\sigma^0(s)-1} p_j + \left\lfloor \left(c - \sum_{\substack{j=1 \\ j \neq s}}^{\sigma^0(s)-1} w_j \right) \frac{p_{\sigma^0(s)}}{w_{\sigma^0(s)}} \right\rfloor, \tag{2.19}$$

$$\overline{U}^1 = p_s + \sum_{j=1}^{\sigma^1(s)-1} p_j + \left\lfloor \left(c - w_s - \sum_{j=1}^{\sigma^1(s)-1} w_j \right) \frac{p_{\sigma^1(s)}}{w_{\sigma^1(s)}} \right\rfloor, \tag{2.20}$$

and the new upper bound

$$U_3 = \max \left(\overline{U}^0, \overline{U}^1 \right).$$

It is self-evident that:

(a) $\overline{U}^0 \leq U^0$ and $\overline{U}^1 \leq U^1$, so $U_3 \leq U_2$;

(b) the time complexity for the computation of U_3 is the same as for U_1 and U_2, i.e. $O(n)$.

Example 2.1 (continued)

From (2.17)–(2.20) we have

$$\sigma^0(5) = 7, \quad \overline{U}^0 = 280 + \left\lfloor 0 \frac{10}{60} \right\rfloor = 280;$$

$$\sigma^1(5) = 4, \quad \overline{U}^1 = 40 + 205 + \left\lfloor 20 \frac{60}{30} \right\rfloor = 285;$$

$$U_3 = 285. \square$$

2.3.2 Bounds from Lagrangian relaxations

Other bounds computable in $O(n)$ time can be described through the terminology introduced in Section 2.2 for the Lagrangian relaxation of the problem. Remember that $z(C(KP)) = z(L(KP, \lambda^*))$ and $| p_j^* |$ (see 2.13) is a lower bound on the decrease of $z(C(KP))$ corresponding to the change of the jth variable from $\tilde{x}_j$ to $1 - \tilde{x}_j$. Müller-Merbach (1978) noted that, in order to obtain an integer solution from the continuous one, either (a) the fractional variable $\overline{x}_s$ alone has to be reduced to 0 (without any changes of the other variables), or (b) at least one of the other variables, say $\overline{x}_j$, has to change its value (from 1 to 0 or from 0 to 1). In case (a) the value of $z(C(KP))$ decreases by $\overline{c}p_s/w_s$, in case (b) by at least $| p_j^* |$. Hence the Müller-Merbach bound

$$U_4 = \max \left(\sum_{j=1}^{s-1} p_j, \max \{ \lfloor z(C(KP)) - | p_j^* | \rfloor : j \in N \setminus \{s\} \} \right). \tag{2.21}$$

It is immediately evident that $U_4 \leq U_1$. No dominance exists, instead, between U_4 and the other bounds. For the instance of example 2.1 we have $U_3 = U_2 < U_4$ (see below), but it is not difficult to find examples (see Müller-Merbach (1978)) for which $U_4 < U_3 \leq U_2$.

The ideas behind bounds U_2, U_3 and U_4 have been further exploited by Dudzinski and Walukiewicz (1984a), who have obtained an upper bound dominating all the above. Consider any feasible solution $\hat{x}$ to KP that we can obtain from the continuous one as follows:

1. **for each** $k \in N \setminus \{s\}$ **do** $\hat{x}_k := \overline{x}_k$;
2. $\hat{x}_s := 0$;

3. **for each** k such that $\hat{x}_k = 0$ **do**
if $w_k \leq c - \sum_{j=1}^{n} w_j \hat{x}_j$ **then** $\hat{x}_k := 1$,

and define $\hat{N} = \{j \in N \backslash \{s\} : \hat{x}_j = 0\}$ ($\hat{x}$ is closely related to the *greedy solution*, discussed in Section 2.4). Noting that an optimal integer solution can be obtained (a) by setting $x_s = 1$ or (b) by setting $x_s = 0$ and $x_j = 1$ for at least one $j \in \hat{N}$, it is immediate to obtain the Dudzinski and Walukiewicz (1984a) bound:

$$U_5 = \max \left(\min \left(\overline{U}^1, \max \{\lfloor z(C(KP)) - p_j^* \rfloor : j = 1, \ldots, s-1\}\right),\right.$$
$$\min \left(\overline{U}^0, \max \{\lfloor z(C(KP)) + p_j^* \rfloor : j \in \hat{N}\}\right),$$
$$\left.\sum_{j=1}^{n} p_j \hat{x}_j\right), \tag{2.22}$$

where $\overline{U}^0$ and $\overline{U}^1$ are given by (2.19) and (2.20). The time complexity is $O(n)$.

Example 2.1 (continued)

From (2.13), $(p_j^*) = (13, 80, 70, 30, 0, -15, -50, -9)$. Hence:

$U_4 = \max (265, \max \{282, 215, 225, 265, 280, 245, 286\}) = 286.$

$(\hat{x}_j) = (1, 1, 1, 1, 0, 1, 0, 0);$

$U_5 = \max (\min (285, \max \{282, 215, 225, 265\}),$

$\min (280, \max \{245, 286\}), 280) = 282.$ □

2.3.3 Bounds from partial enumeration

Bound U_3 of Section 2.3.1 can also be seen as the result of the application of the Dantzig bound at the two terminal nodes of a decision tree having the root node corresponding to KP and two descendent nodes, say N0 and N1, corresponding to the exclusion and inclusion of item s. Clearly, the maximum among the upper bounds corresponding to all the terminal nodes of a decision tree represents a valid upper bound for the original problem corresponding to the root node. So, if $\overline{U}^0$ and $\overline{U}^1$ are the Dantzig bounds corresponding respectively to nodes N0 and N1, U_3 represents a valid upper bound for KP.

An improved bound, U_6, can be obtained by considering decision trees having more than two terminal nodes, as proposed by Martello and Toth (1988).

In order to introduce this bound, suppose s has been determined, and let r, t be any two items such that $1 < r \leq s$ and $s \leq t < n$. We can obtain a feasible

solution for KP by setting $x_j = 1$ for $j < r$, $x_j = 0$ for $j > t$ and finding the optimal solution of subproblem KP(r, t) defined by items $r, r+1, \ldots, t$ with reduced capacity $c(r) = c - \sum_{j=1}^{r-1} w_j$. Suppose now that KP$(r, t)$ is solved through an elementary binary decision-tree which, for $j = r, r+1, \ldots, t$, generates pairs of decision nodes by setting, respectively, $x_j = 1$ and $x_j = 0$; each node k (obtained, say, by fixing x_j) generates a pair of descendent nodes (by fixing x_{j+1}) iff $j < t$ and the solution corresponding to k is feasible. For each node k of the resulting tree, let $f(k)$ be the item from which k has been generated (by setting $x_{f(k)} = 1$ or 0) and denote with x_j^k $(j = r, \ldots, f(k))$ the sequence of values assigned to variables $x_r, \ldots, x_{f(k)}$ along the path in the tree from the root to k. The set of terminal nodes (*leaves*) of the tree can then be partitioned into

$$L_1 = \left\{ l : \sum_{j=r}^{f(l)} w_j x_j^l > c(r) \right\} \qquad \text{(infeasible leaves)},$$

$$L_2 = \left\{ l : f(l) = t \text{ and } \sum_{j=r}^{f(l)} w_j x_j^l \leq c(r) \right\} \qquad \text{(feasible leaves)}.$$

For each $l \in L_1 \cup L_2$, let u_l be any upper bound on the problem defined by (2.1), (2.2) and

$$\begin{cases} x_j = x_j^l & \text{if } j \in \{r, \ldots, f(l)\}, \\ x_j = 0 \text{ or } 1 & \text{if } j \in N \setminus \{r, \ldots, f(l)\}. \end{cases} \tag{2.23}$$

Since all the nonleaf nodes are completely explored by the tree, a valid upper bound for KP is given by

$$U_6 = \max \{u_l : l \in L_1 \cup L_2\}. \tag{2.24}$$

A fast way to compute u_l is the following. Let $p^l = \sum_{j=1}^{r-1} p_j + \sum_{j=r}^{f(l)} p_j x_j^l$, and $d^l = | c(r) - \sum_{j=r}^{f(l)} w_j x_j^l |$; then

$$u_l = \begin{cases} \left\lfloor p^l - d^l \dfrac{p_{r-1}}{w_{r-1}} \right\rfloor & \text{if } l \in L_1; \\ \left\lfloor p^l + d^l \dfrac{p_{t+1}}{w_{t+1}} \right\rfloor & \text{if } l \in L_2, \end{cases} \tag{2.25}$$

which is clearly an upper bound on the continuous solution value for problem (2.1), (2.2), (2.23).

The computation of U_6 requires $O(n)$ time to determine the critical item and define KP(r, t), plus $O(2^{t-r})$ time to explore the binary tree. If $t - r$ is bounded by a constant, the overall time complexity is thus $O(n)$.

Example 2.1 (continued)

Assume $r = 4$ and $t = 6$. The reduced capacity is $c(r) = 60$. The binary tree is given in Figure 2.1. The leaf sets are $L_1 = \{2, 8\}$, $L_2 = \{4, 5, 9, 11, 12\}$. It follows that $U_6 = 280$, which is the optimal solution value. □

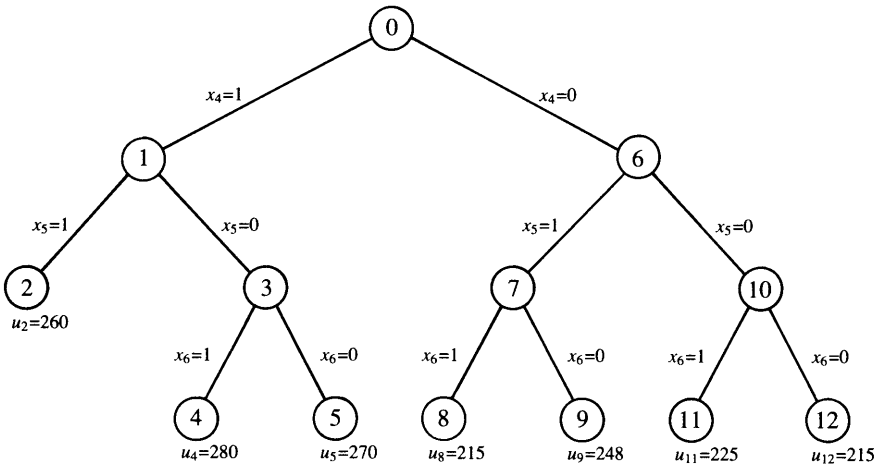

Figure 2.1 Binary tree for upper bound U_6 of Example 2.1

The upper bounds at the leaves can also be evaluated, of course, using any of the bounds previously described, instead of (2.25). If U_k $(k = 1, \ldots, 5)$ is used, then clearly $U_6 \leq U_k$; if (2.25) is used, then no dominance exists between U_6 and the Dudzinsky and Walukiewicz (1984a) bound, so the best upper bound for KP is

$$U = \min (U_5, U_6).$$

U_6 can be strengthened, with very small extra computational effort, by evaluating $w_m = \min \{w_j : j > t\}$. It is not difficult to see that, when $l \in L_2$ and $d^l < w_m$, u_l can be computed as

$$u_l = \max \left(p^l, \left\lfloor p^l + w_m \frac{p_{t+1}}{w_{t+1}} - (w_m - d^l) \frac{p_{r-1}}{w_{r-1}} \right\rfloor \right). \tag{2.26}$$

Finally, we note that the computation of U_6 can be considerably accelerated by using an appropriate branch-and-bound algorithm to solve KP(r, t). At any iteration of such algorithm, let $\bar{z}(r, t)$ be the value of the best solution so far. For any nonleaf node k of the decision-tree, let $\bar{u}_k$ be an upper bound on the optimal solution of the subproblem defined by items $r, \ldots, n$ with reduced capacity $c(r)$, i.e., the subproblem obtained by setting $x_j = 1$ for $j = 1, \ldots, r-1$. $\bar{u}_k$ can be computed as an upper bound of the continuous solution value of the problem, i.e.

$$\overline{u}_k = \sum_{j=r}^{f(k)} p_j x_j^k + \sum_{j=f(k)+1}^{s(k)-1} p_j + \left\lfloor \left(c(r) - \left(\sum_{j=r}^{f(k)} w_j x_j^k + \sum_{j=f(k)+1}^{s(k)-1} w_j \right) \right) \frac{p_{s(k)}}{w_{s(k)}} \right\rfloor, \tag{2.27}$$

where $s(k) = \min(t+1, \min\{i : \sum_{j=r}^{f(k)} w_j x_j^k + \sum_{j=f(k)+1}^{i} w_j > c(r)\})$. If we have $\overline{u}_k \leq \overline{z}(r,t)$, the nodes descending from k need not be generated. In fact, for any leaf l descending from k, it would result that $u_l \leq \sum_{j=1}^{r-1} p_j + \overline{u}_k \leq \sum_{j=1}^{r-1} p_j + z(KP(r,t)) \leq U_6$.

Example 2.1 (continued)

Accelerating the computation through (2.27), we obtain the reduced branch-decision tree of Figure 2.2. □

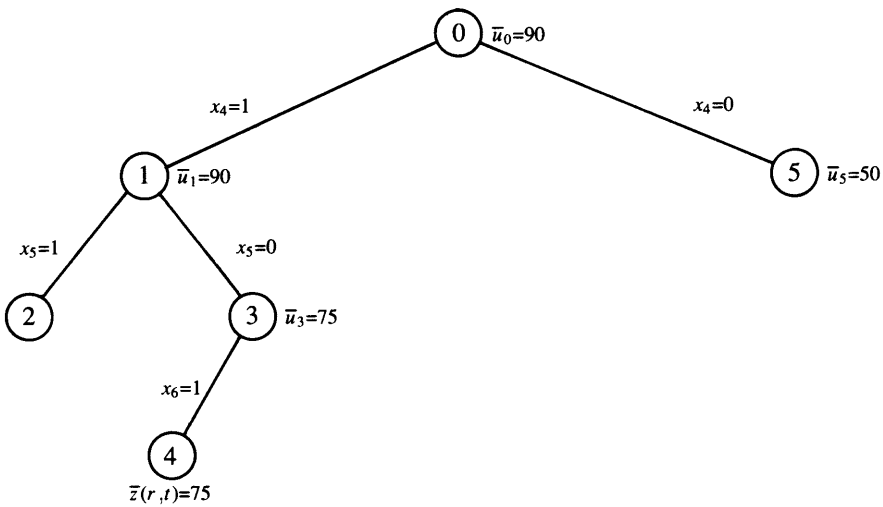

Figure 2.2 Branch-and-bound tree for upper bound U_6 of Example 2.1

2.4 THE GREEDY ALGORITHM

The most immediate way to determine an approximate solution to KP exploits the fact that solution $\overline{x}$ of the continuous relaxation of the problem has only one fractional variable, $\overline{x}_s$ (see Theorem 2.1). Setting this variable to 0 gives a feasible

solution to KP of value

$$z' = \sum_{j=1}^{s-1} p_j .$$

We can expect that z' is, on average, quite close to the optimal solution value z. In fact $z' \leq z \leq U_1 \leq z' + p_s$, i.e. the absolute error is bounded by p_s. The worst-case performance ratio, however, is arbitrarily bad. This is shown by the series of problems with $n = 2$, $p_1 = w_1 = 1$, $p_2 = w_2 = k$ and $c = k$, for which $z' = 1$ and $z = k$, so the ratio z'/z is arbitrarily close to 0 for k sufficiently large.

Noting that the above pathology occurs when p_s is relatively large, we can obtain an improved heuristic by also considering the feasible solution given by the critical item alone and taking the best of the two solution values, i.e.

$$z^h = \max (z', p_s). \tag{2.28}$$

The worst-case performance ratio of the new heuristic is $\frac{1}{2}$. We have already noted, in fact, that $z \leq z' + p_s$, so, from (2.28), $z \leq 2z^h$. To see that $\frac{1}{2}$ is tight, consider the series of problems with $n = 3$, $p_1 = w_1 = 1$, $p_2 = w_2 = p_3 = w_3 = k$ and $c = 2k$: we have $z^h = k + 1$ and $z = 2k$, so z^h/z is arbitrarily close to $\frac{1}{2}$ for k sufficiently large.

The computation of z^h requires $O(n)$ time, once the critical item is known. If the items are sorted as in (2.7), a more effective algorithm is to consider them according to increasing indices and insert each new item into the knapsack if it fits. (Notice that items $1, \ldots, s-1$ are always inserted, so the solution value is at least z'.) This is the most popular heuristic approach to KP, usually called the *Greedy Algorithm*. Again, the worst-case performance can be as bad as 0 (take for example the series of problems introduced for z'), but can be improved to $\frac{1}{2}$ if we also consider the solution given by the item of maximum profit alone, as in the following implementation. We assume that the items are ordered according to (2.7).

```
procedure GREEDY;
input: n, c, (p_j), (w_j);
output: z^g, (x_j);
begin
    c̄ := c;
    z^g := 0;
    j* := 1;
    for j := 1 to n do
        begin
            if w_j > c̄ then x_j := 0
            else
                begin
                    x_j := 1;
```

```
                    c̄ := c̄ − w_j;
                    z^g := z^g + p_j
                end;
            if p_j > p_j* then j* := j
        end;
    if p_j* > z^g then
        begin
            z^g := p_j*;
            for j := 1 to n do x_j := 0;
            x_j* := 1
        end
end.
```

The worst-case performance ratio is $\frac{1}{2}$ since: (a) $p_{j^*} \geq p_s$, so $z^g \geq z^h$; (b) the series of problems introduced for z^h proves the tightness. The time complexity is $O(n)$, plus $O(n \log n)$ for the initial sorting.

