6

0-1 Multiple knapsack problem

6.1 INTRODUCTION

The *0-1 Multiple Knapsack Problem* (MKP) is: given a set of n *items* and a set of m *knapsacks* ($m \leq n$), with

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

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

$$c_i = \textit{capacity} \text{ of knapsack } i,$$

select m disjoint subsets of items so that the total profit of the selected items is a maximum, and each subset can be assigned to a different knapsack whose capacity is no less than the total weight of items in the subset. Formally,

$$\text{maximize} \quad z = \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \tag{6.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_{ij} \leq c_i, \qquad i \in M = \{1, \ldots, m\}, \tag{6.2}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \qquad j \in N = \{1, \ldots, n\}, \tag{6.3}$$

$$x_{ij} = 0 \text{ or } 1, \qquad i \in M, j \in N, \tag{6.4}$$

where

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

When $m = 1$, MKP reduces to the 0-1 (single) knapsack problem considered in Chapter 2.

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

$$p_j \text{ and } c_i \text{ are positive integers,} \tag{6.5}$$

$$w_j \leq \max_{i \in M} \{c_i\} \quad \text{for } j \in N, \tag{6.6}$$

$$c_i \geq \min_{j \in N} \{w_j\} \quad \text{for } i \in M, \tag{6.7}$$

$$\sum_{j=1}^{n} w_j > c_i \quad \text{for } i \in M. \tag{6.8}$$

If assumption (6.5) is violated, fractions can be handled by multiplying through by a proper factor, while nonpositive values can easily be handled by eliminating all items with $p_j \leq 0$ and all knapsacks with $c_i \leq 0$. (There is no easy way, instead, of transforming an instance so as to handle negative weights, since the Glover (1965) technique given in Section 2.1 does not extend to MKP. All the considerations in this Chapter, however, easily extend to the case of nonpositive values.) Items j violating assumption (6.6), as well as knapsacks i violating assumption (6.7), can be eliminated. If a knapsack, say i^*, violates assumption (6.8), then the problem has the trivial solution $x_{i^*j} = 1$ for $j \in N$, $x_{ij} = 0$ for $i \in M \setminus \{i^*\}$ and $j \in N$. Finally, observe that if $m > n$ then the $(m - n)$ knapsacks of smallest capacity can be eliminated.

We will further assume that the items are sorted so that

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}. \tag{6.9}$$

In Section 6.2 we examine the relaxation techniques used for determining upper bounds. Approximate algorithms are considered in Sections 6.3 and 6.6 . In Section 6.4 we describe branch-and-bound algorithms, in Section 6.5 reduction techniques. The results of computational experiments are presented in Section 6.7.

6.2 RELAXATIONS AND UPPER BOUNDS

Two techniques are generally employed to obtain upper bounds for MKP: the surrogate relaxation and the Lagrangian relaxation. As we show in the next section, the continuous relaxation of the former also gives the value of the continuous relaxation of MKP.

6.2.1 Surrogate relaxation

Given a positive vector $(\pi_1, \ldots, \pi_m)$ of multipliers, the standard *surrogate relaxation*, $S(MKP, \pi)$, of MKP is

$$\text{maximize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \tag{6.10}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \pi_i \sum_{j=1}^{n} w_j x_{ij} \leq \sum_{i=1}^{m} \pi_i c_i \tag{6.11}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \qquad j \in N, \tag{6.12}$$

$$x_{ij} = 0 \text{ or } 1, \qquad i \in M, j \in N. \tag{6.13}$$

Note that we do not allow any multiplier, say $\pi_{\bar{i}}$, to take the value zero, since this would immediately produce a useless solution ($x_{\bar{i}j} = 1$ for $j \in N$) of value $\sum_{j=1}^{n} p_j$. The optimal vector of multipliers, i.e. the one producing the minimum value of $z(S(MKP, \pi))$ and hence the tightest upper bound for MKP, is then defined by the following

Theorem 6.1 (Martello and Toth, 1981a) *For any instance of MKP, the optimal vector of multipliers for $S(MKP, \pi)$ is $\pi_i = k$ (k any positive constant) for all $i \in M$.*

Proof. Let $\bar{i} = \arg\min\{\pi_i : i \in M\}$, and suppose that (x_{ij}^*) defines an optimal solution to $S(MKP, \pi)$. A feasible solution of the same value can be obtained by setting $x_{ij}^* = 0$ and $x_{\bar{i}j}^* = 1$ for each $j \in N$ such that $x_{ij}^* = 1$ and $i \neq \bar{i}$ (since the only effect is to decrease the left-hand side of (6.11)). Hence $S(MKP, \pi)$ is equivalent to the 0-1 single knapsack problem

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_{\bar{i}j}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_{\bar{i}j} \leq \left\lfloor \sum_{i=1}^{m} \pi_i c_i / \pi_{\bar{i}} \right\rfloor,$$

$$x_{\bar{i}j} = 0 \text{ or } 1, \qquad j \in N.$$

Since $\lfloor \sum_{i=1}^{m} \pi_i c_i / \pi_{\bar{i}} \rfloor \geq \sum_{i=1}^{m} c_i$, the choice $\pi_i = k$ (k any positive constant) for all $i \in M$ produces the minimum capacity and hence the minimum value of $z(S(MKP, \pi))$. □

By setting $\pi_i = k > 0$ for all $i \in M$, and $y_j = \sum_{i=1}^{m} x_{ij}$ for all $j \in N$, $S(MKP, \pi)$ becomes

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

$$\text{subject to} \quad \sum_{j=1}^{n} w_j y_j \leq \sum_{i=1}^{m} c_i,$$

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

which we denote simply with $S(MKP)$ in the following. Loosely speaking, this relaxation consists in using only one knapsack, of capacity

$$c = \sum_{i=1}^{m} c_i. \tag{6.14}$$

The computation of upper bound $z(S(MKP))$ for MKP has a non-polynomial time complexity, although many instances of the 0-1 knapsack problem can be solved very quickly, as we have seen in Chapter 2. Weaker upper bounds, requiring $O(n)$ time, can however be computed by determining any upper bound on $z(S(MKP))$ through the techniques of Sections 2.2 and 2.3.

A different upper bound for MKP could be computed through its continuous relaxation, $C(MKP)$, given by (6.1), (6.2), (6.3) and

$$0 \leq x_{ij} \leq 1, \qquad i \in M, \ j \in N. \tag{6.15}$$

This relaxation, however, is dominated by any of the previous ones, since it can be proved that its value is equal to that of the continuous relaxation of the surrogate relaxation of the problem, i.e.

Theorem 6.2 $z(C(MKP)) = z(C(S(MKP)))$.