For Example 2.1 we have $z' = z^h = 265$ and $z^g = 280$, which is the optimal solution value since $U_6 = 280$.

When a 0-1 knapsack problem in minimization form (see Section 2.1) is heuristically solved by applying GREEDY to its equivalent maximization instance, we of course obtain a feasible solution, but the worst-case performance is not preserved. Consider, in fact, the series of minimization problems with $n = 3$, $p_1 = w_1 = k$, $p_2 = w_2 = 1$, $p_3 = w_3 = k$ and $q = 1$, for which the optimal solution value is 1. Applying GREEDY to the maximization version (with $c = 2k$), we get $z^g = k + 1$ and hence an arbitrarily bad heuristic solution of value k for the minimization problem.

Other approximate algorithms for KP are considered in Section 2.8.

2.5 BRANCH-AND-BOUND ALGORITHMS

The first branch-and-bound approach to the exact solution of KP was presented by Kolesar (1967). His algorithm consists of a *highest-first* binary branching scheme which: (a) at each node, selects the not-yet-fixed item j having the maximum profit per unit weight, and generates two descendent nodes by fixing x_j, respectively, to 1 and 0; (b) continues the search from the feasible node for which the value of upper bound U_1 is a maximum.

The large computer memory and time requirements of the Kolesar algorithm were greatly reduced by the Greenberg and Hegerich (1970) approach, differing in two main respects: (a) at each node, the continuous relaxation of the induced subproblem is solved and the corresponding critical item $\tilde{s}$ is selected to generate the two descendent nodes (by imposing $x_{\tilde{s}} = 0$ and $x_{\tilde{s}} = 1$); (b) the search continues from the node associated with the exclusion of item $\tilde{s}$ (condition $x_{\tilde{s}} = 0$). When the continuous relaxation has an all-integer solution, the search is resumed from the last node generated by imposing $x_{\tilde{s}} = 1$, i.e. the algorithm is of *depth-first* type.

Horowitz and Sahni (1974) (and, independently, Ahrens and Finke (1975))

derived from the previous scheme a depth-first algorithm in which: (a) selection of the branching variable x_j is the same as in Kolesar; (b) the search continues from the node associated with the insertion of item j (condition $x_j = 1$), i.e. following a greedy strategy.

Other algorithms have been derived from the Greenberg–Hegerich approach (Barr and Ross (1975), Laurière (1978)) and from different techniques (Lageweg and Lenstra (1972), Guignard and Spielberg (1972), Fayard and Plateau (1975), Veliev and Mamedov (1981)). The Horowitz–Sahni one is, however, the most effective, structured and easy to implement, and has constituted the basis for several improvements.

2.5.1 The Horowitz–Sahni algorithm

Assume that the items are sorted as in (2.7). A *forward move* consists of inserting the largest possible set of new consecutive items into the current solution. A *backtracking move* consists of removing the last inserted item from the current solution. Whenever a forward move is exhausted, the upper bound U_1 corresponding to the current solution is computed and compared with the best solution so far, in order to check whether further forward moves could lead to a better one: if so, a new forward move is performed, otherwise a backtracking follows. When the last item has been considered, the current solution is complete and possible updating of the best solution so far occurs. The algorithm stops when no further backtracking can be performed.

In the following description of the algorithm we use the notations

$(\hat{x}_j)$ = current solution;

$\hat{z}$ = current solution value $\left(= \sum_{j=1}^{n} p_j \hat{x}_j\right)$;

$\hat{c}$ = current residual capacity $\left(= c - \sum_{j=1}^{n} w_j \hat{x}_j\right)$;

(x_j) = best solution so far;

z = value of the best solution so far $\left(= \sum_{j=1}^{n} p_j x_j\right)$.

procedure HS:
input: $n, c, (p_j), (w_j)$;
output: $z, (x_j)$;
begin
1. [initialize]
   $z := 0$;
   $\hat{z} := 0$;

$\hat{c} := c;$
$p_{n+1} := 0;$
$w_{n+1} := +\infty;$
$j := 1;$
2. [compute upper bound U_1]
find $r = \min \{i : \sum_{k=j}^{i} w_k > \hat{c}\};$
$u := \sum_{k=j}^{r-1} p_k + \lfloor (\hat{c} - \sum_{k=j}^{r-1} w_k) p_r / w_r \rfloor;$
if $z \geq \hat{z} + u$ **then go to** 5;
3. [perform a forward step]
while $w_j \leq \hat{c}$ **do**
begin
$\hat{c} := \hat{c} - w_j;$
$\hat{z} := \hat{z} + p_j;$
$\hat{x}_j := 1;$
$j := j + 1$
end;
if $j \leq n$ **then**
begin
$\hat{x}_j := 0;$
$j := j + 1$
end;
if $j < n$ **then go to** 2;
if $j = n$ **then go to** 3;
4. [update the best solution so far]
if $\hat{z} > z$ **then**
begin
$z := \hat{z};$
for $k := 1$ **to** n **do** $x_k := \hat{x}_k$
end;
$j := n;$
if $\hat{x}_n = 1$ **then**
begin
$\hat{c} := \hat{c} + w_n;$
$\hat{z} := \hat{z} - p_n;$
$\hat{x}_n := 0$
end;
5. [backtrack]
find $i = \max \{k < j : \hat{x}_k = 1\};$
if no such i **then return** ;
$\hat{c} := \hat{c} + w_i;$
$\hat{z} := \hat{z} - p_i;$
$\hat{x}_i := 0;$
$j := i + 1;$
go to 2
end.

Example 2.2

Consider the instance of KP defined by

$$
\begin{aligned}
n &= 7;\\
(p_j) &= (70, 20, 39, 37, 7, 5, 10);\\
(w_j) &= (31, 10, 20, 19, 4, 3, 6);\\
c &= 50.
\end{aligned}
$$

Figure 2.3 gives the decision-tree produced by procedure HS. □

Several effective algorithms have been obtained by improving the Horowitz–Sahni strategy. Mention is made in particular of those of Nauss (1976) (with Fortran code available), Martello and Toth (1977a) (with Fortran code in Martello and Toth (1978) and Pascal code in Syslo, Deo and Kowalik (1983)), Suhl (1978), Zoltners (1978).

We describe the Martello–Toth algorithm, which is generally considered highly effective.

2.5.2 The Martello–Toth algorithm

The method differs from that of Horowitz and Sahni (1974) in the following main respects (we use the notations introduced in the previous section).

(i) Upper bound U_2 is used instead of U_1.

(ii) The forward move associated with the selection of the jth item is split into two phases: *building of a new current solution* and *saving of the current solution.* In the first phase the largest set N_j of consecutive items which can be inserted into the current solution starting from the jth, is defined, and the upper bound corresponding to the insertion of the jth item is computed. If this bound is less than or equal to the value of the best solution so far, a backtracking move immediately follows. If it is greater, the second phase, that is, insertion of the items of set N_j into the current solution, is performed only if the value of such a new solution does not represent the maximum which can be obtained by inserting the jth item. Otherwise, the best solution so far is changed, but the current solution is not updated, so useless backtrackings on the items in N_j are avoided.

(iii) A particular forward procedure, based on dominance criteria, is performed whenever, before a backtracking move on the ith item, the residual capacity $\hat{c}$ does not allow insertion into the current solution of any item following the ith. The procedure is based on the following consideration: the current solution could be improved only if the ith item is replaced by an item having greater profit and a weight small enough to allow its insertion, or by at least two items having global weight not greater than $w_i + \hat{c}$. By this approach it is generally possible to eliminate most of the useless nodes generated at the lowest levels of the decision-tree.

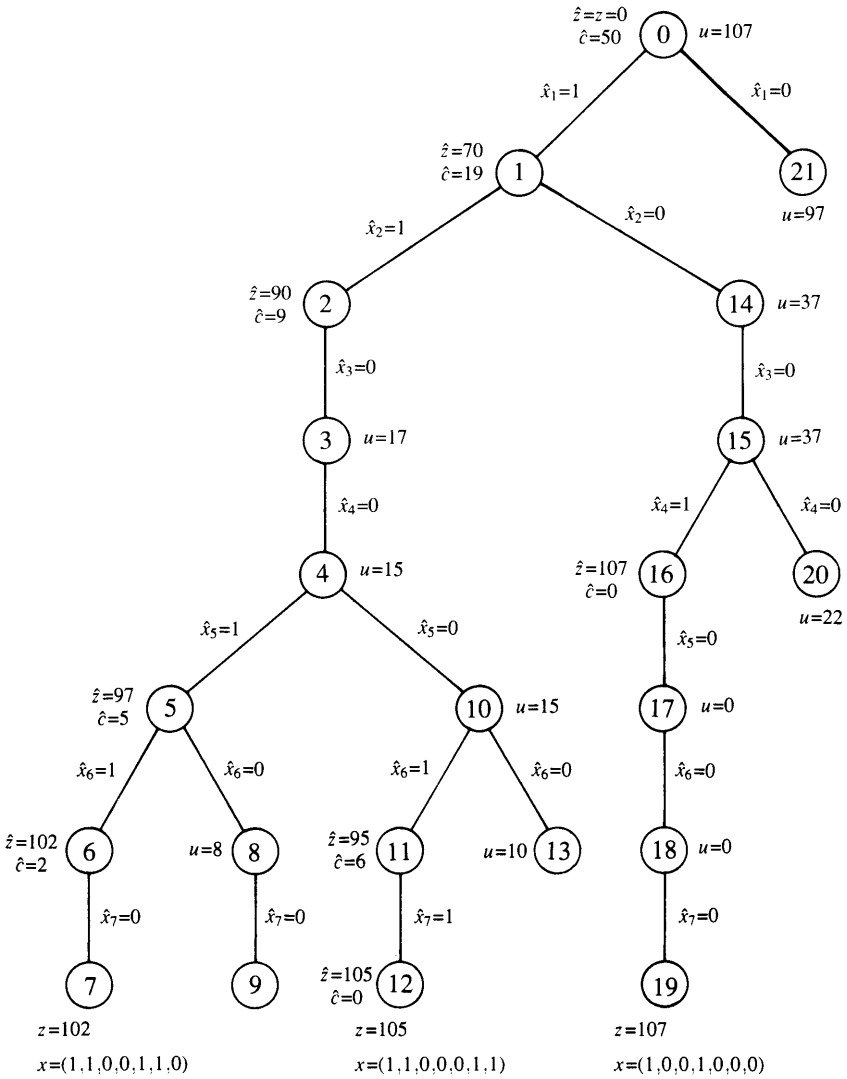

Figure 2.3 Decision-tree of procedure HS for Example 2.2

(iv) The upper bounds associated with the nodes of the decision-tree are computed through a parametric technique based on the storing of information related to the current solution. Suppose, in fact, that the current solution has been built by inserting all the items from the jth to the rth: then, when performing a backtracking on one of these items (say the ith, $j \leq i < r$), if no insertion occurred for the items preceding the jth, it is possible to insert at least items $i+1, \ldots, r$ into the new current solution. To this end, we store in $\overline{r}_i$, $\overline{p}_i$ and

$\overline{w}_i$ the quantities $r+1$, $\sum_{k=i}^{r} p_k$ and $\sum_{k=i}^{r} w_k$, respectively, for $i = j, \ldots, r$, and in $\tilde{r}$ the value $r-1$ (used for subsequent updatings).

Detailed description of the algorithm follows (it is assumed that the items are sorted as in (2.7)).

procedure MT1:
input: $n, c, (p_j), (w_j)$;
output: $z, (x_j)$;
begin
1. [initialize]
$z := 0$;
$\hat{z} := 0$;
$\hat{c} := c$;
$p_{n+1} := 0$;
$w_{n+1} := +\infty$;
for $k := 1$ **to** n **do** $\hat{x}_k := 0$;
compute the upper bound $U = U_2$ on the optimal solution value;
$\overline{w}_1 := 0$;
$\overline{p}_1 := 0$;
$\overline{r}_1 := 1$;
$\tilde{r} := n$;
for $k := n$ **to** 1 **step** -1 **do** compute $m_k = \min\{w_i : i > k\}$;
$j := 1$;
2. [build a new current solution]
while $w_j > \hat{c}$ **do**
if $z \geq \hat{z} + \lfloor \hat{c} p_{j+1} / w_{j+1} \rfloor$ **then go to** 5 **else** $j := j+1$;
find $r = \min\{i : \overline{w}_j + \sum_{k=\overline{r}_j}^{i} w_k > \hat{c}\}$;
$p' := \overline{p}_j + \sum_{k=\overline{r}_j}^{r-1} p_k$;
$w' := \overline{w}_j + \sum_{k=\overline{r}_j}^{r-1} w_k$;
if $r \leq n$ **then** $u := \max(\lfloor (\hat{c} - w') p_{r+1} / w_{r+1} \rfloor,$
$\lfloor p_r - (w_r - (\hat{c} - w')) p_{r-1} / w_{r-1} \rfloor)$
else $u := 0$;
if $z \geq \hat{z} + p' + u$ **then go to** 5;
if $u = 0$ **then go to** 4;
3. [save the current solution]
$\hat{c} := \hat{c} - w'$;
$\hat{z} := \hat{z} + p'$;
for $k := j$ **to** $r-1$ **do** $\hat{x}_k := 1$;
$\overline{w}_j := w'$;
$\overline{p}_j := p'$;
$\overline{r}_j := r$;
for $k := j+1$ **to** $r-1$ **do**
begin
$\overline{w}_k := \overline{w}_{k-1} - w_{k-1}$;
$\overline{p}_k := \overline{p}_{k-1} - p_{k-1}$;
$\overline{r}_k := r$

```
        end;
    for k := r to r̃ do
        begin
            w̄_k := 0;
            p̄_k := 0;
            r̄_k := k
        end;
    r̃ := r − 1;
    j := r + 1;
    if ĉ ≥ m_{j−1} then go to 2;
    if z ≥ ẑ then go to 5;
    p' := 0;
4. [update the best solution so far]
    z := ẑ + p';
    for k := 1 to j − 1 do x_k := x̂_k;
    for k := j to r − 1 do x_k := 1;
    for k := r to n do x_k := 0;
    if z = U then return ;
5. [backtrack]
    find i = max {k < j : x̂_k = 1};
    if no such i then return;
    ĉ := ĉ + w_i;
    ẑ := ẑ − p_i;
    x̂_i := 0;
    j := i + 1;
    if ĉ − w_i ≥ m_i then go to 2;
    j := i;
    h := i;
6. [try to replace item i with item h]
    h := h + 1;
    if z ≥ ẑ + ⌊ĉp_h/w_h⌋ then go to 5;
    if w_h = w_i then go to 6;
    if w_h > w_i then
        begin
            if w_h > ĉ or z ≥ ẑ + p_h then go to 6;
            z := ẑ + p_h;
            for k := 1 to n do x_k := x̂_k;
            x_h := 1;
            if z = U then return;
            i := h;
            go to 6
        end
    else
        begin
            if ĉ − w_h < m_h then go to 6;
            ĉ := ĉ − w_h;
            ẑ := ẑ + p_h;
            x̂_h := 1;
            j := h + 1;
```

```
            w̄_h := w_h;
            p̄_h := p_h;
            r̄_h := h + 1;
            for k := h + 1 to r̃ do
                begin
                    w̄_k := 0;
                    p̄_k := 0;
                    r̄_k := k
                end;
            r̃ := h;
            go to 2
        end
end.
```

The Fortran code corresponding to MT1 is included in the present volume. In addition, a second code, MT1R, is included which accepts on input real values for profits, weights and capacity.

Example 2.2 (continued)

Figure 2.4 gives the decision-tree produced by procedure MT1. □

Branch-and-bound algorithms are nowadays the most common way to effectively find the optimal solution of knapsack problems. More recent techniques imbed the branch-and-bound process into a particular algorithmic framework to solve, with increased efficiency, large instances of the problem. We describe them in Section 2.9.

The other fundamental approach to KP is dynamic programming. This has been the first technique available for exactly solving the problem and, although its importance has decreased in favour of branch-and-bound, it is still interesting because (a) it usually beats the other methods when the instance is very hard (see the computational results of Section 2.10.1), and (b) it can be successfully used in combination with branch-and-bound to produce hybrid algorithms for KP (Plateau and Elkihel, 1985) and for other knapsack-type problems (Martello and Toth (1984a), Section 4.2.2).

2.6 DYNAMIC PROGRAMMING ALGORITHMS

Given a pair of integers m $(1 \leq m \leq n)$ and $\hat{c}$ $(0 \leq \hat{c} \leq c)$, consider the sub-instance of KP consisting of items $1, \ldots, m$ and capacity $\hat{c}$. Let $f_m(\hat{c})$ denote its optimal solution value, i.e.

$$f_m(\hat{c}) = \max \left\{ \sum_{j=1}^{m} p_j x_j : \sum_{j=1}^{m} w_j x_j \leq \hat{c},\ x_j = 0 \text{ or } 1 \text{ for } j = 1, \ldots, m \right\}. \quad (2.29)$$

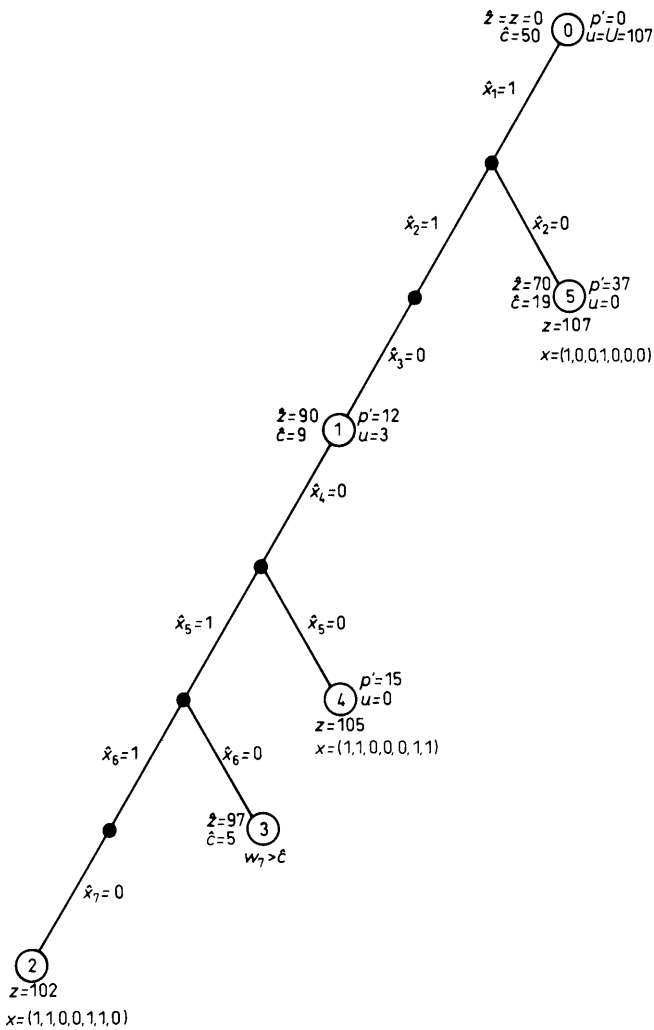

Figure 2.4 Decision-tree of procedure MT1 for Example 2.2

We trivially have

$$f_1(\hat{c}) = \begin{cases} 0 & \text{for } \hat{c} = 0, \ldots, w_1 - 1; \\ p_1 & \text{for } \hat{c} = w_1, \ldots, c. \end{cases}$$

Dynamic programming consists of considering n *stages* (for m increasing from 1 to n) and computing, at each stage $m > 1$, the values $f_m(\hat{c})$ (for $\hat{c}$ increasing from 0 to c) using the classical recursion (Bellman, 1954, 1957; Dantzig, 1957):

$$f_m(\hat{c}) = \begin{cases} f_{m-1}(\hat{c}) & \text{for } \hat{c} = 0, \ldots, w_m - 1; \\ \max\,(f_{m-1}(\hat{c}), f_{m-1}(\hat{c} - w_m) + p_m) & \text{for } \hat{c} = w_m, \ldots, c. \end{cases}$$

We call *states* the feasible solutions corresponding to the $f_m(\hat{c})$ values. The optimal solution of the problem is the state corresponding to $f_n(c)$.

Toth (1980) directly derived from the Bellman recursion an efficient procedure for computing the states of a stage. The following values are assumed to be defined before execution for stage m:

$$v = \min\left(\sum_{j=1}^{m-1} w_j, c\right); \tag{2.30}$$

$$b = 2^{m-1}; \tag{2.31}$$

$$P_{\hat{c}} = f_{m-1}(\hat{c}), \qquad \text{for } \hat{c} = 0, \ldots, v; \tag{2.32}$$

$$X_{\hat{c}} = \{x_{m-1}, x_{m-2}, \ldots, x_1\}, \qquad \text{for } \hat{c} = 0, \ldots, v, \tag{2.33}$$

where x_j defines the value of the jth variable in the partial optimal solution corresponding to $f_{m-1}(\hat{c})$, i.e.

$$\hat{c} = \sum_{j=1}^{m-1} w_j x_j \quad \text{and} \quad f_{m-1}(\hat{c}) = \sum_{j=1}^{m-1} p_j x_j.$$

From a computational point of view, it is convenient to encode each set $X_{\hat{c}}$ as a bit string, so this notation will be used in the following. After execution, values (2.30) to (2.33) are relative to stage m.

```
procedure REC1:
input: v, b, (P_ĉ), (X_ĉ), w_m, p_m;
output: v, b, (P_ĉ), (X_ĉ);
begin
    if v < c then
        begin
            u := v;
            v := min (v + w_m, c);
            for ĉ := u + 1 to v do
                begin
                    P_ĉ := P_u;
                    X_ĉ := X_u
                end
        end;
    for ĉ := v to w_m step −1 do
        if P_ĉ < P_(ĉ−w_m) + p_m then
            begin
```

```
                P_ĉ := P_{ĉ−w_m} + p_m;
                X_ĉ := X_{ĉ−w_m} + b
            end;
    b := 2b
end.
```

An immediate dynamic programming algorithm for KP is thus the following.

```
procedure DP1:
input: n, c, (p_j), (w_j);
output: z, (x_j);
begin
    for ĉ := 0 to w_1 − 1 do
        begin
            P_ĉ := 0;
            X_ĉ := 0
        end;
    v := w_1;
    b := 2;
    P_v := p_1;
    X_v := 1;
    for m := 2 to n do call REC1;
    z := P_c;
    determine (x_j) by decoding X_c
end.
```

Procedure REC1 requires $O(c)$ time, so the time complexity of DP1 is $O(nc)$. The space complexity is $O(nc)$. By encoding $X_{\hat{c}}$ as a bit string in computer words of d bits, the actual storage requirement is $(1 + \lceil n/d \rceil)c$, where $\lceil a \rceil$ is the smallest integer not less than a.

2.6.1 Elimination of dominated states

The number of states considered at each stage can be considerably reduced by eliminating *dominated states*, that is, those states $(P_{\hat{c}}, X_{\hat{c}})$ for which there exists a state (P_y, X_y) with $P_y \geq P_{\hat{c}}$ and $y < \hat{c}$. (Any solution obtainable from $(P_{\hat{c}}, X_{\hat{c}})$ can be obtained from (P_y, X_y).) This technique has been used by Horowitz and Sahni (1974) and Ahrens and Finke (1975). The undominated states of the mth stage can be computed through a procedure proposed by Toth (1980). The following values are assumed to be defined before execution of the procedure for stage m:

$$s = \text{number of states at stage } (m-1); \tag{2.34}$$

$$b = 2^{m-1}; \tag{2.35}$$

$$W1_i = \text{total weight of the } i\text{th state } (i = 1, \ldots, s); \tag{2.36}$$

$P1_i$ = total profit of the ith state ($i = 1, \ldots, s$); (2.37)

$X1_i = \{x_{m-1}, x_{m-2}, \ldots, x_1\}$, for $i = 1, \ldots, s$, (2.38)

where x_j defines the value of the jth variable in the partial optimal solution of the ith state, i.e.

$$W1_i = \sum_{j=1}^{m-1} w_j x_j \quad \text{and} \quad P1_i = \sum_{j=1}^{m-1} p_j x_j .$$

Vector $W1$ (and, hence, $P1$) is assumed to be ordered according to strictly increasing values.

The procedure uses index i to scan the states of the current stage and index k to store the states of the new stage. Each current state can produce a new state of total weight $y = W1_i + w_m$, so the current states of total weight $W1_h < y$, and then the new state, are stored in the new stage, but only if they are not dominated by a state already stored. After execution, values (2.34) and (2.35) are relative to the new stage, while the new values of (2.36), (2.37) and (2.38) are given by ($W2_k$), ($P2_k$) and ($X2_k$), respectively. Sets $X1_i$ and $X2_k$ are encoded as bit strings. Vectors ($W2_k$) and ($P2_k$) result ordered according to strictly increasing values. On input, it is assumed that $W1_0 = P1_0 = X1_0 = 0$.

```
procedure REC2:
input: s, b, (W1_i), (P1_i), (X1_i), w_m, p_m, c;
output: s, b, (W2_k), (P2_k), (X2_k);
begin
    i := 0;
    k := 0;
    h := 1;
    y := w_m;
    W1_{s+1} := +∞;
    W2_0 := 0;
    P2_0 := 0;
    X2_0 := 0;
    while min (y, W1_h) ≤ c do
        if W1_h ≤ y then
            begin
                comment: define the next state (p, x);
                p := P1_h;
                x := X1_h;
                if W1_h = y then
                    begin
                        if P1_i + p_m > p then
                            begin
                                p := P1_i + p_m;
```

```
                        x := X1_i + b
                    end;
                i := i + 1;
                y := W1_i + w_m
            end;
        comment: store the next state, if not dominated;
        if p > P2_k then
            begin
                k := k + 1;
                W2_k := W1_h;
                P2_k := p;
                X2_k := x
            end;
        h := h + 1
    end
else
    begin
        comment: store the new state, if not dominated;
        if P1_i + p_m > P2_k then
            begin
                k := k + 1;
                W2_k := y;
                P2_k := P1_i + p_m;
                X2_k := X1_i + b
            end;
        i := i + 1;
        y := W1_i + w_m
    end;
s := k;
b := 2b
end.
```

A dynamic programming algorithm using REC2 to solve KP is the following.

```
procedure DP2:
input: n, c, (p_j), (w_j);
output: z, (x_j);
begin
    W1_0 := 0;
    P1_0 := 0;
    X1_0 := 0;
    s := 1;
    b := 2;
    W1_1 := w_1;
    P1_1 := p_1;
    X1_1 := 1;
    for m := 2 to n do
        begin
```

```
            call REC2;
            rename W2_s, P2_s and X2_s as W1_s, P1_s and X1_s, respectively
        end;
    z := P1_s;
    determine (x_j) by decoding X1_s
end.
```

The time complexity of REC2 is $O(s)$. Since s is bounded by $\min(2^m - 1, c)$, the time complexity of DP2 is $O(\min(2^{n+1}, nc))$.

Procedure DP2 requires no specific ordering of the items. Its efficiency, however, improves considerably if they are sorted according to decreasing p_j/w_j ratios since, in this case, the number of undominated states is reduced. Hence, this ordering is assumed in the following.

Example 2.3

Consider the instance of KP defined by

$$n = 6;$$

$$(p_j) = (50, 50, 64, 46, 50, 5);$$

$$(w_j) = (56, 59, 80, 64, 75, 17);$$

$$c = 190.$$

Figure 2.5 gives, for each stage m and for each undominated state i, the values W_i, P_i, corresponding, in DP2, alternatively to $W1_i, P1_i$ and $W2_i, P2_i$. The optimal solution, of value 150, is $(x_j) = (1, 1, 0, 0, 1, 0)$. For the same example, procedure DP1 generates 866 states. □

	$m = 1$		$m = 2$		$m = 3$		$m = 4$		$m = 5$		$m = 6$	
i	W_i	P_i	W_i	P_i	W_i	P_i	W_i	P_i	W_i	P_i	W_i	P_i
0	0	0	0	0	0	0	0	0	0	0	0	0
1	56	50	56	50	56	50	56	50	56	50	17	5
2			115	100	80	64	80	64	80	64	56	50
3					115	100	115	100	115	100	73	55
4					136	114	136	114	136	114	80	64
5							179	146	179	146	97	69
6									190	150	115	100
7											132	105
8											136	114
9											153	119
10											179	146
11											190	150

Figure 2.5 States of procedure DP2 for Example 2.3

2.6.2 The Horowitz–Sahni algorithm

Horowitz and Sahni (1974) presented an algorithm based on the subdivision of the original problem of n variables into two subproblems, respectively of $q = \lceil n/2 \rceil$ and $r = n - q$ variables. For each subproblem a list containing all the undominated states relative to the last stage is computed; the two lists are then combined in order to find the optimal solution.

The main feature of the algorithm is the need, in the worst case, for two lists of $2^q - 1$ states each, instead of a single list of $2^n - 1$ states. Hence the time and space complexities decrease to $O(\min(2^{n/2}, nc))$, with a square root improvement in the most favourable case. In many cases, however, the number of undominated states is much lower than $2^{n/2}$, since (a) many states are dominated and (b) for n sufficiently large, we have, in general, $c \ll 2^{n/2}$.

Ahrens and Finke (1975) proposed an algorithm where the technique of Horowitz and Sahni is combined with a branch-and-bound procedure in order to further reduce storage requirements. The algorithm works very well for hard problems having low values of n and very high values of w_i and c, but has the disadvantage of always executing the branch-and-bound procedure, even when the storage requirements are not excessive.

We illustrate the Horowitz–Sahni algorithm with a numerical example.

Example 2.3 (continued)

We have $q = 3$. The algorithm generates the first list for m =1, 2, 3, and the second for $m = 4, 5, 6$. The corresponding undominated states are given in Figure 2.6. Combining the lists corresponding to $m = 3$ and $m = 6$ we get the final list of Figure 2.5. □

	$m = 1$	$m = 2$	$m = 3$	$m = 6$	$m = 5$	$m = 4$
i	W_i P_i	W_i P_i	W_i P_i	W_i P_i	W_i P_i	W_i P_i
0	0 0	0 0	0 0	0 0	0 0	0 0
1	56 50	56 50	56 50	17 5	64 46	64 46
2		115 100	80 64	64 46	75 50	
3			115 100	75 50	139 96	
4			136 114	81 51		
5				92 55		
6				139 96		
7				156 101		

Figure 2.6 States of the Horowitz–Sahni algorithm for Example 2.3

2.6.3 The Toth algorithm

Toth (1980) presented a dynamic programming algorithm based on (a) the elimination of useless states and (b) a combination of procedures REC1 and REC2.

Several states computed by REC1 or REC2 are of no use for the following stages since, of course, we are only interested in states capable of producing, at the final stage, the optimal solution. Useless states produced by REC1 can be eliminated by the following rule:

If a state, defined at the mth stage, has a total weight W satisfying one of the conditions

$$\text{(i)}\quad W < c - \sum_{j=m+1}^{n} w_j,$$

$$\text{(ii)}\quad c - \min_{m<j\leq n}\{w_j\} < W < c,$$

then the state will never produce P_c and, hence, can be eliminated.

A similar rule can be given for REC2 (in this case, however, it is necessary to keep the largest-weight state satisfying (i)), and the last, i.e. sth, state. The rule cannot be extended, instead, to the Horowitz–Sahni algorithm, since, in order to combine the two lists, all the undominated states relative to the two subproblems must be known.

Example 2.3 (continued)

The states generated by DP2, with REC2 improved through the above elimination rule, are given in Figure 2.7. □

	$m=1$		$m=2$		$m=3$		$m=4$		$m=5$		$m=6$	
i	W_i	P_i	W_i	P_i	W_i	P_i	W_i	P_i	W_i	P_i	W_i	P_i
0	0	0	0	0	0	0	0	0	0	0	0	0
1	56	50	56	50	56	50	80	64	136	114	190	150
2			115	100	80	64	115	100	190	150		
3					115	100	136	114				
4					136	114	179	146				

Figure 2.7 States of the improved version of DP2 for Example 2.3

Algorithm DP2 is generally more efficient than DP1, because of the fewer number of states produced. Notice however that, for the computation of a single state, the time and space requirements of DP2 are higher. So, for hard problems, where very few states are dominated, and hence the two algorithms generate almost the same lists, DP1 must be preferred to DP2. A dynamic programming algorithm which effectively solves both easy and hard problems can thus be obtained by combining the best characteristics of the two approaches. This is achieved by using

procedure REC2 as long as the number of generated states is low, and then passing to REC1. Simple heuristic rules to determine the iteration at which the procedure must be changed can be found in Toth (1980).

2.7 REDUCTION ALGORITHMS

The size of an instance of KP can be reduced by applying procedures to fix the optimal value of as many variables as possible. These procedures partition set $N = \{1, 2, \ldots, n\}$ into three subsets:

$$J1 = \{j \in N : x_j = 1 \text{ in any optimal solution to KP}\},$$

$$J0 = \{j \in N : x_j = 0 \text{ in any optimal solution to KP}\},$$

$$F = N \setminus (J1 \cup J0).$$

The original KP can then be transformed into the reduced form

$$\text{maximize} \quad z = \sum_{j \in F} p_j x_j + \hat{p}$$

$$\text{subject to} \quad \sum_{j \in F} w_j x_j \leq \hat{c},$$

$$x_j = 0 \text{ or } 1, \qquad j \in F,$$

where $\hat{p} = \sum_{j \in J1} p_j$, $\hat{c} = c - \sum_{j \in J1} w_j$.