Proof. It is clear that, by setting $\pi_i = k > 0$ for all i, $C(S(MKP))$, which is obtained from (6.10)–(6.13) by relaxing (6.13) to (6.15), coincides with $S(C(MKP))$, which is obtained from (6.1), (6.2), (6.3) and (6.15) by relaxing (6.2) to (6.11). Hence we have $z(C(S(MKP))) \geq z(C(MKP))$. We now prove that $z(C(MKP)) \geq z(C(S(MKP)))$ also holds.

The exact solution $(\bar{y}_j)$ of the continuous relaxation of $S(MKP)$ can easily be determined as follows. If $\sum_{j=1}^{n} w_j \leq c$, where c is given by (6.14), then $\bar{y}_j = 1$ for $j = 1, \ldots, n$ and $z(C(S(MKP))) = \sum_{j=1}^{n} p_j$. Otherwise, from Theorem 2.1,

$$\bar{y}_j = 1 \quad \text{for } j = 1, \ldots, s-1,$$

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

$$\bar{y}_s = \left(c - \sum_{j=1}^{s-1} w_j \right) / w_s,$$

where

$$s = \min \left\{ j : \sum_{k=1}^{j} w_k > c \right\}, \tag{6.16}$$

and

$$z(C(S(MKP))) = \sum_{j=1}^{s-1} p_j + \left(c - \sum_{j=1}^{s-1} w_j \right) p_s / w_s. \tag{6.17}$$

It is now easy to show that there exists a feasible solution $(\bar{x}_{ij})$ to $C(MKP)$ for which $\sum_{i=1}^{m} \bar{x}_{ij} = \bar{y}_j$ for all $j \in N$. Such a solution, in fact, can be determined by consecutively inserting items $j = 1, 2, \ldots$ into knapsack 1 (and setting $\bar{x}_{1,j} = 1$, $\bar{x}_{ij} = 0$ for $i \neq 1$), until the first item, say j^*, is found which does not fit since the residual capacity $\bar{c}_1$ is less than w_{j^*}. We then insert the maximum possible fraction of w_{j^*} into knapsack 1 (by setting $\bar{x}_{1,j^*} = \bar{c}_1 / w_{j^*}$) and continue with the residual weight $\bar{w}_{j^*} = w_{j^*} - \bar{c}_1$ and knapsack 2, and so on. Hence $z(C(MKP)) \geq z(C(S(MKP)))$. □

Example 6.1

Consider the instance of MKP defined by

$n = 6;$

$m = 2;$

$(p_j) = (110, 150, 70, 80, 30, 5);$

$(w_j) = (40, 60, 30, 40, 20, 5);$

$(c_i) = (65, 85).$

The surrogate relaxation is the 0-1 single knapsack problem defined by (p_j), (w_j) and $c = 150$. Its optimal solution can be computed through any of the exact algorithms of Chapter 2:

$(x_j) = (1, 1, 1, 0, 1, 0)$, $z(S(MKP)) = 360$.

Less tight values can be computed, in $O(n)$ time, through any of the upper bounds of Sections 2.2, 2.3. Using the Dantzig (1957) bound (Theorem 2.1), we get

$s = 4$, $(\bar{x}_j) = (1, 1, 1, \frac{1}{2}, 0, 0)$, $U_1 = 370$ $(= z(C(MKP)))$.

This is also the value produced by the continuous relaxation of the given problem since, following the proof of Theorem 6.2, we can obtain, from $(\bar{x}_j)$,

$(\bar{x}_{1,j}) = (1, \frac{5}{12}, 0, 0, 0, 0)$,

$(\bar{x}_{2,j}) = (0, \frac{7}{12}, 1, \frac{1}{2}, 0, 0)$.

Using the Martello and Toth (1977a) bound (Theorem 2.2), we get $U_2 = 363$. □

6.2.2 Lagrangian relaxation

Given a vector $(\lambda_1, \ldots, \lambda_n)$ of nonnegative multipliers, the *Lagrangian relaxation* $L(MKP, \lambda)$ of MKP is

$$\text{maximize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} - \sum_{j=1}^{n} \lambda_j \left(\sum_{i=1}^{m} x_{ij} - 1 \right) \tag{6.18}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_{ij} \leq c_i, \qquad i \in M, \tag{6.19}$$

$$x_{ij} = 0 \text{ or } 1, \qquad i \in M, j \in N. \tag{6.20}$$

Since (6.18) can be written as

$$\text{maximize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} \tilde{p}_j x_{ij} + \sum_{j=1}^{n} \lambda_j, \tag{6.21}$$

where

$$\tilde{p}_j = p_j - \lambda_j, \qquad j \in N, \tag{6.22}$$

the relaxed problem can be decomposed into a series of m independent 0-1 single knapsack problems $(KP_i, i = 1, \ldots, m)$, of the form

$$\text{maximize} \qquad z_i = \sum_{j=1}^{n} \tilde{p}_j x_{ij}$$

$$\text{subject to} \qquad \sum_{j=1}^{n} w_j x_{ij} \leq c_i,$$

$$x_{ij} = 0 \text{ or } 1, \qquad j \in N.$$

Note that all these problems have the same vectors of profits and weights, so the only difference between them is given by the capacity. Solving them, we obtain the solution of $L(MKP, \lambda)$, of value

$$z(L(MKP, \lambda)) = \sum_{i=1}^{m} z_i + \sum_{j=1}^{n} \lambda_j. \tag{6.23}$$

For the Lagrangian relaxation there is no counterpart of Theorem 6.1, i.e. it is not known how to determine analytically the vector (λ_j) producing the lowest possible value of $z(L(MKP, \lambda))$. An approximation of the optimum (λ_j) can be obtained through subgradient optimization techniques which are, however, generally time consuming. Hung and Fisk (1978) were the first to use this relaxation to determine upper bounds for MKP, although Ross and Soland (1975) had used a similar approach for the generalized assignment problem (see Section 7.2.1), of which MKP is a particular case. They chose for (λ_j) the optimal dual variables associated with constraints (6.3) in $C(MKP)$. Using the complementary slackness conditions, it is not difficult to check that such values are

$$\overline{\lambda}_j = \begin{cases} p_j - w_j \dfrac{p_s}{w_s} & \text{if } j < s; \\ 0 & \text{if } j \geq s, \end{cases} \tag{6.24}$$

where s is the critical item of $S(MKP)$, defined by (6.14) and (6.16). (For $S(MKP)$, Hung and Fisk (1978) used the same idea, previously suggested by Balas (1967) and Geoffrion (1969), choosing for (π_i) the optimal dual variables associated with constraints (6.2) in $C(MKP)$, i.e. $\overline{\pi}_i = p_s/w_s$ for all i. Note that, on the basis of Theorem 6.1, this is an optimal choice.)