Ingargiola and Korsh (1973) proposed the first method for determining $J1$ and $J0$. The basic idea is the following. If setting a variable x_j to a given value b ($b = 0$ or 1) produces infeasibility or implies a solution worse than an existing one, then x_j must take the value $(1 - b)$ in any optimal solution. Let l be the value of a feasible solution to KP, and, for $j \in N$, let u_j^1 (resp. u_j^0) be an upper bound for KP with the additional constraint $x_j = 1$ (resp. $x_j = 0$). Then we have

$$J1 = \{j \in N : u_j^0 < l\}, \tag{2.39}$$

$$J0 = \{j \in N : u_j^1 < l\}. \tag{2.40}$$

In the Ingargiola–Korsh algorithm, u_j^1 and u_j^0 are computed using the Dantzig bound. Let s be the critical item (see Section 2.2.1) and U_1 the Dantzig bound for the original problem. Then $u_j^1 = U_1$ for any $j < s$ and $u_j^0 = U_1$ for any $j > s$. Hence values $j > s$ (resp. $j < s$) need not be considered in determining $J1$ (resp. $J0$), since $U_1 \geq l$. The algorithm initializes l to $\sum_{j=1}^{s-1} p_j$ and improves it during execution. It is assumed that the items are ordered according to (2.7). Remember that $\sigma^1(j)$ and $\sigma^0(j)$ represent the critical item when it is imposed, respectively,

$x_j = 1$ and $x_j = 0$ (see (2.17) and (2.18)).

procedure IKR:
input: $n, c, (p_j), (w_j)$;
output: $J1, J0$;
begin
  $J1 := \emptyset$;
  $J0 := \emptyset$;
  determine $s = \min \{j : \sum_{i=1}^{j} w_i > c\}$;
  $l := \sum_{j=1}^{s-1} p_j$;
  for $j := 1$ **to** s **do**
    begin
      determine $\sigma^0(j)$ and compute u_j^0;
      $l := \max (l, \sum_{\substack{i=1 \\ i \neq j}}^{\sigma^0(j)-1} p_i)$;
      if $u_j^0 < l$ **then** $J1 := J1 \cup \{j\}$
    end;
  for $j := s$ **to** n **do**
    begin
      determine $\sigma^1(j)$ and compute u_j^1;
      $l := \max (l, p_j + \sum_{i=1}^{\sigma^1(j)-1} p_i)$;
      if $u_j^1 < l$ **then** $J0 := J0 \cup \{j\}$
    end
end.

Notice that the variables corresponding to items in $J1$ and $J0$ must take the fixed value in *any* optimal solution to KP, thus including the solution of value l when this is optimal. However, given a feasible solution $\tilde{x}$ of value l, we are only interested in finding a *better* one. Hence stronger definitions of $J1$ and $J0$ are obtained by replacing strict inequalities with inequalities in (2.39), (2.40), i.e.

$$J1 = \{j \in N : u_j^0 \leq l\}, \tag{2.41}$$

$$J0 = \{j \in N : u_j^1 \leq l\}. \tag{2.42}$$

If it turns out that the reduced problem is infeasible or has an optimal solution less than l, then $\tilde{x}$ is the optimal solution to the original problem.

Example 2.4

We use the same instance as in Example 2.2, whose optimal solution, of value 107, is $x = (1, 0, 0, 1, 0, 0, 0)$:

$n = 7;$

$(p_j) = (70, 20, 39, 37, 7, 5, 10);$

$(w_j) = (31, 10, 20, 19, 4, 3, 6);$

$c = 50.$

Applying procedure IKR we get:

$s = 3; \quad l = 90;$

$j = 1: \quad u_1^0 = 97; \quad l = 96;$

$j = 2: \quad u_2^0 = 107;$

$j = 3: \quad u_3^0 = 107;$

$j = 3: \quad u_3^1 = 106;$

$j = 4: \quad u_4^1 = 107; \quad l = 107;$

$j = 5: \quad u_5^1 = 106;$

$j = 6: \quad u_6^1 = 106;$

$j = 7: \quad u_7^1 = 105;$

so $J1 = \emptyset$, $J0 = \{5, 6, 7\}$. □

In order to use definitions (2.41), (2.42) it is simply necessary to replace the $<$ sign with $\leq$ in the two tests of procedure IKR. With this modification we get $J1 = \emptyset$, $J0 = \{4, 5, 6, 7\}$. The optimal solution value of the reduced problem is then 90, implying that the feasible solution of value $l = 107$ is optimal. (Notice that it is worth storing the solution vector corresponding to l during execution.)

Recently, Murphy (1986) erroneously claimed that definitions (2.41), (2.42) of $J1$ and $J0$ are incorrect. Balas, Nauss and Zemel (1987) have pointed out its mistake.

The time complexity of the Ingargiola–Korsh procedure is $O(n^2)$, since $O(n)$ time is required for each $\sigma^0(j)$ or $\sigma^1(j)$ computation (although one can expect that, on average, these values can be determined with few operations, starting from s). The time complexity does not change if u_j^0 and u_j^1 are computed through one of the improved upper bounding techniques of Section 2.3.

An $O(n)$ reduction algorithm has been independently obtained by Fayard and Plateau (1975) and Dembo and Hammer (1980). The method, FPDHR, computes u_j^0 and u_j^1 through the values $p_j^* = p_j - w_j p_s / w_s$ (see (2.13)). Recalling that $| p_j^* |$ represents a lower bound on the decrease of $z(C(KP))$ corresponding to the change of the jth variable from $\bar{x}_j$ to $1 - \bar{x}_j$, we have

$$u_j^0 = \lfloor z(C(KP)) - p_j^* \rfloor, \qquad j = 1, \ldots, s;$$

$$u_j^1 = \lfloor z(C(KP)) + p_j^* \rfloor, \qquad j = s, \ldots, n,$$

which are computed in constant time, once $z(C(KP))$ is known. It is easy to see that the values u_j^0 and u_j^1 obtained in this way are not lower than those of procedure IKR, so the method is generally less effective, in the sense that the resulting sets $J0$ and $J1$ have smaller cardinality.

$O(n^2)$ reduction algorithms more effective than the Ingargiola–Korsh method have been obtained by Toth (1976), Laurière (1978) and Fayard and Plateau (1982).

An effective reduction method, still dominating the Ingargiola–Korsh one, but requiring $O(n\log n)$ time, has been proposed by Martello and Toth (1988). The algorithm differs from procedure IKR in the following main respects:

(a) u_j^0 and u_j^1 are computed through the stronger bound U_2;

(b) $J1$ and $J0$ are determined at the end, thus using the best heuristic solution found;

(c) at each iteration, upper bound and improved heuristic solution value are computed in $O(\log n)$ time by initially defining $\overline{w}_j = \sum_{i=1}^{j} w_i$ and $\overline{p}_j = \sum_{i=1}^{j} p_i$ $(j = 1, \ldots, n)$ and then determining, through binary search, the current critical item $\overline{s}$ (i.e. $\sigma^0(j)$ or $\sigma^1(j)$).

The procedure assumes that the items are ordered according to (2.7) and that $p_j/w_j = -\infty$ if $j < 1$, $p_j/w_j = +\infty$ if $j > n$.

procedure MTR:
input: $n, c, (p_j), (w_j)$;
output: $J1, J0, l$;
begin
  for $j := 0$ **to** n **do** compute $\overline{p}_j = \sum_{i=1}^{j} p_i$ and $\overline{w}_j = \sum_{i=1}^{j} w_i$;
  find, through binary search, s such that $\overline{w}_{s-1} \leq c < \overline{w}_s$;
  $l := \overline{p}_{s-1}$;
  $\overline{c} := c - \overline{w}_{s-1}$;
  for $j := s + 1$ **to** n **do**
    if $w_j \leq \overline{c}$ **then**
      begin
        $l := l + p_j$;
        $\overline{c} := \overline{c} - w_j$
      end;
  for $j := 1$ **to** s **do**
    begin
      find, through binary search, $\overline{s}$ such that
        $\overline{w}_{\overline{s}-1} \leq c + w_j < \overline{w}_{\overline{s}}$;
      $\overline{c} := c + w_j - \overline{w}_{\overline{s}-1}$;

$u_j^0 := \overline{p}_{\overline{s}-1} - p_j + \max\left(\lfloor \overline{c} p_{\overline{s}+1} / w_{\overline{s}+1} \rfloor,\right.$
$\left. \lfloor p_{\overline{s}} - (w_{\overline{s}} - \overline{c}) p_{\overline{s}-1} / w_{\overline{s}-1} \rfloor\right);$
$l := \max\left(l, \overline{p}_{\overline{s}-1} - p_j\right)$
end;
for $j := s$ **to** n **do**
begin
find, through binary search, $\overline{s}$ such that
$\overline{w}_{\overline{s}-1} \leq c - w_j < \overline{w}_{\overline{s}}$;
$\overline{c} := c - w_j - \overline{w}_{\overline{s}-1}$;
$u_j^1 := \overline{p}_{\overline{s}-1} + p_j + \max\left(\lfloor \overline{c} p_{\overline{s}+1} / w_{\overline{s}+1} \rfloor,\ \lfloor p_{\overline{s}} - (w_{\overline{s}} - \overline{c}) p_{\overline{s}-1} / w_{\overline{s}-1} \rfloor\right)$;
$l := \max\left(l, \overline{p}_{\overline{s}-1} + p_j\right)$
end;
$J1 := \{j \leq s : u_j^0 \leq l\}$;
$J0 := \{j \geq s : u_j^1 \leq l\}$
end.

Example 2.4 (continued)

Applying procedure MTR we have

$(\overline{p}_j) = (0, 70, 90, 129, 166, 173, 178, 188)$;
$(\overline{w}_j) = (0, 31, 41, 61, 80, 84, 87, 93)$;
$s = 3$, $l = 90$, $\overline{c} = 9$;
$l = 102$, $\overline{c} = 2$;
$j = 1$: $\overline{s} = 5$, $\overline{c} = 1$, $u_1^0 = 97$;
$j = 2$: $\overline{s} = 3$, $\overline{c} = 19$, $u_2^0 = 107$;
$j = 3$: $\overline{s} = 4$, $\overline{c} = 9$, $u_3^0 = 107$;
$j = 3$: $\overline{s} = 1$, $\overline{c} = 30$, $u_3^1 = 99$;
$j = 4$: $\overline{s} = 2$, $\overline{c} = 0$, $u_4^1 = 107$, $l = 107$;
$j = 5$: $\overline{s} = 3$, $\overline{c} = 5$, $u_5^1 = 106$;
$j = 6$: $\overline{s} = 3$, $\overline{c} = 6$, $u_6^1 = 106$;
$j = 7$: $\overline{s} = 3$, $\overline{c} = 3$, $u_7^1 = 105$;
$J1 = \{1, 2, 3\}$, $J0 = \{3, 4, 5, 6, 7\}$.

The reduced problem is infeasible (x_3 is fixed both to 1 and to 0 and, in addition, $\sum_{j \in J1} w_j > c$), so the feasible solution of value 107 is optimal. □

Procedure MTR computes the initial value of l through the greedy algorithm. Any other heuristic, requiring no more than $O(n \log n)$ time, could be used with no time complexity alteration.

The number of fixed variables can be further increased by imposing conditions (2.5), (2.6) to the reduced problem, i.e. setting $J0 = J0 \cup \{j \in F : w_j >$

$c - \sum_{j \in J1} w_j\}$ and, if $\sum_{j \in F} w_j \leq c - \sum_{j \in J1} w_j$, $J1 = J1 \cup F$. In addition, the procedure can be re-executed for the items in F (since the values of u_j^0 and u_j^1 relative to the reduced problem can decrease) until no further variable is fixed. This, however, would increase the time complexity by a factor n, unless the number of re-executions is bounded by a constant.

2.8 APPROXIMATE ALGORITHMS

In Section 2.4 we have described the greedy algorithm, which provides an approximate solution to KP with worst-case performance ratio equal to $\frac{1}{2}$, in time $O(n)$ plus $O(n \log n)$ for the initial sorting. Better accuracy can be obtained through approximation schemes, which allow one to obtain any prefixed performance ratio. In this section we examine polynomial-time and fully polynomial-time approximation schemes for KP. Besides these deterministic results, the probabilistic behaviour of some approximate algorithms has been investigated. A thorough analysis of probabilistic aspects is outwith the scope of this book. The main results are outlined in Section 2.8.3 and, for the subset-sum problem, in Section 4.3.4. (The contents of such sections are based on Karp, Lenstra, McDiarmid and Rinnooy Kan (1985).)

2.8.1 Polynomial-time approximation schemes

The first approximation scheme for KP was proposed by Sahni (1975) and makes use of a greedy-type procedure which finds a heuristic solution by filling, in order of decreasing p_j/w_j ratios, that part of c which is left vacant after the items of a given set M have been put into the knapsack. Given $M \subset N$ and assuming that the items are sorted according to (2.7), the procedure is as follows.

```
procedure GS:
input: n, c, (p_j), (w_j), M;
output: z^g, X;
begin
    z^g := 0;
    ĉ := c − Σ_{j∈M} w_j;
    X := ∅;
    for j := 1 to n do
        if j ∉ M and w_j ≤ ĉ then
            begin
                z^g := z^g + p_j;
                ĉ := ĉ − w_j;
                X := X ∪ {j}
            end
end.
```

Given a non-negative integer parameter k, the Sahni scheme S(k) is

```
procedure S(k):
input: n, c, (p_j), (w_j);
output: z^h, X^h;
begin
    z^h := 0;
    for each M ⊂ {1, ..., n} such that |M| ≤ k and Σ_{j∈M} w_j ≤ c do
        begin
            call GS;
            if z^g + Σ_{j∈M} p_j > z^h then
                begin
                    z^h := z^g + Σ_{j∈M} p_j;
                    X^h := X ∪ M
                end
        end
end.
```

Since the time complexity of procedure GS is $O(n)$ and the number of times it is executed is $O(n^k)$, the time complexity of S(k) is $O(n^{k+1})$. The space complexity is $O(n)$.

Theorem 2.3 (Sahni, 1975) *The worst-case performance ratio of* S(k) *is* $r(\mathrm{S}(k)) = k/(k+1)$.

Proof. (a) Let Y be the set of items inserted into the knapsack in the optimal solution. If $|Y| \leq k$, then S(k) gives the optimum, since all combinations of size $|Y|$ are tried. Hence, assume $|Y| > k$. Let $\overline{M}$ be the set of the k items of highest profit in Y, and denote the remaining items of Y with $j_1, \ldots, j_r$, assuming $p_{j_i}/w_{j_i} \geq p_{j_{i+1}}/w_{j_{i+1}}$ ($i = 1, \ldots, r-1$). Hence, if z is the optimal solution value, we have

$$p_{j_i} \leq \frac{z}{k+1} \qquad \text{for } i = 1, \ldots, r. \tag{2.43}$$

Consider now the iteration of S(k) in which $M = \overline{M}$, and let j_m be the first item of $\{j_1, \ldots, j_r\}$ not inserted into the knapsack by GS. If no such item exists then the heuristic solution is optimal. Otherwise we can write z as

$$z = \sum_{i \in \overline{M}} p_i + \sum_{i=1}^{m-1} p_{j_i} + \sum_{i=m}^{r} p_{j_i}, \tag{2.44}$$

while for the heuristic solution value returned by GS we have

$$z^g \geq \sum_{i \in \overline{M}} p_i + \sum_{i=1}^{m-1} p_{j_i} + \sum_{i \in Q} p_i, \tag{2.45}$$

where Q denotes the set of those items of $N \setminus \overline{M}$ which are in the heuristic solution but not in $\{j_1, \ldots, j_r\}$ and whose index is less than j_m. Let $c^* = c - \sum_{i \in \overline{M}} w_i - \sum_{i=1}^{m-1} w_{j_i}$ and $\bar{c} = c^* - \sum_{i \in Q} w_i$ be the residual capacities available, respectively, in the optimal and the heuristic solution for the items of $N \setminus \overline{M}$ following j_{m-1}. Hence, from (2.44),

$$z \leq \sum_{i \in \overline{M}} p_i + \sum_{i=1}^{m-1} p_{j_i} + c^* \frac{p_{j_m}}{w_{j_m}};$$

by definition of m we have $\bar{c} < w_{j_m}$ and $p_i/w_i \geq p_{j_m}/w_{j_m}$ for $i \in Q$, so

$$z < \sum_{i \in \overline{M}} p_i + \sum_{i=1}^{m-1} p_{j_i} + p_{j_m} + \sum_{i \in Q} p_i.$$

Hence, from (2.45), $z < z^g + p_{j_m}$ and, from (2.43),

$$\frac{z^g}{z} > \frac{k}{k+1}.$$

(b) To prove that the bound is tight, consider the series of instances with: $n = k+2$; $p_1 = 2$, $w_1 = 1$; $p_j = w_j = L > 2$ for $j = 2, \ldots, k+2$; $c = (k+1)L$. The optimal solution value is $z = (k+1)L$, while $S(k)$ gives $z^h = kL + 2$. Hence, for L sufficiently large, the ratio z^k/z is arbitrarily close to $k/(k+1)$. □

Let $\overline{M}$ denote the maximum cardinality subset of $\{1, \ldots, n\}$ such that $\sum_{j \in \overline{M}} w_j \leq c$. Then, clearly, for any $k \geq |\overline{M}|$, $S(k)$ gives the optimal solution.

Example 2.5

Consider the instance of KP defined by

$n = 8;$

$(p_j) = (350, 400, 450, 20, 70, 8, 5, 5);$

$(w_j) = (25, 35, 45, 5, 25, 3, 2, 2);$

$c = 104.$

The optimal solution $X = \{1, 3, 4, 5, 7, 8\}$ has value $z = 900$.

Applying $S(k)$ with $k = 0$, we get the greedy solution: $X^h = \{1, 2, 4, 5, 6, 7, 8\}$, $z^h = 858$.

Applying $S(k)$ with $k = 1$, we re-obtain the greedy solution for $M = \{1\}, \{2\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}$. For $M = \{3\}$, we obtain $X^h = \{1, 3, 4, 5, 6\}$, $z^h = 898$.