With choice (6.24), in each KP_i $(i = 1, \ldots, m)$ we have $\tilde{p}_j/w_j = p_s/w_s$ for $j \leq s$ and $\tilde{p}_j/w_j \leq p_s/w_s$ for $j > s$. It follows that $z(C(L(MKP, \overline{\lambda}))) = (p_s/w_s)\sum_{i=1}^{m} c_i + \sum_{j=1}^{n} \overline{\lambda}_j$, so from (6.17) and Theorem 6.2,

$$z(C(L(MKP, \overline{\lambda}))) = z(C(S(MKP))) = z(C(MKP)),$$

i.e. both the Lagrangian relaxation with multipliers $\overline{\lambda}_j$ and the surrogate relaxation

with multipliers $\pi_i = k > 0$ for all i, dominate the continuous relaxation. No dominance exists, instead, between them.

Computing $z(L(MKP, \overline{\lambda}))$ requires a non-polynomial time, but upper bounds on this value, still dominating $z(C(MKP))$, can be determined in polynomial time, by using any upper bound of Sections 2.2 and 2.3 for the m 0-1 single knapsack problems generated.

Example 6.1 (continued)

From (6.24), we get

$(\overline{\lambda}_j) = (30, 30, 10, 0, 0, 0), (\tilde{p}_j) = (80, 120, 60, 80, 30, 5).$

By exactly solving KP_1 and KP_2, we have

$(x_{1,j}) = (0, 1, 0, 0, 0, 1), z_1 = 125,$

$(x_{2,j}) = (1, 0, 0, 1, 0, 1), z_2 = 165.$

Hence $z(L(MKP, \overline{\lambda})) = 360$, i.e. the Lagrangian and surrogate relaxation produce the same value in this case.

By using U_1 or U_2 (see Sections 2.2.1 and 2.3.1) instead of the optimal solution values, the upper bound would result in 370 (= 130 + 170 + 70). □

It is worth noting that feasibility of the solution of $L(MKP, \lambda)$ for MKP can easily be verified, in $O(nm)$ time, by checking conditions (6.3) (for the example above, $x_{1,6} + x_{2,6} \leq 1$ is not satisfied). This is not the case for $S(MKP)$, for which testing feasibility is an NP-complete problem. In fact, determining whether a subset of items can be inserted into knapsacks of given capacities generalizes the bin-packing problem (see Chapter 8) to the case in which containers of different capacity are allowed.

We finally note that a second Lagrangian relaxation is possible. For a given vector $(\mu_1, \ldots, \mu_m)$ of positive multipliers, $L(MKP, \mu)$ is

$$\text{maximize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} - \sum_{i=1}^{m} \mu_i \left(\sum_{j=1}^{n} w_j x_{ij} - c_i \right) \tag{6.25}$$

$$\text{subject to} \quad \sum_{i=1}^{m} x_{ij} \leq 1, \qquad j \in N,$$

$$x_{ij} = 0 \text{ or } 1, \qquad i \in M, j \in N.$$

Note that, as in the case of $S(MKP, \pi)$, we do not allow any multiplier to take

the value zero, which again would produce a useless solution value. By writing (6.25) as

$$\text{maximize} \sum_{i=1}^{m} \sum_{j=1}^{n} (p_j - \mu_i w_j) x_{ij} + \sum_{i=1}^{m} \mu_i c_i,$$

it is clear that the optimal solution can be obtained by determining $i^* = \arg\min\{\mu_i : i \in M\}$, and setting, for each $j \in N$: $x_{i^*j} = 1$ if $p_j - \mu_{i^*} w_j > 0$, $x_{i^*j} = 0$ otherwise, and $x_{ij} = 0$ for all $i \in M \setminus \{i^*\}$. Since this is also the optimal solution of $C(L(MKP, \mu))$, we have $z(L(MKP, \mu)) \geq z(C(MKP))$, i.e. this relaxation cannot produce, for MKP, a bound tighter than the continuous one. (Using $\overline{\mu}_i = p_s/w_s$ for all $i \in M$, we have $z(L(MKP, \overline{\mu})) = \sum_{j=1}^{s-1}(p_j - (p_s/w_s)w_j) + cp_s/w_s = z(C(MKP))$, with c and s given by (6.14) and (6.16), respectively.)

6.2.3 Worst-case performance of the upper bounds

We have seen that the most natural polynomially-computable upper bound for MKP is

$$U_1 = \lfloor z(C(MKP)) \rfloor = \lfloor z(C(S(MKP))) \rfloor = \lfloor z(C(L(MKP, \overline{\lambda}))) \rfloor.$$

Theorem 6.3 $\rho(U_1) = m + 1$.

Proof. We first prove that $\rho(U_1) \leq m + 1$, by showing that

$$z(C(S(MKP))) \leq (m + 1) z(MKP).$$

Consider the solution of $C(S(MKP))$ and let us assume, by the moment, that $\sum_{j=1}^{n} w_j > \sum_{i=1}^{m} c_i$. Let s_i denote the *critical item relative to knapsack* i ($i \in M$) defined as

$$s_i = \min \left\{ k : \sum_{j=1}^{k} w_j > \sum_{l=1}^{i} c_l \right\}. \tag{6.26}$$

Note that, from Theorem 6.2, the only fractional variable in the solution is $\overline{y}_s$, with $s \equiv s_m$. Hence the solution value can be written as

$$z(C(S(MKP))) = \sum_{j=1}^{s_1 - 1} p_j + p_{s_1} + \sum_{j=s_1+1}^{s_2 - 1} p_j + p_{s_2} + \ldots + \sum_{j=s_{m-1}+1}^{s_m - 1} p_j + \left(c - \sum_{j=1}^{s-1} w_j \right) \frac{p_s}{w_s}, \tag{6.27}$$

from which we have the thesis, since

(a) Selecting items $\{s_{i-1}+1,\ldots,s_i-1\}$ (where $s_0=0$) for insertion into knapsack i ($i=1,\ldots,m$), we obtain a feasible solution for MKP, so $z(MKP)\geq\sum_{i=1}^{m}\sum_{j=s_{i-1}+1}^{s_i-1}p_j$;

(b) From assumption (6.6), $z(MKP)\geq p_{s_i}$ for all $i\in M$, hence also $z(MKP)\geq(c-\sum_{j=1}^{s-1}w_j)p_s/w_s$.

If $\sum_{j=1}^{n}w_j\leq\sum_{i=1}^{m}c_i$ the result holds a fortiori, since some terms of (6.27) are null.

To see that $m+1$ is tight, consider the series of instances with: $n\geq 2m$; $c_1=2k$ $(k\geq 2), c_2=\ldots=c_m=k$; $p_1=\ldots=p_{m+1}=k, w_1=\ldots=w_{m+1}=k+1$; $p_{m+2}=\ldots=p_n=1, w_{m+2}=\ldots=w_n=k$. We have $s\leq m+1, z(C(S(MKP)))=(m+1)k(k/(k+1)), z(MKP)=k+(m-1)$, so the ratio $U_1/z(MKP)$ can be arbitrarily close to $(m+1)$, for k sufficiently large. □

Any upper bound U, computable in polynomial time by applying the bounds of Sections 2.2 and 2.3 to $S(MKP)$ or to $L(MKP,\bar{\lambda})$, dominates U_1, hence $\rho(U)\leq m+1$. Indeed, this value is also tight, as can be verified through counter-examples obtained from that of Theorem 6.3 by adding a sufficiently large number of items with $p_j=k$ and $w_j=k+1$.

Finally note that, obviously, $\rho(U)\leq m+1$ also holds for those upper bounds U which can be obtained, in non-polynomial time, by exactly solving $S(MKP)$ or $L(MKP,\bar{\lambda})$.

6.3 GREEDY ALGORITHMS

As in the case of the 0-1 single knapsack problem (Section 2.4), also for MKP the continuous solution produces an immediate feasible solution, consisting (see (6.26), (6.27)) of the assignment of items $s_{i-1}+1,\ldots,s_i-1$ to knapsack i ($i=1,\ldots,m$) and having value

$$z'=\sum_{i=1}^{m}\sum_{j=s_{i-1}+1}^{s_i-1}p_j. \tag{6.28}$$

Since $z'\leq z\leq U_1\leq z'+\sum_{i=1}^{m}p_{s_i}$, where $z=z(MKP)$, the absolute error of z' is less than $\sum_{i=1}^{m}p_{s_i}$. The worst-case relative error, instead, is arbitrarily bad, as shown by the series of instances with $n=2m, c_i=k\geq m$ for $i=1,\ldots,m$, $p_j=w_j=1$ and $p_{j+1}=w_{j+1}=k$ for $j=1,3,\ldots,n-1$, for which $z'=m$ and $z=mk$, so the ratio z'/z is arbitrarily close to 0 for k sufficiently large.

In this case too we can improve on the heuristic by also considering the solution consisting of the best critical item alone, i.e.

$$z^h = \max\left(z', \max_{i \in M}\{p_{s_i}\}\right).$$

The worst-case performance ratio of z^h is $1/(m+1)$. Since, in fact, $z^h \geq z'$ and $z^h \geq p_{s_i}$ for $i = 1, \ldots, m$, we have, from $z \leq z' + \sum_{i=1}^{m} p_{s_i}$, that $z \leq (m+1)z^h$. The series of instances with $n = 2m+2, c_1 = 2k (k > m), c_i = k$ for $i = 2, \ldots, m, p_j = w_j = 1$ and $p_{j+1} = w_{j+1} = k$ for $j = 1, 3, \ldots, n-1$ proves the tightness, since $z^h = k + m + 1$ and $z = (m+1)k$, so the ratio z^h/z is arbitrarily close to $1/(m+1)$ for k sufficiently large. Notice that the "improved" heuristic solution $z^g = \max(z', \max_{j \in N}\{p_j\})$ has the same worst-case performance.

For the heuristic solutions considered so far, value z' can be obtained, without solving $C(MKP)$, by an $O(n)$ greedy algorithm which starts by determining the critical item $s \equiv s_m$ through the procedure of Section 2.2.2, and re-indexing the items so that $j < s$ (resp. $j > s$) if $p_j/w_j > p_s/w_s$ (resp. $p_j/w_j < p_s/w_s$). Indices i and j are then initialized to 1 and the following steps are iteratively executed: (1) if $w_j \leq \overline{c}_i$ ($\overline{c}_i$ the residual capacity of knapsack i), then assign j to i and set $j = j+1$; (2) otherwise, (a) reject the current item (by setting $j = j+1$), (b) decide that the current knapsack is full (by setting $i = i+1$), and (c) waste (!) part of the capacity of the next knapsack (by setting $\overline{c}_i = c_i - (w_{j-1} - \overline{c}_{i-1})$). Clearly, this is a "stupid" algorithm, whose average performance can be immediately improved by eliminating step (c). The worst-case performance ratio, however, is not improved, since for the tightness counter-example above we still have $z^g = k + m + 1$. Trying to further improve the algorithm, we could observe that, in case (2), it rejects an item which could fit into some other knapsack and "closes" a knapsack which could contain some more items. However, if we restrict our attention to $O(n)$ algorithms which only go forward, i.e. never decrease the value of j or i, then by performing, in case (2), only step (a) or only step (b), the worst-case performance is not improved. If just j is increased, then for the same tightness counter-example we continue to have $z^g = k+m+1$. If just i is increased, then for the series of instances with $n = m+3, c_1 = 2k\ (k > 1), c_i = k$ for $i = 2, \ldots, m$, $p_1 = w_1 = p_2 = w_2 = k+1$ and $p_j = w_j = k$ for $j = 3, \ldots, n$, we have $z = (m+1)k$ and $z^g = k+1$.

Other heuristic algorithms which, for example, for each item j perform a search among the knapsacks, are considered in Section 6.6.

6.4 EXACT ALGORITHMS

The optimal solution of MKP is usually obtained through branch-and-bound. Dynamic programming is in fact impractical for problems of this kind, both as regards computing times and storage requirements. (Note in addition that this approach would, for a strongly NP-hard problem, produce a strictly exponential time complexity.)

Algorithms for MKP are generally oriented either to the case of low values of the ratio n/m or to the case of high values of this ratio. Algorithms for the first class (which has applications, for example, when m liquids, which cannot be mixed,

have to be loaded into n tanks) have been presented by Neebe and Dannenbring (1977) and by Christofides, Mingozzi and Toth (1979). In the following we will review algorithms for the second class, which has been more extensively studied.

6.4.1 Branch-and-bound algorithms

Hung and Fisk (1978) proposed a depth-first branch-and-bound algorithm in which successive levels of the branch-decision tree are constructed by selecting an item and assigning it to each knapsack in turn. When all the knapsacks have been considered, the item is assigned to a dummy knapsack, $m+1$, implying its exclusion from the current solution. Two implementations have been obtained by computing the upper bound associated with each node as the solution of the Lagrangian relaxation, or the surrogate relaxation of the current problem. The corresponding multipliers, $\overline{\lambda}$ and $\overline{\pi}$, have been determined as the optimal dual variables associated with constraints (6.3) and (6.2), respectively, in the continuous relaxation of the current problem (see Section 6.2.2). The choice of the item to be selected at each level of the decision-tree depends on the relaxation employed: in the Lagrangian case, the algorithm selects the item which, in the solution of the relaxed problem, has been inserted in the highest number of knapsacks; in the surrogate case, the item of lowest index is selected from among those which are still unassigned (i.e., at each level j, item j is selected). The items are sorted according to (6.9), the knapsacks so that

$$c_1 \geq c_2 \geq \ldots \geq c_m.$$

Once the branching item has been selected, it is assigned to knapsacks according to the increasing order of their indices. Figure 6.1 shows the decision nodes generated, when $m = 4$, for branching item j.