Applying $S(k)$ with $k = 2$, we obtain the optimal solution when $M = \{3, 7\}$. □

The Sahni algorithm is a *polynomial-time approximation scheme*, in the sense that any prefixed worst-case performance ratio can be obtained in a time bounded by a polynomial. However, the degree of the polynomial increases with k, so the time complexity of the algorithm is exponential in k, i.e. in the inverse of the *worst-case relative error* $\varepsilon = 1 - r$.

2.8.2 Fully polynomial-time approximation schemes

Ibarra and Kim (1975) have obtained a *fully polynomial-time approximation scheme*, i.e. a parametric algorithm which allows one to obtain any worst-case relative error (note that imposing ε is equivalent to imposing r) in polynomial time and space, and such that the time and space complexities grow polynomially also with the inverse of the worst-case relative error ε. The basic ideas in the Ibarra–Kim algorithm are: (a) to separate items according to profits into a class of "large" items and one of "small" items; (b) to solve the problem for the large items only, with profits scaled by a suitable scale factor δ, through dynamic programming. The dynamic programming list is stored in a table T of length $\lfloor (3/\varepsilon)^2 \rfloor + 1$; $T(k)$ = *"undefined"* or is of the form $(L(k), P(k), W(k))$, where $L(k)$ is a subset of $\{1,\ldots,n\}$, $P(k) = \sum_{j \in L(k)} p_j$, $W(k) = \sum_{j \in L(k)} w_j$ and $k = \sum_{j \in L(k)} \bar{p}_j$ with $\bar{p}_j = \lfloor p_j/\delta \rfloor$. It is assumed that the items are ordered according to (2.7) and that the "small" items are inserted in set S preserving this order.

procedure IK(ε) :
input: $n, c, (p_j), (w_j)$;
output: z^h, X^h;
begin
  find the critical item s (see Section 2.2.1);
  if $\sum_{j=1}^{s-1} w_j = c$ **then**
    begin
      $z^h := \sum_{j=1}^{s-1} p_j$;
      $X^h := \{1,\ldots,s-1\}$;
      return
    end;
  $\tilde{z} := \sum_{j=1}^{s} p_j$
  comment: $\tilde{z}/2 \leq z < \tilde{z}$, since $z \geq \max(\sum_{j=1}^{s-1} p_j, p_s)$;
  $\delta := \tilde{z}(\varepsilon/3)^2$;
  $S := \emptyset$;
  $T(0) := (L(0), P(0), W(0)) := (\emptyset, 0, 0)$;
  $q := \lfloor \tilde{z}/\delta \rfloor$ (**comment**: $q = \lfloor (3/\varepsilon)^2 \rfloor$);
  comment: dynamic programming phase;
  for $i := 1$ **to** q **do** $T(i) :=$ *"undefined"*;
  for $j := 1$ **to** n **do**
    if $p_j \leq \varepsilon\tilde{z}/3$ **then** $S := S \cup \{j\}$
    else

```
            begin
                p̄_j := ⌊p_j/δ⌋;
                for i := q − p̄_j to 0 step −1 do
                    if T(i) ≠ "undefined" and W(i) + w_j ≤ c then
                        if T(i + p̄_j) = "undefined"
                          or W(i + p̄_j) > W(i) + w_j then
                            T(i + p̄_j) := (L(i) ∪ {j}, P(i) + p_j, W(i) + w_j)
            end;
    comment: greedy phase;
    z^h := 0;
    for i := 0 to q do
        if T(i) ≠ "undefined" then
            begin
                z̄ := P(i) + Σ_{j∈A} p_j, where A is obtained by filling the residual
                  capacity c − W(i) with items of S in the greedy way;
                if z̄ > z^h then
                    begin
                        z^h := z̄;
                        X^h := L(i) ∪ A
                    end
            end
end.
```

The dynamic programming recursion is executed n times and, at each iteration, no more than q states are considered: since each state takes a constant amount of time, the dynamic programming phase has time complexity $O(nq)$. The final greedy phase is performed at most q times, each iteration taking $O(n)$ time. Hence the overall time complexity of IK(ε) is $O(nq)$, i.e. $O(n/\varepsilon^2)$ by definition of q, plus $O(n\log n)$ for the initial sorting.

The space required by the algorithm is determined by the $\lfloor(3/\varepsilon)^2\rfloor$ entries of table T. Each entry needs no more than $2 + t$ words, where t is the number of items defining the state. If $\bar{p}_{i_1},\ldots,\bar{p}_{i_t}$ are the scaled profits of such items, we have $t \leq q/\min\{\bar{p}_{i_1},\ldots,\bar{p}_{i_t}\} \leq 3/\varepsilon$. Hence the overall space complexity of IK(ε) is $O(n)$ (for the input) + $O(1/\varepsilon^3)$.

Theorem 2.4 (Ibarra and Kim, 1975) *For any instance of KP,* $(z - z^h)/z \leq \varepsilon$, *where* z *is the optimal solution value and* z^h *the value returned by* IK(ε).

Proof. If the algorithm terminates in the initial phase with $z^h = \sum_{j=1}^{s-1} p_j$ then z^h gives the optimal solution. Otherwise, let $\{i_1,\ldots,i_k\}$ be the (possibly empty) set of items with $p_{i_l} > \frac{1}{3}\varepsilon\tilde{z}$ in the optimal solution, i.e.

$$z = \sum_{l=1}^{k} p_{i_l} + \alpha,$$

where α is a sum of profits of items in S. Defining $\overline{p}^* = \sum_{l=1}^{k} \overline{p}_{i_l}$ and $w^* = \sum_{l=1}^{k} w_{i_l}$, we have, at the end of the dynamic programming phase, $T(\overline{p}^*) \neq$ "*undefined*" and $W(\overline{p}^*) \leq w^*$ (since $W(i)$ is never increased by the algorithm). Let $L(\overline{p}^*) = \{j_1, \ldots, j_h\}$. (This implies that $\overline{p}^* = \sum_{l=1}^{h} \overline{p}_{j_l}$ and $W(\overline{p}^*) = \sum_{l=1}^{h} w_{j_l}$.) Then the sum $\overline{z} = \sum_{l=1}^{h} p_{j_l} + \beta$, where β is a sum of profits of elements in S, has been considered in the greedy phase (when $i = \overline{p}^*$), so $z^h \geq \overline{z}$. Observe that $\overline{p}_j = \lfloor p_j/\delta \rfloor \geq 3/\varepsilon$, from which $\overline{p}_j \delta \leq p_j \leq (\overline{p}_j + 1)\delta = \overline{p}_j \delta(1 + 1/\overline{p}_j) \leq \overline{p}_j \delta(1 + \varepsilon/3)$. It follows that

$$\overline{p}^* \delta + \alpha \leq z \leq \overline{p}^* \delta(1 + \tfrac{1}{3}\varepsilon) + \alpha,$$

$$\overline{p}^* \delta + \beta \leq \overline{z} \leq \overline{p}^* \delta(1 + \tfrac{1}{3}\varepsilon) + \beta,$$

from which

$$\frac{z - \overline{z}}{z} \leq \frac{\overline{p}^* \delta \varepsilon/3 + (\alpha - \beta)}{z} \leq \tfrac{1}{3}\varepsilon + \frac{\alpha - \beta}{z}.$$

Since $W(\overline{p}^*) \leq w^*$ and the items in S are ordered by decreasing p_j/w_j ratios, it follows that $(\alpha - \beta)$ cannot be greater than the maximum profit of an item in S, i.e. $\alpha - \beta \leq \frac{1}{3}\varepsilon\tilde{z}$. Hence $(z - \overline{z})/z \leq \frac{1}{3}\varepsilon(1 + \tilde{z}/z)$. Since $\overline{z} \leq z^h$ and $\tilde{z} \leq 2z$, then $(z - z^h)/z \leq \varepsilon$. □

Example 2.5 (continued)

We apply IK(ε) with $\varepsilon = \frac{1}{2}$.

$s = 3$;

$\tilde{z} = 1200$;

$\delta = \dfrac{100}{3}$;

$S = \emptyset$ (items with $p_j \leq \varepsilon\tilde{z}/3 = 200$ will be inserted in S);

$T(0) = (\emptyset, 0, 0)$;

$q = 36$;

dynamic programming phase:

$j = 1$: $\overline{p}_1 = 10$, $T(10) = (\{1\}, 350, 25)$;

$j = 2$: $\overline{p}_2 = 12$, $T(22) = (\{1, 2\}, 750, 60)$,

$T(12) = (\{2\}, 400, 35)$;

$j = 3$: $\overline{p}_3 = 13$, $T(25) = (\{2, 3\}, 850, 80)$,

$$T(23) = (\{1, 3\}, 800, 70),$$

$$T(13) = (\{3\}, 450, 45);$$

$j = 4, \ldots, 8$: $S = \{4, 5, 6, 7, 8\}$;

greedy phase:

for all the entries of table T save $T(23)$ and $T(25)$, we have $c - W(i) \geq \sum_{j \in S} w_j = 37$. Hence the best solution produced by such states is $P(22) + \sum_{j \in S} p_j = 858$. $T(23)$ gives $P(23) + \sum_{j \in \{4,5,6\}} p_j = 898$; $T(25)$ gives $P(25) + \sum_{j \in \{4,6,7,8\}} p_j = 888$. It follows that $z^h = 898$, $X^h = \{1, 3, 4, 5, 6\}$.

The solution does not change for all values $\varepsilon \geq \frac{1}{50}$. For $\varepsilon < \frac{1}{50}$, we have $\varepsilon\tilde{z}/3 < 8$, so items 1–6 are considered "large" and the algorithm finds the optimal solution using entry $T(i) = (\{1, 3, 4, 5\}, 890, 100)$. The value of q, however, is at least 22 500 instead of 36. □

Ibarra and Kim (1975) have also proposed a modified implementation having improved time complexity $O(n\log n) + O((1/\varepsilon^4)\log(1/\varepsilon))$, with the second term independent of n. Further improvements have been obtained by Lawler (1979), who used a median-finding routine (to eliminate sorting) and a more efficient scaling technique to obtain time complexity $O(n\log(1/\varepsilon) + 1/\varepsilon^4)$ and space complexity $O(n + 1/\varepsilon^3)$. Magazine and Oguz (1981) have further revised the Lawler (1979) scheme, obtaining time complexity $O(n^2\log(n/\varepsilon))$ and space complexity $O(n/\varepsilon)$.

A fully polynomial-time approximation scheme for the minimization version of KP was found, independently of the Ibarra–Kim result, by Babat (1975). Its time and space complexity of $O(n^4/\varepsilon)$ was improved to $O(n^2/\varepsilon)$ by Gens and Levner (1979).

Note that the core memory requirements of the fully polynomial-time approximation schemes depend on ε and can become impractical for small values of this parameter. On the contrary, the space complexity of Sahni's polynomial-time approximation scheme is $O(n)$, independently of r.

2.8.3 Probabilistic analysis

The first probabilistic result for KP was obtained by d'Atri (1979). Assuming that profits and weights are independently drawn from the uniform distribution over $\{1, 2, \ldots, n\}$, and the capacity from the uniform distribution over $\{1, 2, \ldots, kn\}$ (k an integer constant), he proved that there exists an $O(n)$ time algorithm giving the optimal solution with probability tending to 1 as $n \to \infty$.

Lueker (1982) investigated the properties of the average value of $(z(C(KP)) - z(KP))$ (difference between the solution value of the continuous relaxation and the optimal solution value of KP). Assuming that profits and weights are independently generated from the uniform distribution between 0 and 1 by a Poisson process with n as the expected number of items, and that the capacity is $c = \beta n$ for some constant β, he proved that:

(a) if $\beta > \frac{1}{2}$ then all items fit in the knapsack with probability tending to 1, so the question is trivial;

(b) if $\beta \leq \frac{1}{2}$ then the expected value of $(z(C(KP)) - z(KP))$ is $O(\log^2 n/n)$ and $\Omega(1/n)$.

Goldberg and Marchetti-Spaccamela (1984) improved the $\Omega(1/n)$ lower bound to $\Omega(\log^2 n/n)$, thus proving that the expected value of the difference is $\Theta(\log^2 n/n)$. In addition, they proved that, for every fixed $\varepsilon > 0$, there is a polynomial-time algorithm which finds the optimal solution to KP with probability at least $1 - \varepsilon$. (As a function of $1/\varepsilon$, the running time of the algorithm is exponential.)

Meanti, Rinnooy Kan, Stougie and Vercellis (1989) have determined, for the same probabilistic model, the expected value of the critical ratio p_s/w_s as a function of β, namely $1/\sqrt{6\beta}$ for $0 < \beta < \frac{1}{6}$, $\frac{3}{2} - 3\beta$ for $\frac{1}{6} \leq \beta < \frac{1}{2}$. The result has been used by Marchetti-Spaccamela and Vercellis (1987) to analyse the probabilistic behaviour of an *on-line* version of the greedy algorithm. (An on-line algorithm for KP is required to decide whether or not to include each item in the knapsack as it is input, i.e. as its profit and weight become known.)

The probabilistic properties of different greedy algorithms for KP have been studied in Szkatula and Libura (1987).

2.9 EXACT ALGORITHMS FOR LARGE-SIZE PROBLEMS

As will be shown in Section 2.10, many instances of KP can be solved by branch-and-bound algorithms for very high values of n. For such problems, the preliminary sorting of the items requires, on average, a comparatively high computing time (for example, when $n > 2000$ the sorting time is about 80 per cent of the total time required by the algorithm of Section 2.5.2). In the present section we examine algorithms which do not require preliminary sorting of all the items.

The first algorithm of this kind was presented by Balas and Zemel (1980) and is based on the so-called "core problem". Suppose, without loss of generality, that $p_j/w_j > p_{j+1}/w_{j+1}$ for $j = 1, \ldots, n-1$, and, for an optimal solution (x_j^*), define the *core* as

$$C = \{j_1, \ldots, j_2\},$$

where

$$j_1 = \min\{j : x_j^* = 0\}, \qquad j_2 = \max\{j : x_j^* = 1\};$$

the *core problem* is then defined as

$$\text{maximize} \quad \tilde{z} = \sum_{j \in C} p_j x_j$$

$$\text{subject to} \quad \sum_{j \in C} w_j x_j \leq c - \sum_{j \in \{i : p_i / w_i > p_{j_1} / w_{j_1}\}} w_j,$$

$$x_j = 0 \text{ or } 1, \qquad \text{for } j \in C.$$

In general, for large problems, the size of the core is a very small fraction of n. Hence, if we knew "a priori" the values of j_1 and j_2, we could easily solve the complete problem by setting $x_j^* = 1$ for all $j \in J1 = \{k : p_k/w_k > p_{j_1}/w_{j_1}\}$, $x_j^* = 0$ for all $j \in J0 = \{k : p_k/w_k < p_{j_2}/w_{j_2}\}$ and solving the core problem through any branch-and-bound algorithm (so that only the items in C would have to be sorted). Notice that $J1$ and $J0$ are conceptually close to the sets of the same name determined by reduction procedures.

Indices j_1 and j_2 cannot be "a priori" identified, but a good approximation of the core problem can be obtained if we consider that, in most cases, given the critical item s, we have $j_1 \geq s - (\vartheta/2)$ and $j_2 \leq s + (\vartheta/2)$ for some $\vartheta \ll n$.

2.9.1 The Balas–Zemel algorithm

Balas and Zemel (1980) proposed the following procedure for determining, given a prefixed value ϑ, an approximate partition $(J1, C, J0)$ of N. The methodology is very close to that used in Section 2.2.2 to determine the critical item s and the continuous solution $(\overline{x}_j)$, so we only give the statements differing from the corresponding ones in procedure CRITICAL_ITEM:

```
procedure BZC:
input: n, c, (p_j), (w_j), ϑ;
output: J1, C, (x̄_j);
begin
    ...
    while partition = "no" and |JC| > ϑ do
        begin
            determine the median r_t of the first 3 ratios p_j/w_j in JC;
            ...
        end;
    if |JC| ≤ ϑ then
        begin
            C := JC;
            sort the items in C according to decreasing p_j/w_j ratios;
            determine the critical item s and the solution (x̄_j) of the continuous
            relaxation through the Dantzig method applied to the items in C
            with the residual capacity c̄
        end
    else
        begin
            let E = {e_1, ..., e_q};
```

$\sigma := \min \{j : \sum_{i=1}^{j} w_{e_i} > \bar{c} - c'\}$;
$s := e_\sigma$;
for each $j \in J1 \cup G \cup \{e_1, \ldots, e_{\sigma-1}\}$ **do** $\bar{x}_j := 1$;
for each $j \in J0 \cup L \cup \{e_{\sigma+1}, \ldots, e_q\}$ **do** $\bar{x}_j := 0$;
$\bar{x}_s := (c - \sum_{j \in \{1,\ldots,n\} \setminus \{s\}} w_j \bar{x}_j)/w_s$;
define C as a sorted subset of JC such that $|C| = \vartheta$ and s is contained, if possible, in the middle third of C, and correspondingly enlarge set $J1$
end
end.

Determining the median of the first three ratios (instead of that of all the ratios) in JC increases the time complexity of the algorithm to $O(n^2)$, but is indicated in Balas and Zemel (1980) as the method giving the best experimental results. They had also conjectured that the expected size of the core problem is constant, and experimentally determined it as $\vartheta = 25$. The conjecture has been contradicted by Goldberg and Marchetti-Spaccamela (1984), who proved that the expected core problem size grows (very slowly) with n.

The Balas–Zemel method also makes use of a heuristic procedure H and a reduction procedure R. These can be summarized as follows:

procedure H:
input: $C, J1$;
output: $z, (x_j)$;
begin
given an approximate core problem C and a set $J1$ of items j such that x_j is fixed to 1, find an approximate solution for C by using dominance relations between the items;
define the corresponding approximate solution (x_j), and its value z, for KP
end.

procedure R:
input: C;
output: $J1', J0'$;
begin
fix as many variables of C as possible by applying the reduction test of algorithm FPDHR, then that of algorithm IKR (see Section 2.7), modified so as to compute an upper bound on the continuous solution value when the items are not sorted;
define subsets $J1'$ and $J0'$, containing the variables fixed, respectively, to 1 and to 0
end.

The Balas–Zemel idea is first to solve, without sorting, the continuous relaxation of KP, thus determining the Dantzig upper bound (see Section 2.2.1), and then searching for heuristic solutions of approximate core problems giving the upper

bound value for KP. When such attempts fail, the reduced problem is solved through an exact procedure. The algorithm can be outlined as follows (γ is a given threshold value for which Balas and Zemel used $\gamma = 50$).

procedure BZ:
input: $n, c, (p_j), (w_j), \vartheta, \gamma$;
output: $z, (x_j)$;
begin
  call BZC ;
  $z^c := \sum_{j=1}^{n} p_j \bar{x}_j$;
  call H ;
  if $z = \lfloor z^c \rfloor$ **then return**;
  $C := \{1, \ldots, n\}$;
  call R;
  $J1 := J1'$;
  $J0 := J0'$;
  $C := C \setminus (J1 \cup J0)$ (**comment**: new core);
  if $|C| > \gamma$ **then**
    begin
      call H ;
      if $z = \lfloor z^c \rfloor$ **then return**;
      call R;
      $J1 := J1 \cup J1'$;
      $J0 := J0 \cup J0'$;
      $C := C \setminus (J1' \cup J0')$ (**comment**: reduced core);
    end;
  sort the items in C according to decreasing p_j/w_j ratios;
  exactly solve the core problem through the Zoltners (1978) algorithm;
  define the corresponding values of z and (x_j) for KP
end.

Two effective algorithms for solving KP without sorting all the items have been derived from the Balas–Zemel idea by Fayard and Plateau (1982) and Martello and Toth (1988).

2.9.2 The Fayard–Plateau algorithm

The algorithm, published together with an effective Fortran implementation (see Fayard and Plateau (1982)), can be briefly described as follows.

procedure FP:
input: $n, c, (p_j), (w_j)$;
output: $z, (x_j)$;
begin
  $N := \{1, \ldots, n\}$;

use a procedure similar to CRITICAL_ ITEM (see Section 2.2.2) to determine the critical item s and the subset $J1 \subset N$ such that, in the continuous solution of KP, $x_j = 1$ for $j \in J1$;
$\overline{c} := c - \sum_{j \in J1} w_j$;
$z^c := \sum_{j \in J1} p_j + \overline{c} p_s / w_s$;
apply the greedy algorithm (without sorting) to the items in $N \backslash J1$ with the residual capacity $\overline{c}$, and let (x_j) $(j \in N \backslash J1)$ be the approximate solution found;
$z := \sum_{j \in J1} p_j + \sum_{j \in N \backslash J1} p_j x_j$;
if $z = \lfloor z^c \rfloor$ **then return** ;
apply reduction algorithm FPDHR (see Section 2.7), defining sets $J1'$ and $J0'$;
$C := N \backslash (J1' \cup J0')$ (**comment**: reduced problem);
sort the items in C according to increasing values of $|\tilde{p}_j| = |p_j - w_j p_s / w_s|$;
exactly solve the reduced problem through a specific enumerative technique;
define the corresponding values of z and (x_j) for KP
end.

2.9.3 The Martello–Toth algorithm

The Martello and Toth (1988) algorithm can be sketched as follows.

Step 1. Partition N into $J1, J0$ and C through a modification of the Balas–Zemel method. Sort the items in C.

Step 2. Exactly solve the core problem, thus obtaining an approximate solution for KP, and compute upper bound U_6 (see Section 2.3.3). If its value equals that of the approximate solution then this is clearly optimal: stop. Otherwise

Step 3. Reduce KP with no further sorting: if all variables x_j such that $j \in J1$ or $j \in J0$ are fixed (respectively to 1 and to 0), then we have it that C is the exact core, so the approximate solution of Step 2 is optimal: stop. Otherwise

Step 4. Sort the items corresponding to variables not fixed by reduction and exactly solve the corresponding problem.

The algorithm improves upon the previous works in four main respects:

(a) the approximate solution determined at Step 2 is more precise (often optimal); this is obtained through more careful definition of the approximate core and through exact (instead of heuristic) solution of the corresponding problem;

(b) there is a higher probability that such an approximate solution can be proved

to be optimal either at Step 2 (because of a tighter upper bound computation) or at Step 3 (missing in previous works);

(c) the procedures for determining the approximate core (Step 1) and reducing KP (Step 3) have been implemented more efficiently;

(d) the exact solution of the subproblems (Steps 2 and 4) has been obtained by adapting an effective branch-and-bound algorithm (procedure MT1 of Section 2.5.2).

Step 1

The procedure to determine the approximate core problem receives in input four parameters: ϑ (desired core problem size), α, β (tolerances) and η (bound on the number of iterations). It returns a partition $(J1, C, J0)$ of N, where C defines an approximate core problem having residual capacity $\bar{c} = c - \sum_{j \in J1} w_j$, such that

(i) $(1 - \alpha)\vartheta \leq |C| \leq (1 + \beta)\vartheta$,

(ii) $\sum_{j \in C} w_j > \bar{c} > 0$,

(iii) $\max \{p_k/w_k : k \in J0\} \leq p_j/w_j \leq \min \{p_k/w_k : k \in J1\}$ for all $j \in C$.

$J1$ and $J0$ are initialized to empty, and C to N. At any iteration we try to move elements from N to $J1$ or $J0$, until $|C|$ is inside the prefixed range. Following Balas and Zemel (1980), this is obtained by partitioning (through a tentative value λ) set C into three sets of items j such that p_j/w_j is less than λ (set L), equal to λ (set E) or greater than λ (set G). Three possibilities are then considered, according to the value of the current residual capacity $\bar{c}$:

(a) $\sum_{j \in G} w_j \leq \bar{c} < \sum_{j \in G \cup E} w_j$, i.e., $\lambda = p_s/w_s$: if $|E|$ is large enough, the desired core is defined; otherwise λ is increased or decreased, according to the values of $|G|$ and $|L|$, so that $|C|$ results closer to the desired size at the next iteration;

(b) $\sum_{j \in G} w_j > \bar{c}$, i.e., $\lambda < p_s/w_s$: if $|G|$ is large enough we move the elements of $L \cup E$ from C to $J0$ and increase λ; otherwise we decrease λ so that, at the next iteration, $|G|$ results larger;

(c) $\sum_{j \in G \cup E} w_j < \bar{c}$, i.e., $\lambda > p_s/w_s$: if $|L|$ is large enough we move the elements of $G \cup E$ from C to $J1$ and decrease λ; otherwise we increase λ so that, at the next iteration, $|L|$ results larger.

In the following description of procedure CORE, $M3(S)$ denotes the median of the profit/weight ratios of the first, last and middle element of S. If the desired C is not obtained within η iterations, execution is halted and the current partition $(J1, C, J0)$ is returned. In this case, however, condition (i) above is not satisfied, i.e. $|C|$ is not inside the prefixed range.

```
procedure CORE:
input: n, c, (p_j), (w_j), ϑ, α, β, η;
output: J1, C, J0;
begin
    J1 := Ø;
    J0 := Ø;
    C := {1, ..., n};
    c̄ := c;
    k := 0;
    λ := M3(C);
    while |C| > (1 + β)ϑ and k < η do
        begin
            G := {j ∈ C : p_j/w_j > λ};
            L := {j ∈ C : p_j/w_j < λ};
            E := {j ∈ C : p_j/w_j = λ};
            c' := Σ_{j∈G} w_j;
            c'' := c' + Σ_{j∈E} w_j;
            if c' ≤ c̄ < c'' then
                if |E| ≥ (1 − α)ϑ then
                    begin
                        let E = {e_1, ..., e_q};
                        σ := min {j : Σ_{i=1}^{j} w_{e_i} > c̄ − c'};
                        s := e_σ;
                        C := {e_r, ..., e_t} with r, t such that
                            t − r + 1 is as close as possible to ϑ
                            and (t + r)/2 to s;
                        J0 := J0 ∪ L ∪ {e_{t+1}, ..., e_q};
                        J1 := J1 ∪ G ∪ {e_1, ..., e_{r−1}}
                    end
                else
                    if |G ∪ E| < ϑ then λ := M3(L)
                    else λ := M3(G)
            else
                if c' > c̄ then
                    if |G| < (1 − α)ϑ then λ := M3(L)
                    else
                        begin
                            J0 := J0 ∪ L ∪ E;
                            C := G;
                            λ := M3(C)
                        end
                else
                    if |L| < (1 − α)ϑ then λ := M3(G)
                    else
                        begin
                            J1 := J1 ∪ G ∪ E;
                            C := L;
                            c̄ := c̄ − c'';
```

```
                    λ := M3(C)
                end;
            k := k + 1
        end
end.
```

The heaviest computations in the "while" loop (partitioning of C and definition of c' and c'') require $O(n)$ time. Hence, if η is a prefixed constant, the procedure runs in linear time.

Steps 2, 4

Exact solutions $(\hat{x}_j)$ of the core problem and of the reduced problem are obtained through procedure MT1 of Section 2.5.2, modified so as also to compute, if required, the value u of upper bound U_6 (Section 2.3.3) for KP. We refer to this procedure as MT1′ and call it by giving the sets C (free items) and $J1$ (items j such that x_j is fixed to 1).

```
procedure MT1′:
input: n, c, (p_j), (w_j), C, J1, bound;
output: (x̂_j), u;
begin
    define the sub-instance KP′ consisting of the items in C with residual capacity
      c − Σ_{j∈J1} w_j;
    if bound = "no" then call MT1 for KP′
    else call MT1 for KP′ with determination of u = U_6;
    let (x̂_j) be the solution vector returned by MT1
end.
```

Step 3

Reduction without sorting is obtained through the following procedure, which receives in input the partition determined at Step 1 (with only the items in C sorted according to decreasing p_j/w_j ratios) and the value z^h of the approximate solution found at Step 2. The procedure defines sets $\overline{J1}$ and $\overline{J0}$ according to the same rules as in procedure MTR (Section 2.7), but computing weaker bounds u_j^0 and u_j^1 when the current critical item $\bar{s}$ is not in C.

```
procedure MTR′:
input: n, c, (p_j), (w_j), z^h, J1, C, J0;
output: J̄1, J̄0;
begin
    comment: it is assumed that the items in C are 1, 2, ..., f (f =
                  |C|), sorted according to decreasing p_j/w_j ratios;
    c̄ := c − Σ_{j∈J1} w_j;
    p̄ := Σ_{j∈J1} p_j;
```

for $j := 1$ **to** f **do** compute $\overline{w}_j = \sum_{i=1}^{j} w_i$ and $\overline{p}_j = \sum_{i=1}^{j} p_i$;
find, through binary search, $s \in C$ such that $\overline{w}_{s-1} \leq \overline{c} < \overline{w}_s$;
for each $j \in J1 \cup \{1, \ldots, s\}$ **do**
  if $\overline{c} + w_j < \overline{w}_f$ **then**
    begin
      find, through binary search, $\overline{s} \in C$ such that
        $\overline{w}_{\overline{s}-1} \leq \overline{c} + w_j < \overline{w}_{\overline{s}}$;
      $\overline{\overline{c}} := \overline{c} + w_j - \overline{w}_{\overline{s}-1}$;
      $u_j^0 := \overline{p} - p_j + \overline{p}_{\overline{s}-1} + \max(\lfloor \overline{\overline{c}} p_{\overline{s}+1} / w_{\overline{s}+1} \rfloor,$
        $\lfloor p_{\overline{s}} - (w_{\overline{s}} - \overline{\overline{c}}) p_{\overline{s}-1} / w_{\overline{s}-1} \rfloor)$;
      $z^h := \max(z^h, \overline{p} - p_j + \overline{p}_{\overline{s}-1})$
    end
  else
    begin
      $u_j^0 := \overline{p} - p_j + \overline{p}_f + \lfloor (\overline{c} + w_j - \overline{w}_f) p_f / w_f \rfloor$;
      $z^h := \max(z^h, \overline{p} - p_j + \overline{p}_f)$
    end;
for each $j \in J0 \cup \{s, \ldots, f\}$ **do**
  if $\overline{c} - w_j \geq \overline{w}_1$ **then**
    begin
      find, through binary search, $\overline{s} \in C$ such that
        $\overline{w}_{\overline{s}-1} \leq \overline{c} - w_j < \overline{w}_{\overline{s}}$;
      $\overline{\overline{c}} := \overline{c} - w_j - \overline{w}_{\overline{s}-1}$;
      $u_j^1 := \overline{p} + p_j + \overline{p}_{\overline{s}-1} + \max(\lfloor \overline{\overline{c}} p_{\overline{s}+1} / w_{\overline{s}+1} \rfloor,$
        $\lfloor p_{\overline{s}} - (w_{\overline{s}} - \overline{\overline{c}}) p_{\overline{s}-1} / w_{\overline{s}-1} \rfloor)$;
      $z^h := \max(z^h, \overline{p} + p_j + \overline{p}_{\overline{s}-1})$
    end
  else
    begin
      $u_j^1 := \lfloor \overline{p} + p_j + (\overline{c} - w_j) p_1 / w_1 \rfloor$;
      if $\overline{c} - w_j \geq 0$ **then** $z^h := \max(z^h, \overline{p} + p_j)$
    end;
$\overline{J0} := \{j \in J0 \cup \{s, \ldots, f\} : u_j^1 \leq z^h\}$;
$\overline{J1} := \{j \in J1 \cup \{1, \ldots, s\} : u_j^0 \leq z^h\}$
end.

The heaviest computations are involved in the two "for each" loops: for $O(n)$ times a binary search, of time complexity $O(\log|C|)$, is performed. The overall time complexity is thus $O(n \log|C|)$, i.e. $O(n)$ for fixed $|C|$.

Algorithm

The algorithm exactly solves KP through Steps 1–4, unless the size of core C determined at Step 1 is too large. If this is the case, the solution is obtained

through standard sorting, reduction and branch-and-bound. On input, the items are not assumed to be sorted.

procedure MT2:
input: $n, c, (p_j), (w_j), \vartheta, \alpha, \beta, \eta$;
output: $z, (x_j)$;
begin
  for $j := 1$ **to** n **do** $x_j := 0$;
  comment: Step 1;
  call CORE;
  if $|C| \leq (1-\alpha)n$ **then**
    begin
      sort the items in C by decreasing p_j/w_j ratios;
      comment: Step 2;
      bound := "yes";
      call MT1′;
      $z^h := \sum_{j \in J1} p_j + \sum_{j \in C} p_j \hat{x}_j$;
      if $z^h = u$ **then**
        for each $j \in J1 \cup \{k \in C : \hat{x}_k = 1\}$ **do** $x_j := 1$
      else (**comment**: Step 3)
        begin
          call MTR′;
          if $\overline{J1} \supseteq J1$ and $\overline{J0} \supseteq J0$ **then**
            for each $j \in J1 \cup \{k \in C : \hat{x}_k = 1\}$ **do** $x_j := 1$
          else (**comment**: Step 4)
            begin
              $C := \{1, \ldots, n\} \backslash (\overline{J1} \cup \overline{J0})$;
              sort the items in C according to
                decreasing p_j/w_j ratios;
              bound := "no";
              $J1 := \overline{J1}$;
              call MT1′;
              for each $j \in \overline{J1} \cup \{k \in C : \hat{x}_k = 1\}$ **do** $x_j := 1$
            end
        end
    end
  else (**comment**: standard solution)
    begin
      sort all the items according to decreasing p_j/w_j ratios;
      call MTR;
      $z^h := l$;
      $C := \{1, \ldots, n\} \backslash (J1 \cup J0)$;
      bound:= "no";
      call MT1′;
      for each $j \in J1 \cup \{k \in C : \hat{x}_k = 1\}$ **do** $x_j := 1$
    end;

```
    z := Σ_{j=1}^{n} p_j x_j;
    if z < z^h then
        begin
            define the solution vector (x_j) corresponding to z^h;
            z := z^h
        end
end.
```

On the basis of the computational experiments reported in the next section, the four parameters needed by MT2 have been determined as

$$\vartheta = \begin{cases} n & \text{if } n < 200, \\ 2\sqrt{n} & \text{otherwise;} \end{cases}$$

$$\alpha = 0.2;$$

$$\beta = 1.0;$$

$$\eta = 20.$$

The Fortran implementation of MT2 is included in the present volume.

2.10 COMPUTATIONAL EXPERIMENTS

In this section we analyse the experimental behaviour of exact and approximate algorithms for KP on sets of randomly generated test problems. Since the difficulty of such problems is greatly affected by the correlation between profits and weights, we consider three randomly generated data sets:

uncorrelated: p_j and w_j uniformly random in $[1, v]$;

weakly correlated: w_j uniformly random in $[1, v]$,
p_j uniformly random in $[w_j - r, w_j + r]$;

strongly correlated: w_j uniformly random in $[1, v]$,
$p_j = w_j + r$.