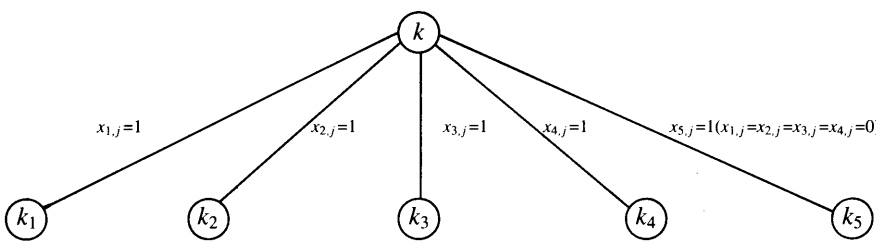

Figure 6.1 Branching strategy for the algorithms of Hung and Fisk (1978)

Martello and Toth (1980a) proposed a depth-first branch-and-bound algorithm using a different branching strategy based on the solution, at each decision node, of the current problem with constraints (6.3) dropped out. From (6.18)–(6.20) it is clear that the resulting relaxed problem coincides with a Lagrangian relaxation with

$\lambda_j = 0$ for $j = 1, \ldots, n$. In the following, this is denoted by $L(MKP, 0)$. For the instance of Example 6.1, we obtain: $(x_{1,j}) = (0, 1, 0, 0, 0, 1)$, $z_1 = 155$, $(x_{2,j}) = (1, 0, 0, 1, 0, 1)$, $z_2 = 195$, so $z(L(MKP, 0)) = 350$. In this case $L(MKP, 0)$ gives a better result than $L(MKP, \overline{\lambda})$. It is not difficult, however, to construct examples for which $z(L(MKP, \overline{\lambda})) < z(L(MKP, 0))$, i.e. neither of the two choices for λ dominates the other. In general, one can expect that the choice $\lambda = \overline{\lambda}$ produces tighter bounds. However, use of $\lambda = (0, \ldots, 0)$ in a branch-and-bound algorithm gives two important advantages:

(a) if no item is assigned to more than one knapsack in the solution of $L(MKP, 0)$, a *feasible and optimal* solution of the current problem has been found, and a backtracking can be immediately performed. If the same happens for $L(MKP, \lambda)$, with $\lambda \neq (0, \ldots, 0)$, the solution is just *feasible* (it is also optimal only when the corresponding value of the original objective function (6.1) equals $z(L(MKP, \lambda))$);

(b) since $(\tilde{p}_j)$ does not change from one level to another, the computation of the upper bounds associated with the decision nodes involves the solution of a lesser number of different 0-1 single knapsack problems.

The strategy adopted in Martello and Toth (1980a) is to select an item for branching which, in solution $(\hat{x}_{ij})$ to the current $L(MKP, 0)$, is inserted into $\overline{m} > 1$ knapsacks (namely, that having the maximum value of $(p_j / w_j) \sum_{i \in M} \hat{x}_{ij}$ is selected). $\overline{m}$ nodes are then generated, by assigning the item in turn to $\overline{m} - 1$ of such knapsacks and by excluding it from these. Suppose that, in the case of Figure 6.1, we have, for the selected item j, $\hat{x}_{1,j} = \hat{x}_{2,j} = \hat{x}_{3,j} = 1$ and $\hat{x}_{4,j} = 0$. Figure 6.2 shows the decision nodes generated.

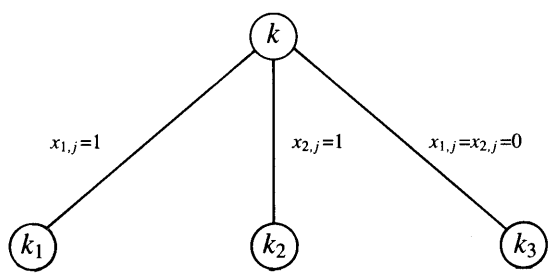

Figure 6.2 Branching strategy for the Martello and Toth (1980a) algorithm

In order to compute the upper bound associated with node k_1 it is sufficient to solve two single knapsack problems: the former for knapsack 2 with condition $x_{2,j} = 0$, the latter for knapsack 3 with condition $x_{3,j} = 0$ (the solutions for knapsacks 1 and 4 are unchanged with respect to those corresponding to the father node k). The upper bound associated with node k_2 can now be computed by solving only the single knapsack problem for knapsack 1 with condition $x_{1,j} = 0$, the

solution of knapsack 3 with condition $x_{3,j} = 0$ having already been computed. Obviously, no single knapsack need now be solved to compute the upper bound associated with node k_3. In general, it is clear that $\overline{m} - 1$ single knapsacks have to be solved for the first node considered, then one for the second node and none for the remaining $\overline{m} - 2$ nodes. Hence, in the worst case ($\overline{m} = m$), only m single knapsack problems have to be solved in order to compute the upper bounds associated with the nodes which each node generates.

In addition we can compute, without solving any further single knapsack problem, the upper bound corresponding to the exclusion of the branching item j from all the $\overline{m}$ knapsacks considered: if this bound is not greater than the best solution so far, it is possible to associate a stronger condition with the branch leading to the $\overline{m}$th node by assigning the object to the $\overline{m}$th knapsack without changing the corresponding upper bound. In the example of Figure 6.2, condition $x_{1,j} = x_{2,j} = 0$ would be replaced by $x_{3,j} = 1$.

A further advantage of this strategy is that, since all the upper bounds associated with the $\overline{m}$ generated nodes are easily computed, the nodes can be explored in decreasing order of their upper bound values.

6.4.2 The "bound-and-bound" method

In Martello and Toth (1981a), MKP has been solved by introducing a modification of the branch-and-bound technique, based on the computation at each decision node not only of an upper bound, but also of a lower bound for the current problem. The method, which has been called *bound-and-bound*, can be used, in principle, to solve any integer linear program. In the next section we describe the resulting algorithm for MKP. Here we introduce the method for the general *0-1 Linear Programming Problem* (ZOLP)

$$\text{maximize} \quad \sum_{j \in N} p_j x_j$$

$$\text{subject to} \quad \sum_{j \in N} a_{ij} x_j \leq b_i, \qquad i \in M,$$

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

Let us suppose, for the sake of simplicity, that all coefficients are non-negative. We define a *partial solution* S as a set, represented as a stack, containing the indices of those variables whose value is fixed: an index in S is *labelled* if the value of the corresponding variable is fixed to 0, *unlabelled* if it is fixed to 1. The *current problem* induced by S, $ZOLP(S)$, is ZOLP with the additional constraints $x_j = 0$ ($j \in S, j$ labelled), $x_j = 1$ ($j \in S, j$ unlabelled).

Let $U(S)$ be any *upper bound* on $z(ZOLP(S))$. Let H be a heuristic procedure which, when applied to $ZOLP(S)$, has the following properties:

(i) a feasible solution ($\tilde{x}_j$) is always found, if one exists;

(ii) this solution is maximal, in the sense that no $\tilde{x}_j$ having value 0 can be set to 1 without violating the constraints.

The value of the solution produced by H, $L(S) = \sum_{j \in N} p_j \tilde{x}_j$, is obviously a *lower bound* on $z(ZOLP(S))$.

A bound-and-bound algorithm for the optimal solution of ZOLP works as follows.

```
procedure BOUND_ AND_ BOUND:
input: N, M, (p_j), (a_ij), (b_i);
output: z, (x_j);
begin
1. [initialize]
   S := Ø;
   z := −∞ ;
2. [heuristic]
   apply heuristic procedure H to ZOLP(S);
   if ZOLP(S) has no feasible solution then go to 4;
   if L(S) > z then
        begin
            z := L(S);
            for each j ∈ N do x_j := x̃_j;
            if z = U(S) then go to 4
        end;
3. [define a new current solution]
   let j be the first index in N\S such that x̃_j = 1;
   if no such j then go to 4 ;
   push j (unlabelled) on S;
   if U(S) > z then go to 3;
4. [backtrack]
   while S ≠ Ø do
        begin
            let j be the index on top of S;
            if j is labelled then pop j from S;
            else
                begin
                    label j;
                    if U(S) > z then go to 2 else go to 4
                end
        end
end.
```

The main conceptual difference between this approach and a standard depth-first branch-and-bound one is that the branching phase is here performed by updating the partial solution through the heuristic solution determining the current lower bound. This gives two advantages:

(a) For all S for which $L(S) = U(S)$, $(\tilde{x}_j)$ is obviously an optimal solution to $ZOLP(S)$, so it is possible to avoid exploration of the decision nodes descending from the current one;

(b) For all S for which $L(S) < U(S)$, S is updated through the heuristic solution previously found by procedure H, so the resulting partial solution can generally be expected to be better than that which would be obtained by a series of forward steps, each fixing a variable independently of the following ones.

On the other hand, in case (b) it is possible that the computational effort spent to obtain $L(S)$ through H may be partially useless: this happens when, after few iterations of Step 3, condition $U(S) \leq z$ holds.

In general, the bound and bound approach is suitable for problems having the following properties:

(i) a "fast" heuristic procedure producing "good" lower bounds can be found;

(ii) the relaxation technique utilized to obtain the upper bounds leads to solutions whose feasibility for the current problem is difficult to check or is seldom verified.

6.4.3 A bound-and-bound algorithm

Martello and Toth (1981a) have derived from the previous framework an algorithm for MKP which consists of an enumerative scheme where each node of the decision-tree generates two branches either by assigning an item j to a knapsack i or by excluding j from i. For the sake of clarity, we give a description close to that of the general algorithm of the previous section, although this is not the most suitable for effective implementation. Stack S_k $(k = 1, \ldots, m)$ contains those items that are currently assigned to knapsack k or excluded from it.

Let $S = \{S_1, \ldots, S_m\}$. At each iteration, i denotes the current knapsack and the algorithm inserts in i the next item j selected, for knapsack i, by the current heuristic solution. Only when no further item can be inserted in i is knapsack $i+1$ considered. Hence, at any iteration, knapsacks $1, \ldots, i-1$ are completely loaded, knapsack i is partially loaded and knapsacks $i+1, \ldots, m$ are empty.

Upper bounds $U = U(S)$ are computed, through surrogate relaxation, by procedure UPPER. Lower bounds $L = L(S)$ and the corresponding heuristic solutions $\tilde{x}$ are computed by procedure LOWER, which finds an optimal solution for the current knapsack, then excludes the items inserted in it and finds an optimal solution for the next knapsack, and so on. For both procedures, on input i is the current knapsack and $(\hat{x}_{kj})$ $(k = 1, \ldots, i;\ j = 1, \ldots, n)$ contains the current solution.

procedure UPPER:
input: $n, m, (p_j), (w_j), (c_k), (\hat{x}_{kj}), (S_k), i$;
output: U;
begin
$\overline{c} := (c_i - \sum_{j \in S_i} w_j \hat{x}_{ij}) + \sum_{k=i+1}^{m} c_k$;
$\overline{N} := \{j : \hat{x}_{kj} = 0 \text{ for } k = 1, \ldots, i\}$;
determine the optimal solution value $\overline{z}$ of the 0-1 single knapsack problem defined by the items in $\overline{N}$ and by capacity $\overline{c}$;
$U := \sum_{k=1}^{i} \sum_{j \in S_k} p_j \hat{x}_{kj} + \overline{z}$
end.

procedure LOWER:
input: $n, m, (p_j), (w_j), (c_k), (\hat{x}_{kj}), (S_k), i$;
output: $L, (\tilde{x}_{kj})$;
begin
$L := \sum_{k=1}^{i} \sum_{j \in S_k} p_j \hat{x}_{kj}$;
$N' := \{j : \hat{x}_{kj} = 0 \text{ for } k = 1, \ldots, i\}$;
$\overline{N} := N' \backslash S_i$;
$\overline{c} := c_i - \sum_{j \in S_i} w_j \hat{x}_{ij}$;
$k := i$;
repeat
determine the optimal solution value $\overline{z}$ of the 0-1 single knapsack problem defined by the items in $\overline{N}$ and by capacity $\overline{c}$, and store the solution vector in row k of $\tilde{x}$;
$L := L + \overline{z}$;
$N' := N' \backslash \{j : \tilde{x}_{kj} = 1\}$;
$\overline{N} := N'$;
$k := k + 1$;
$\overline{c} := c_k$
until $k > m$
end.