Increasing correlation means decreasing value of the difference $\max_j\{p_j/w_j\} - \min_j\{p_j/w_j\}$, hence increasing expected difficulty of the corresponding problems. According to our experience, weakly correlated problems are closer to real world situations.

For each data set we consider two values of the capacity: $c = 2v$ and $c = 0.5\sum_{j=1}^{n} w_j$. In the first case the optimal solution contains very few items, so the generated instances are expected to be easier than in the second case, where about half of the items are in the optimal solution. (Further increasing the value of c does not significantly increase the computing times.)

2.10.1 Exact algorithms

We give separate tables for small-size problems ($n \leq 200$) and large-size problems ($n \geq 500$).

We compare the Fortran IV implementations of the following algorithms:

HS = Horowitz and Sahni (1974), Section 2.5.1;

MTR+HS = HS preceded by reduction procedure MTR of Section 2.7;

NA = Nauss (1976), with its own reduction procedure;

MT1 = Martello and Toth (1977a), Section 2.5.2;

MTR+MT1 = Martello and Toth (1977a) preceded by MTR;

MTR+DPT = Toth (1980), Section 2.6.3, preceded by MTR;

BZ = Balas and Zemel (1980), Section 2.9.1, with its own reduction procedure;

FP = Fayard and Plateau (1982), Section 2.9.2, with its own reduction procedure;

MT2 = Martello and Toth (1988), Section 2.9.3, with MTR and MTR′.

NA, MT1, FP and MT2 are published codes, whose characteristics are given in Table 2.1. HS, MTR and DPT have been coded by us. For BZ we give the computing times presented by the authors.

Table 2.1 Fortran codes for KP

Authors	Core memory	Number of statements	List
Nauss (1976)	$8n$	280	Available from the author
Martello and Toth (1977a)	$8n$	280	This volume (also in Martello and Toth (1978))
Fayard and Plateau (1982)	$7n$	600	In Fayard and Plateau (1982)
Martello and Toth (1988)	$8n$	1400	This volume

All runs (except those of Table 2.8) were executed on a CDC-Cyber 730. For each data set, value of c and value of n, the tables give the average running time, expressed in seconds, computed over 20 problem instances. Since Balas and Zemel (1980) give times obtained on a CDC-6600, which we verified to be at least two times faster than the CDC-Cyber 730 on problems of this kind, the times given in the tables for BZ are those reported by the authors multiplied by 2.

Code FP includes its own sorting procedure. The sortings needed by HS, NA, MT1, DPT and MT2 were obtained through a subroutine (included in MT2), derived

Table 2.2 Sorting times. CDC-Cyber 730 in seconds. Average times over 20 problems

n	50	100	200	500	1000	2000	5000	10 000
time	0.008	0.018	0.041	0.114	0.250	0.529	1.416	3.010

Table 2.3 Uncorrelated problems: p_j and w_j uniformly random in [1,100]. CDC-Cyber 730 in seconds. Average times over 20 problems

c	n	HS	MTR +HS	NA	MT1	MTR +MT1	FP	MTR +DPT
200	50	0.022	0.013	0.015	0.015	0.012	0.013	0.013
	100	0.039	0.024	0.025	0.026	0.025	0.018	0.029
	200	0.081	0.050	0.055	0.051	0.050	0.032	0.055
$0.5\sum_{j=1}^{n} w_j$	50	0.031	0.016	0.015	0.016	0.013	0.013	0.020
	100	0.075	0.028	0.029	0.030	0.026	0.021	0.043
	200	0.237	0.065	0.073	0.068	0.057	0.053	0.090

Table 2.4 Weakly correlated problems: w_j uniformly random in [1,100], p_j in $[w_j-10, w_j+10]$. CDC-Cyber 730 in seconds. Average times over 20 problems

c	n	HS	MTR +HS	NA	MT1	MTR +MT1	FP	MTR +DPT
200	50	0.031	0.018	0.019	0.017	0.014	0.016	0.022
	100	0.049	0.029	0.038	0.032	0.024	0.023	0.041
	200	0.091	0.052	0.060	0.055	0.048	0.030	0.066
$0.5\sum_{j=1}^{n} w_j$	50	0.038	0.025	0.035	0.022	0.020	0.021	0.071
	100	0.079	0.042	0.086	0.040	0.031	0.039	0.158
	200	0.185	0.070	0.151	0.069	0.055	0.057	0.223

Table 2.5 Strongly correlated problems: w_j uniformly random in [1,100], $p_j = w_j + 10$. CDC-Cyber 730 in seconds. Average times over 20 problems

c	n	HS	MTR +HS	NA	MT1	MTR +MT1	FP	MTR +DPT
200	50	0.165	0.101	0.117	0.028	0.025	0.047	0.041
	100	1.035	0.392	0.259	0.052	0.047	0.096	0.070
	200	3.584	2.785	3.595	0.367	0.311	0.928	0.111
$0.5\sum_{j=1}^{n} w_j$	50	time	time	time	4.870	4.019	17.895	0.370
	100	—	—	—	time	time	time	1.409
	200	—	—	—	—	—	—	3.936

from subroutine SORTZV of the CERN Library, whose experimental behaviour is given in Table 2.2. All the times in the following tables include sorting and reduction times.

Tables 2.3, 2.4 and 2.5 compare algorithms HS, MTR+HS, NA, MT1, MTR+MT1, FP and MTR+DPT on small-size problems (we do not give the times of MT2, which are almost equal to those of MTR+MT1). For all data sets, $v = 100$ and $r = 10$. Table 2.3 refers to uncorrelated problems, Table 2.4 to weakly correlated problems. All algorithms solved the problems very quickly with the exception of HS and, for weakly correlated problems, MTR+DPT. MT1 is only slightly improved by previous application of MTR, contrary to what happens for HS. Table 2.5 refers to strongly correlated problems. Because of the high times generally involved, a time limit of 500 seconds was assigned to each algorithm for solution of the 60 problems generated for each value of c. The dynamic programming approach appears clearly superior to all branch-and-bound algorithms (among which MT1 has the best performance).

For large-size instances we do not consider strongly correlated problems, because of the impractical times involved. Tables 2.6 and 2.7 compare algorithms MT1, BZ, FP and MT2. Dynamic programming is not considered because of excessive memory requirements, HS and NA because of clear inferiority. The problems were generated with $v = 1\,000$, $r = 100$ and $c = 0.5 \sum_{j=1}^{n} w_j$.

FP is fast for $n \leq 2\,000$ but very slow for $n \geq 5\,000$, while BZ has the opposite behaviour. MT2 has about the same times as FP for $n \leq 2\,000$, the same times as BZ for $n = 5\,000$, and slightly higher than BZ for $n = 10\,000$, so it can be considered, on average, the best code. MT1, which is not designed for large

Table 2.6 Uncorrelated problems: p_j and w_j uniformly random in [1,1000]; $c = 0.5 \sum_{j=1}^{n} w_j$. CDC-Cyber 730 in seconds. Average times over 20 problems

n	MT1	BZ	FP	MT2
500	0.199	—	0.104	0.157
1 000	0.381	0.372	0.188	0.258
2 000	0.787	0.606	0.358	0.462
5 000	1.993	0.958	1.745	0.982
10 000	4.265	1.514	7.661	1.979

Table 2.7 Weakly correlated problems: w_j uniformly random in [1,1000], p_j in $[w_j - 100, w_j + 100]$; $c = 0.5 \sum_{j=1}^{n} w_j$. CDC-Cyber 730 in seconds. Average times over 20 problems

n	MT1	BZ	FP	MT2
500	0.367	—	0.185	0.209
1 000	0.663	0.588	0.271	0.293
2 000	1.080	0.586	0.404	0.491
5 000	2.188	0.744	1.782	0.771
10 000	3.856	1.018	19.481	1.608

Table 2.8 Algorithm MT2. w_j uniformly random in [1,1000]; $c = 0.5 \sum_{j=1}^{n} w_j$. HP 9000/840 in seconds. Average times over 20 problems

n	Uncorrelated problems: p_j uniformly random in [1,1000]	Weakly correlated problems: p_j uniformly random in $[w_j - 100, w_j + 100]$
50	0.008	0.015
100	0.016	0.038
200	0.025	0.070
500	0.067	0.076
1 000	0.122	0.160
2 000	0.220	0.260
5 000	0.515	0.414
10 000	0.872	0.739
20 000	1.507	1.330
30 000	2.222	3.474
40 000	2.835	2.664
50 000	3.562	3.492
60 000	4.185	504.935
70 000	4.731	4.644
80 000	5.176	5.515
90 000	5.723	6.108
100 000	7.001	7.046
150 000	9.739	time limit
200 000	14.372	—
250 000	17.135	—

problems, is generally the worst algorithm. However, about 80 per cent of its time is spent in sorting, so its use can be convenient when several problems are to be solved for the same item set and different values of c. A situation of this kind arises for multiple knapsack problems, as will be seen in Section 6.4.

$n = 10\,000$ is the highest value obtainable with the CDC-Cyber 730 computer available at the University of Bologna, because of a core memory limitation of 100 Kwords. Hence, we experimented the computational behaviour of MT2 for higher values of n on an HP 9000/840 with 10 Mbytes available. We used the Fortran compiler with option "-o", producing an object with no special optimization. The results obtained for uncorrelated and weakly correlated problems are shown in Table 2.8. Uncorrelated problems were solved up to $n = 250\,000$ with very regular average times, growing less than linearly with n. Weakly correlated problems show an almost linear growing rate, but less regularity; for high values of n, certain instances required extremely high times (for $n = 60\,000$ one of the instances took almost 3 hours CPU time, for $n = 150\,000$ execution was halted after 4 hours).

2.10.2 Approximate algorithms

In Tables 2.9–2.11 we experimentally compare the polynomial-time approximation scheme of Sahni (Section 2.8.1) and a heuristic version of algorithm MT2

Table 2.9 Uncorrelated problems: p_j and w_j uniformly random in [1,1000]; $c = 0.5 \sum_{j=1}^{n} w_j$. HP 9000/840 in seconds. Average times (average percentage errors) over 20 problems

n	MT2 approx. time (% error)	S(0) time (% error)	S(1) time (% error)	S(2) time (% error)
50	0.004(0.10569)	0.005(5.36560)	0.017(5.13968)	0.319(5.05006)
100	0.009(0.05345)	0.009(2.25800)	0.060(2.21412)	2.454(2.19447)
200	0.015(0.03294)	0.017(1.15739)	0.210(1.12217)	19.376(1.11691)
500	0.029(0.00767)	0.049(0.49120)	1.242(0.47978)	299.593(0.47577)
1 000	0.058(0.00418)	0.105(0.21213)	4.894(0.20748)	—
2 000	0.117(0.00251)	0.224(0.10531)	19.545(0.10338)	—
5 000	0.296(0.00182)	0.618(0.05540)	125.510(0.05488)	—
10 000	0.641(0.00076)	1.320(0.02045)	—	—
20 000	1.248(0.00032)	2.852(0.00897)	—	—
30 000	1.873(0.00016)	4.363(0.00786)	—	—
40 000	2.696(0.00016)	6.472(0.00521)	—	—
50 000	3.399(0.00011)	8.071(0.00428)	—	—
60 000	3.993(0.00009)	9.778(0.00403)	—	—
70 000	4.652(0.00003)	11.420(0.00301)	—	—
80 000	5.307(0.00008)	13.075(0.00329)	—	—
90 000	5.842(0.00016)	14.658(0.00247)	—	—
100 000	6.865(0.00007)	16.347(0.00231)	—	—
150 000	9.592(0.00005)	25.357(0.00156)	—	—
200 000	13.223(0.00008)	35.050(0.00144)	—	—
250 000	16.688(0.00010)	44.725(0.00094)	—	—

(Section 2.9.3). The fully polynomial-time approximation schemes are not included since a limited series of experiments showed a dramatic inferiority of these algorithms (see also Section 4.4.2, where this trend is confirmed for the subset-sum problem).

The heuristic version of MT2 was obtained by halting execution at the end of Step 2, and returning the approximate solution of value z^h. In order to obtain a small core problem, procedure CORE was executed with parameters

$$\vartheta = 5;$$

$$\alpha = 0.0;$$

$$\beta = 1.0;$$

$$\eta = 200.$$

As for the Sahni scheme S(k), we experimented S(0), S(1) and S(2), since the time complexity $O(n^{k+1})$ makes the algorithm impractical for $k \geq 3$.

Tables 2.9, 2.10 and 2.11 give the results for the three data sets, with $v = 1\,000$, $r = 100$ and $c = 0.5 \sum_{j=1}^{n} w_j$. For each approximate algorithm, we give (in brackets) the average percentage error. This was computed as $100(z - z^a)/z$, where z^a is the approximate solution value and z either the optimal solution value (when

Table 2.10 Weakly correlated problems: w_j uniformly random in [1,1000], p_j in $[w_j - 100, w_j + 100]$; $c = 0.5 \sum_{j=1}^{n} w_j$. HP 9000/840 in seconds. Average times (average percentage errors) over 20 problems

n	MT2 approx. time (% error)	S(0) time (% error)	S(1) time (% error)	S(2) time (% error)
50	0.006(0.17208)	0.004(2.13512)	0.017(1.81004)	0.302(1.77572)
100	0.008(0.04296)	0.008(0.87730)	0.055(0.78573)	2.281(0.76862)
200	0.013(0.06922)	0.015(0.31819)	0.194(0.28838)	17.779(0.28216)
500	0.033(0.01174)	0.046(0.14959)	1.139(0.14300)	273.118(0.14135)
1 000	0.058(0.00774)	0.103(0.08226)	4.432(0.07842)	—
2 000	0.114(0.00589)	0.222(0.03740)	17.626(0.03634)	—
5 000	0.312(0.00407)	0.619(0.01445)	113.527(0.01413)	—
10 000	0.645(0.00261)	1.324(0.00630)	—	—
20 000	1.297(0.00155)	2.802(0.00312)	—	—
30 000	1.943(0.00104)	4.372(0.00216)	—	—
40 000	2.667(0.00052)	6.432(0.00177)	—	—
50 000	3.374(0.00036)	8.013(0.00139)	—	—
60 000	4.544(0.00028)	9.377(0.00095)	—	—
70 000	4.662(0.00040)	11.069(0.00083)	—	—
80 000	6.029(0.00031)	13.041(0.00070)	—	—
90 000	6.249(0.00040)	15.662(0.00071)	—	—
100 000	6.618(0.00017)	16.358(0.00050)	—	—
150 000	10.231(0.00019)	25.530(0.00041)	—	—
200 000	12.991(0.00004)	35.230(0.00027)	—	—
250 000	16.062(0.00009)	45.234(0.00020)	—	—

Table 2.11 Strongly correlated problems: w_j uniformly random in [1,1000], $p_j = w_j + 100$; $c = 0.5 \sum_{j=1}^{n} w_j$. HP 9000/840 in seconds. Average times (average percentage errors) over 20 problems

n	MT2 approx. time (% error)	S(0) time (% error)	S(1) time (% error)	S(2) time (% error)
50	0.008(1.50585)	0.003(3.25234)	0.019(1.68977)	0.340(0.74661)
100	0.008(0.81601)	0.007(1.43595)	0.061(0.73186)	2.574(0.39229)
200	0.015(0.51026)	0.017(0.77478)	0.226(0.40653)	20.877(0.26096)
500	0.029(0.27305)	0.046(0.33453)	1.372(0.17836)	316.804(0.09783)
1 000	0.059(0.10765)	0.111(0.15991)	5.388(0.08409)	—
2 000	0.119(0.06850)	0.236(0.08866)	21.173(0.05196)	—
5 000	0.315(0.02148)	0.614(0.02740)	132.973(0.01421)	—
10 000	0.679(0.01384)	1.341(0.01573)	—	—
20 000	1.266(0.00559)	2.787(0.00694)	—	—
30 000	1.879(0.00512)	4.333(0.00504)	—	—
40 000	2.603(0.00292)	6.022(0.00372)	—	—
50 000	3.182(0.00240)	7.598(0.00239)	—	—
60 000	3.795(0.00224)	9.194(0.00252)	—	—
70 000	4.529(0.00167)	10.760(0.00214)	—	—
80 000	5.090(0.00154)	12.324(0.00185)	—	—
90 000	5.595(0.00115)	13.968(0.00179)	—	—
100 000	6.320(0.00132)	15.569(0.00165)	—	—
150 000	9.141(0.00083)	24.583(0.00082)	—	—
200 000	12.005(0.00077)	34.400(0.00083)	—	—
250 000	15.950(0.00055)	44.001(0.00044)	—	—

available) or an upper bound determined by the approximate version of MT2. The execution of each approximate algorithm was halted as soon as the average computing time exceeded 100 seconds.

Table 2.9 shows that it is not convenient to heuristically solve uncorrelated problems, since the exact version of MT2 requires about the same times as its approximate version, which in turn dominates S(k). The same consideration holds for weakly correlated problems with $n \leq 50\,000$ (Table 2.10); for $n > 50\,000$, the approximate version of MT2 dominates S(0), while S(1) and S(2) have impractical time requirements. Table 2.11 shows that the approximate version of MT2; which dominates S(0), must be recommended for large-size strongly correlated problems; for small values of n, S(1) and S(2) can produce better approximations but require dramatically higher computing times.