The bound-and-bound algorithm for MKP follows. Note that no current solution is defined (hence no backtracking is performed) for knapsack m since, given $\hat{x}_{kj}$ for $k = 1, \ldots, m-1$, the solution produced by LOWER for knapsack m is optimal. It follows that it is convenient to sort the knapsacks so that

$$c_1 \leq c_2 \leq \ldots \leq c_m.$$

The items are assumed to be sorted according to (6.9).

procedure MTM:
input: $n, m, (p_j), (w_j), (c_i)$;
output: $z, (x_{ij})$;
begin
1. [initialize]
for $k := 1$ **to** m **do** $S_k := \emptyset$;

```
for k := 1 to m do for j := 1 to n do x̂_kj := 0;
z := 0;
i := 1;
call UPPER yielding U;
UB := U;
2. [heuristic]
call LOWER yielding L and x̃;
if L > z then
    begin
        z := L;
        for k := 1 to m do for j := 1 to n do x_kj := x̂_kj;
        for k := i to m do for j := 1 to n do
            if x̃_kj = 1 then x_kj := 1;
        if z = UB then return ;
        if z = U then go to 4
    end;
3. [define a new current solution]
repeat
    I := {l : x̃_il = 1};
    while I ≠ Ø do
        begin
            let j = min{l : l ∈ I};
            I := I\{j};
            push j on S_i;
            x̂_ij := 1;
            call UPPER yielding U;
            if U ≤ z then go to 4
        end;
    i := i + 1
until i = m;
i := m − 1;
4. [backtrack]
repeat
    while S_i≠Ø do
        begin
            let j be the item on top of S_i;
            if x̂_ij = 0 then pop j from S_i;
            else
                begin
                    x̂_ij = 0;
                    call UPPER yielding U;
                    if U > z then go to 2
                end
        end;
    i := i − 1
until i = 0
end.
```

The Fortran implementation of procedure MTM (also presented in Martello and Toth (1985b)) is included in the present volume. With respect to the above description, it also includes a technique for the parametric computation of upper bounds U. In procedures UPPER and LOWER, the 0-1 single knapsack problems are solved through procedure MT1 of Section 2.5.2. (At each execution, the items are already sorted according to (6.9), so there would be no advantage in using procedure MT2 of Section 2.9.3.)

Example 6.2

Consider the instance of MKP defined by

$n = 10;$

$m = 2;$

$(p_j) = (78, 35, 89, 36, 94, 75, 74, 79, 80, 16);$

$(w_j) = (18, 9, 23, 20, 59, 61, 70, 75, 76, 30);$

$(c_i) = (103, 156).$

Applying procedure MTM, we obtain the branch decision-tree of Figure 6.3. At the nodes, $\hat{z}$ gives the current solution value, $(\hat{c}_i)$ the current residual capacities.

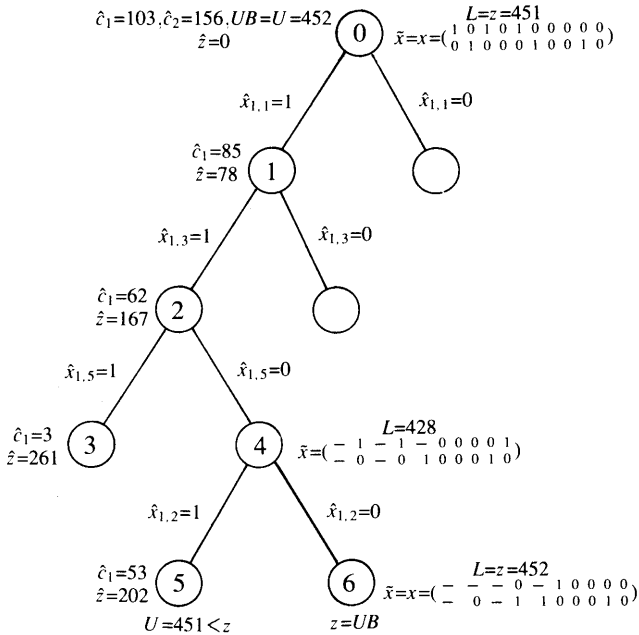

Figure 6.3 Decision-tree of procedure MTM for Example 6.2

The value of U is not given for the nodes for which the parametric computation was able to ensure that its value was the same as for the father node. The optimal solution is

$$(x_{ij}) = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix};$$

$z = 452$. □

A modified version of procedure MTM, in which dominance criteria among nodes of the decision-tree are applied, has been proposed by Fischetti and Toth (1988). Its performance is experimentally better for low values of the ratio n/m.

6.5 REDUCTION ALGORITHMS

The size of an instance of MKP can be reduced, as for the 0-1 knapsack problem (Section 2.7), by determining two sets, $J1$ and $J0$, containing those items which, respectively, must be and cannot be in an optimal solution. In this case, however, only $J0$ allows one to reduce the size of the problem by eliminating the corresponding items, while $J1$ cannot specify in which knapsack the items must be inserted, so it only gives information which can be imbedded in an implicit enumeration algorithm.

Ingargiola and Korsh (1975) presented a specific reduction procedure, based on dominance between items. Let $j\mathrm{D}k$ indicate that item j *dominates* item k, in the sense that, for any feasible solution that includes k but excludes j, there is a better feasible solution that includes j and excludes k. Consequently, if we can determine, for $j = 1, \ldots, n$, a set D_j of items dominated by j, we can exclude all of them from the solution as soon as item j is excluded. If the items are sorted according to (6.9), D_j obviously contains all items $k > j$ such that $w_k \geq w_j$ and $p_k \leq p_j$, plus other items which can be determined as follows.

procedure IKRM:
input: $n, (p_k), (w_k), j$;
output: D_j;
begin
  $D_j := \{k : k > j,\ w_k \geq w_j \text{ and } p_k \leq p_j\}$;
  repeat
    $d := |D_j|$;
    for each $k \in \{l : p_l/w_l \leq p_j/w_j\} \backslash (D_j \cup \{j\})$ **do**
      if $\exists A \subseteq D_j : w_j + \sum_{a \in A} w_a \leq w_k$ and
        $p_j + \sum_{a \in A} p_a \geq p_k$
      then $D_j := D_j \cup \{k\}$
  until $|D_j| = d$
end.

The items added to D_j in the repeat-until loop are dominated by j since, for any solution that includes k but excludes j and, hence, all $a \in A$, there is a better solution that includes $\{j\} \cup A$ and excludes k. Once sets D_j ($j = 1, \ldots, n$) have been determined, if a feasible solution of value, say, $\bar{z}$ is known, a set of items which must be in an optimal solution is

$$J1 = \left\{ j : \sum_{k \in N \setminus (\{j\} \cup D_j)} p_k \leq \bar{z} \right\},$$

since the exclusion of any item $j \in J1$, and hence of all items in D_j, would not leave enough items to obtain a better solution. Observe now that, for any item k, set

$$I_k = \{j : j \, \mathrm{D}k, j \notin J1\}$$

contains items which must be included in any solution including k. Hence a set of items which must be excluded from an optimal solution is

$$J0 = \left\{ k : w_k + \sum_{j \in I_k} w_j > \sum_{i \in M} c_i - \sum_{j \in J1} w_j \right\}.$$

The time complexity of IKRM is $O(n^2 \varphi(n))$, where $\varphi(n)$ is the time required for the search of a suitable subset $A \subseteq D_j$. Exactly performing this search, however, requires exponential time, so a heuristic search should be used to obtain a polynomial algorithm. In any case, the overall time complexity for determining $J1$ and $J0$ is $O(n^3 \varphi(n))$, so the method can be useful only for low values of n or for very difficult problems.