The Fortran code corresponding to MT2, included in the volume, allows use either of the exact or the approximate version through an input parameter.

2.11 FACETS OF THE KNAPSACK POLYTOPE

In this section we give an outline of the main results obtained in the study of the knapsack polytope. Since such results did not lead, up to now, to the design of effective algorithms for KP, the purpose of the present section is only to introduce the reader to the principal polyhedral concepts and to indicate the relevant literature concerning knapsack problems. Detailed introductions to the theory of polyhedra can be found in Bachem and Grötschel (1982), Pulleyblank (1983), Schrijver (1986) and Nemhauser and Wolsey (1988), among others.

We start with some basic definitions. Given a vector $a \in \mathbb{R}^n$ and a scalar $a_0 \in \mathbb{R}$, the set $\{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_j x_j = a_0\}$ is called a *hyperplane*. A hyperplane defines two *halfspaces*, namely $\{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_j x_j \leq a_0\}$ and $\{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_j x_j \geq a_0\}$. The intersection of finitely many halfspaces, when it is bounded and non-empty, is called a *polytope*. Hence, polytopes can be written as $P = \{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_{ij} x_j \leq a_{i0} \quad \text{for } i = 1, \ldots, r\}$; alternatively, they can be described as the convex hull of finitely many points, i.e. $P = \text{conv}(S)$, with $S \subset \mathbb{R}^n$ and $|S|$ finite. m points $x^1, \ldots, x^m \in \mathbb{R}^n$ are called *affinely independent* if the equations $\sum_{k=1}^{m} \lambda_k x^k = 0$ and $\sum_{k=1}^{m} \lambda_k = 0$ imply $\lambda_k = 0$ for $k = 1, \ldots, m$. The *dimension* of a polytope $P \subset \mathbb{R}^n$, dim (P), is $|\overline{P}| - 1$, where $\overline{P}$ is the largest subset of affinely independent points of P. A subset F of a polytope $P \subset \mathbb{R}^n$ is called a *face* of P if there exists an inequality $\sum_{j=1}^{n} a_j x_j \leq a_0$ which is satisfied by any $x \in P$ and such that $F = \{x \in P : \sum_{j=1}^{n} a_j x_j = a_0\}$. In other words, a face is the intersection of the polytope and a hyperplane defining a halfspace containing the polytope itself. A face F of P such that dim (F) = dim (P) -1 is called a *facet* of P. Hence an inequality $\sum_{j=1}^{n} a_j x_j \leq a_0$ defines a facet of P if (a) it is satisfied by any $x \in P$, and (b) it is satisfied with equality by exactly dim (P) affinely independent $x \in P$. The set of inequalities defining all the distinct facets of a polytope P

constitutes the minimal inequality representation of P. Hence the importance of facets in order to apply linear programming techniques to combinatorial problems.

Coming to KP, its constraint set (conditions (2.2), (2.3)) defines the *knapsack polytope*

$$K = \text{conv}\left\{x \in \mathbb{R}^n : \sum_{j=1}^{n} w_j x_j \leq c, \quad x_j \in \{0,1\} \quad \text{for } j = 1, \ldots, n\right\}.$$

It is easy to verify that, with assumption (2.6) ($w_j \leq c$ for all j),

$$\dim(K) = n.$$

In fact (a) $\dim(K) \leq n$ (obvious), and (b) $\dim(K) \geq n$, since K contains the $n+1$ affinely independent points x^k $(k = 0, \ldots, n)$, where $x^0 = (0, \ldots, 0)$ and x^k corresponds to unit vector e_k $(k = 1, \ldots, n)$. The two main classes of facets of K are based on *minimal covers* and *(1,k)-configurations.*

A set $S \subset N = \{1, \ldots, n\}$ is called a *cover* for K if

$$\sum_{j \in S} w_j > c.$$

A cover is called *minimal* if

$$\sum_{j \in S \setminus \{i\}} w_j \leq c \quad \text{for any } i \in S.$$

The set $E(S) = S \cup S'$, where

$$S' = \{j \in N \setminus S \ : \ w_j \geq \max_{i \in S} \{w_i\}\},$$

is called the *extension* of S to N. Let $\mathcal{S}$ be the family of all minimal covers S for K. Balas and Jeroslow (1972) have shown that constraints (2.2), (2.3) are equivalent to the set of *canonical inequalities*

$$\sum_{j \in E(S)} x_j \leq |S| - 1 \quad \text{for all } S \in \mathcal{S}, \tag{2.46}$$

in the sense that $x \in \{0,1\}^n$ satisfies (2.2), (2.3) if and only if it satisfies (2.46). Balas (1975), Hammer, Johnson and Peled (1975) and Wolsey (1975) have given necessary and sufficient conditions for a canonical inequality to be a facet of K.

A rich family of facets of K can be obtained by "lifting" facets of lower dimensional polytopes. Given a minimal cover S for K, let $K_S \subset \mathbb{R}^{|S|}$ denote the $|S|$-dimensional polytope

$$K_S = \text{conv}\left\{x \in \{0,1\}^{|S|} : \sum_{j \in S} w_j x_j \leq c\right\}, \tag{2.47}$$

i.e. the subset of K containing only points x such that $x_j = 0$ for all $j \in N \backslash S$. It is known (see, for instance, Balas (1975), Padberg (1975), Wolsey (1975)) that the inequality

$$\sum_{j \in S} x_j \leq |S| - 1$$

defines a facet of the lower dimensional polytope K_S. Nemhauser and Trotter (1974) and Padberg (1975) have given a *sequential lifting* procedure to determine integer coefficients β_j $(j \in N \backslash S)$ such that the inequality

$$\sum_{j \in S} x_j + \sum_{j \in N \backslash S} \beta_j x_j \leq |S| - 1$$

defines a facet of K. Calculating these coefficients requires solution of a sequence of $|N \backslash S|$ 0-1 knapsack problems. Furthermore, the facet obtained depends on the sequence in which indices $j \in N \backslash S$ are considered. Zemel (1978) and Balas and Zemel (1978) have given a characterization of the entire class of facets associated with minimal covers, and a *simultaneous lifting* procedure to obtain them. These facets have in general fractional coefficients (those with integer coefficients coincide with the facets produced by sequential lifting).

A richer class of facetial inequalities of K is given by $(1,k)$-configurations (Padberg, 1979, 1980). Given a subset $M \subset N$ and $t \in N \backslash M$, define the set $S = M \cup \{t\}$. S is a $(1,k)$-*configuration* for K if (a) $\sum_{j \in M} w_j \leq c$ and (b) $Q \cup \{t\}$ is a minimal cover for every $Q \subseteq M$ with $|Q| = k$, where k is any given integer satisfying $2 \leq k \leq |M|$. Note that if $k = |M|$, a $(1,k)$-configuration is a minimal cover for K (and, conversely, any minimal cover S can be expressed as a $(1,k)$-configuration, with $k = |S| - 1$, for any $t \in S$). Padberg (1980) proved that, given a $(1,k)$-configuration $S = M \cup \{t\}$ of K, the complete and irredundant set of facets of the lower dimensional polytope K_S (see 2.47) is given by the inequalities

$$(r - k + 1)x_t + \sum_{j \in S(r)} x_j \leq r,$$

where $S(r) \subseteq M$ is any subset of cardinality r, and r is any integer satisfying $k \leq r \leq |M|$. Sequential or simultaneous lifting procedures can then be used to obtain facets of the knapsack polytope K.

Recently, Gottlieb and Rao (1988) have studied a class of facets of K, containing fractional coefficients, which can be derived from disjoint and overlapping minimal covers and $(1,k)$-configurations. For such class, they have given necessary and sufficient conditions which can easily be verified without use of the computationally

heavy simultaneous lifting procedures. The computational complexity of lifted inequalities has been analysed by Hartvigsen and Zemel (1987) and Zemel (1988).

2.12 THE MULTIPLE-CHOICE KNAPSACK PROBLEM

The *Multiple-Choice Knapsack Problem* (MCKP), also known as the *Knapsack Problem with Generalized Upper Bound (GUB) Constraints*, is a 0-1 knapsack problem in which a partition $N_1, \ldots, N_r$ of the item set N is given, and it is required that exactly one item per subset is selected. Formally,

$$\text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{2.48}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c, \tag{2.49}$$

$$\sum_{j \in N_k} x_j = 1, \quad k = 1, \ldots, r, \tag{2.50}$$

$$x_j = 0 \text{ or } 1, \quad j \in N = \{1, \ldots, n\} = \bigcup_{k=1}^{r} N_k, \tag{2.51}$$

assuming

$$N_h \cap N_k = \emptyset \quad \text{for all } h \neq k.$$

The problem is NP-hard, since any instance of KP, having r elements of profit p_j and weight w_j $(j = 1, \ldots, r)$ and capacity c, is equivalent to the instance of MCKP obtained by setting $n = 2r$, $p_j = w_j = 0$ for $j = r+1, \ldots, 2r$ and $N_k = \{k, r+k\}$ for $k = 1, \ldots, r$.

MCKP can be solved in pseudo-polynomial time through dynamic programming as follows. Given a pair of integers l $(1 \leq l \leq r)$ and $\hat{c}$ $(0 \leq \hat{c} \leq c)$, consider the sub-instance of MCKP consisting of subsets $N_1, \ldots, N_l$ and capacity $\hat{c}$. Let $f_l(\hat{c})$ denote its optimal solution value, i.e.

$$f_l(\hat{c}) = \max \left\{ \sum_{j \in \hat{N}} p_j x_j : \sum_{j \in \hat{N}} w_j x_j \leq \hat{c}, \sum_{j \in N_k} x_j = 1 \text{ for } k = 1, \ldots, l, \right.$$

$$\left. x_j = 0 \text{ or } 1 \text{ for } j \in \hat{N} \right\},$$

where $\hat{N} = \bigcup_{k=1}^{l} N_k$, and assume that $f_l(\hat{c}) = -\infty$ if the sub-instance has no feasible solution. Let

$$\overline{w}_k = \min\{w_j : j \in N_k\} \quad \text{for } k = 1, \ldots, r;$$

clearly,

$$f_1(\hat{c}) = \begin{cases} -\infty & \text{for } \hat{c} = 0, \ldots, \overline{w}_1 - 1; \\ \max\{p_j : j \in N_1, w_j \leq \hat{c}\} & \text{for } \hat{c} = \overline{w}_1, \ldots, c; \end{cases}$$

for $l = 2, \ldots, r$ we then have

$$f_l(\hat{c}) = \begin{cases} -\infty & \text{for } \hat{c} = 0, \ldots, \sum_{k=1}^{l} \overline{w}_k - 1; \\ \max\{f_{l-1}(\hat{c} - w_j) + p_j : j \in N_l, w_j \leq \hat{c}\} & \\ & \text{for } \hat{c} = \sum_{k=1}^{l} \overline{w}_k, \ldots, c. \end{cases}$$

The optimal solution is the state corresponding to $f_r(c)$. If we have $\sum_{k=1}^{r} \overline{w}_k > c$ then the instance has no feasible solution, and we obtain $f_r(c) = -\infty$. For each value of l, the above computation requires $O(|N_l|c)$ operations, so the overall time complexity of the method is $O(nc)$.

The execution of any algorithm for MCKP can be conveniently preceded by a reduction phase, using the following

Dominance Criterion 2.1. *For any $N_k (k = 1, \ldots, r)$, if there exist two items $i, j \in N_k$ such that*

$$p_i \leq p_j \quad \text{and} \quad w_i \geq w_j$$

then there exists an optimal solution to MCKP in which $x_i = 0$, i.e. item i is dominated.

Proof. Obvious from (2.50). □

As is the case for KP, dynamic programming can solve only instances of limited size. Larger instances are generally solved through branch-and-bound algorithms, based on the exact solution of the continuous relaxation of the problem, $C(MCKP)$, defined by (2.48)–(2.50) and

$$0 \leq x_j \leq 1, \quad j \in N. \tag{2.52}$$

An instance of $C(MCKP)$ can be further reduced through the following

Dominance Criterion 2.2. *For any $N_k (k = 1, \ldots, r)$, if there exist three items $h, i, j \in N_k$ such that*

$$w_h < w_i < w_j \quad \text{and} \quad \frac{p_i - p_h}{w_i - w_h} \leq \frac{p_j - p_i}{w_j - w_i} \tag{2.53}$$

then there exists an optimal solution to $C(MCKP)$ *in which* $x_i = 0$, *i.e. item* i *is dominated.*

We do not give a formal proof of this criterion. However, it can be intuitively verified by representing the items of N_k as in Figure 2.8 and observing that

(i) after application of Dominance Criterion 2.1, the remaining items can only correspond to points in the shaded triangles;

(ii) for $C(MCKP)$, all points i of each triangle are dominated by the pair of vertices h, j (since for any value $x_i \neq 0$, there can be found a combination of values x_h, x_j producing a higher profit).

Hence

(iii) after application of Dominance Criterion 2.2, only those items remain which

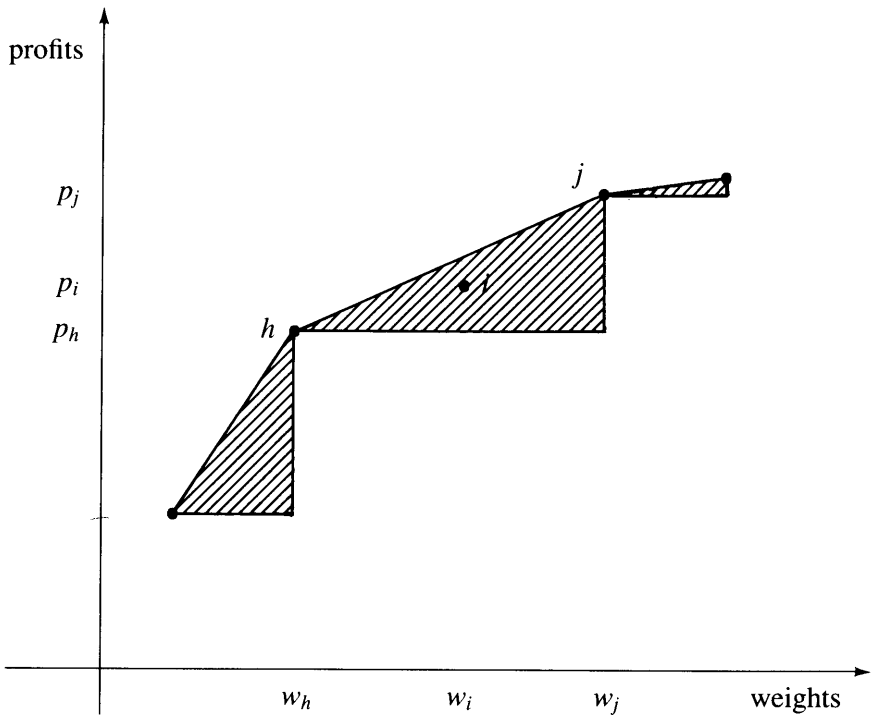

Figure 2.8 Representation of items for Dominance Criteria 2.1 and 2.2

correspond to the vertices defining the segments of the piecewise (concave) linear function.

In addition, by analysing the structure of the Linear Program corresponding to $C(MCKP)$, it is not difficult to see that

(iv) in the optimal solution of $C(MCKP)$, $r-1$ variables (corresponding to items in $r-1$ different subsets) have value 1; for the remaining subset, either one variable has value 1 or two variables (corresponding to consecutive vertices in Figure 2.8) have a fractional value.

Formal proofs of all the above properties can be found, e.g., in Sinha and Zoltners (1979).

As previously mentioned, the reduction and optimal solution of $C(MCKP)$ play a central role in all branch-and-bound algorithms for MCKP.

The reduction, based on Dominance Criteria 2.1 and 2.2, is obtained (see, e.g., Sinha and Zoltners, (1979)) by sorting the items in each subset according to increasing weights and then applying the criteria. The time complexity for this phase is clearly $O(\sum_{k=1}^{r} |N_k| \log |N_k|)$, i.e. $O(n \log \max\{|N_k| : 1 \leq k \leq r\})$.

$O(n \log r)$ algorithms for the solution of the reduced $C(MCKP)$ instance have been presented by Sinha and Zoltners (1979) and Glover and Klingman (1979). Zemel (1980) has improved the time complexity for this second phase to $O(n)$. A further improvement has been obtained by Dudzinski and Walukiewicz (1984b), who have presented an $O(r \log^2(n/r))$ algorithm.

The reduction phase is clearly the heaviest part of the process. However, in a branch-and-bound algorithm for MCKP, it is performed only at the root node, while the second phase must be iterated during execution.

Algorithms for solving $C(MCKP)$ in $O(n)$ time, without sorting and reducing the items, have been independently developed by Dyer (1984) and Zemel (1984). These results, however, have not been used, so far, in branch-and-bound algorithms for MCKP, since the reduction phase is essential for the effective solution of the problem.

Branch-and-bound algorithms for MCKP have been presented by Nauss (1978), Sinha and Zoltners (1979), Armstrong, Kung, Sinha and Zoltners (1983), Dyer, Kayal and Walker (1984), Dudzinski and Walukiewicz (1984b, 1987).

The Fortran implementation of the Dyer, Kayal and Walker (1984) algorithm can be obtained from Professor Martin E. Dyer.