6.6 APPROXIMATE ALGORITHMS

6.6.1 On the existence of approximation schemes

Let P be a maximization problem whose solution values are all positive integers. Let *length*(I) and *max*(I) denote, for any instance $I \in P$, the number of symbols required for encoding I and the magnitude of the largest number in I, respectively. Let $z(I)$ denote the optimal solution value for I. Then

Theorem 6.4 (Garey and Johnson, 1978) *If P is NP-hard in the strong sense and there exists a two-variable polynomial q such that, for any instance $I \in P$,*

$$z(I) < q(length(I), max(I)),$$

then P cannot be solved by a fully polynomial time approximation scheme unless $\mathcal{P} = \mathcal{NP}$.

Proof. Suppose such a scheme exists. By prefixing $1/\varepsilon = q(length(I), max(I))$, it would produce, in time polynomial in $length(I)$ and $1/\varepsilon$ (hence in pseudo-polynomial time) a solution of value $z^h(I)$ satisfying $(z(I) - z^h(I))/z^h(I) \leq \varepsilon < 1/z(I)$, i.e. $z(I) - z^h(I) < 1$, hence optimal. But this is impossible, P being NP-hard in the strong sense. □ (The analogous result for minimization problems also holds.)

Theorem 6.4 rules out the existence of a *fully polynomial-time approximation scheme* for MKP, since the problem is NP-hard in the strong sense (see Section 1.3) and its solution value satisfies $z < n \max_j \{p_j\} + 1$. Note that the same consideration applies to MKP in minimization form (defined by *minimize* (6.1), subject to: (6.2) with $\leq$ replaced by $\geq$, (6.3) and (6.4)), since its solution value satisfies $z > \min_j \{p_j\} - 1$.

As for the existence of a *polynomial-time approximation scheme*, the following general property can be used:

Theorem 6.5 (Garey and Johnson, 1979) *Let P be a minimization (resp. maximization) problem whose solution values are all positive integers and suppose that, for some fixed positive integer k, the decision problem "Given $I \in P$, is $z(I) \leq k$ (resp. $z(I) \geq k$) ?" is NP-complete. Then, if $\mathcal{P} \neq \mathcal{NP}$, no polynomial-time algorithm for P can produce a solution of value $z^h(I)$ satisfying*

$$\frac{z^h(I)}{z(I)} < 1 + \frac{1}{k} \quad \left(\text{resp. } \frac{z(I)}{z^h(I)} < 1 + \frac{1}{k} \right)$$

and P cannot be solved by a polynomial-time approximation scheme.

Proof. We prove the thesis for the minimization case. Suppose such an algorithm exists. If $z^h(I) \leq k$ then, trivially, $z(I) \leq k$. Otherwise, $z^h(I) \geq k + 1$, so $z(I) > z^h(I)k/(k+1) \geq k$. Hence a contradiction, since the algorithm would solve an NP-complete problem in polynomial time. (The proof for the maximization case is almost identical.) □

We can use Theorem 6.5 to exclude the existence of a polynomial-time approximation scheme for MKP in minimization form. We use the value $k = 1$. Given any instance $(w_1, \ldots, w_n)$ of PARTITION (see Section 1.3), define an instance of MKP in minimization form having $p_1 = 1, p_2 = \ldots = p_n = 0$, an additional item with $p_{n+1} = 2$ and $w_{n+1} = \sum_{j=1}^{n} w_j$, and two knapsacks with $c_1 = c_2 = \frac{1}{2} \sum_{j=1}^{n} w_j$. Deciding whether the solution value is no greater than 1 is NP-complete, since the answer is *yes* if and only if the answer for the instance of PARTITION is *yes*.

For MKP in maximization form, instead, no proof is known, to our knowledge, for ruling out the existence of a polynomial-time approximation scheme, although no such scheme is known.

6.6.2 Polynomial-time approximation algorithms

In Section 6.3 we have examined the worst-case performance of an $O(n)$ greedy algorithm for MKP. In Section 6.4.3 we have introduced an approximate algorithm (LOWER) requiring exact solution of m single knapsack problems, hence, in the worst case, a non-polynomial running time. A different non-polynomial heuristic approach has been proposed by Fisk and Hung (1979), based on the exact solution of the surrogate relaxation, $S(MKP)$, of the problem. Let X_S denote the subset of items producing $z(S(MKP))$. The algorithm considers the items of X_S in decreasing order of weight, and tries to insert each item in a randomly selected knapsack or, if it does not fit, in any of the remaining knapsacks. When an item cannot be inserted in any knapsack, for each pair of knapsacks it attempts exchanges between items (one for one, then two for one, then one for two) until an exchange is found which fully utilizes the available space in one of the knapsacks. If all the items of X_S are inserted, an optimal solution is found; otherwise, the current (suboptimal) feasible solution can be improved by inserting in the knapsacks, in a greedy way, as many items of $N \setminus X_S$ as possible.

Martello and Toth (1981b) proposed a polynomial-time approximate algorithm which works as follows. The items are sorted according to (6.9), and the knapsacks so that

$$c_1 \leq c_2 \leq \ldots \leq c_m. \tag{6.29}$$

An initial feasible solution is determined by applying the greedy algorithm (Section 2.4) to the first knapsack, then to the second one by using only the remaining items, and so on. This is obtained by calling m times the following procedure, giving the capacity $\bar{c}_i = c_i$ of the current knapsack and the current solution, of value z, stored, for $j = 1, \ldots, n$, in

$$y_j = \begin{cases} 0 \text{ if item } j \text{ is currently unassigned;} \\ \text{index of the knapsack it is assigned to, otherwise.} \end{cases}$$

```
procedure GREEDYS:
input: n, (p_j), (w_j), z, (y_j), i, c̄_i;
output: z, (y_j);
begin
    for j := 1 to n do
        if y_j = 0 and w_j ≤ c̄_i then
            begin
                y_j := i;
                c̄_i := c̄_i − w_j;
                z := z + p_j
            end
end.
```

After GREEDYS has been called m times, the algorithm improves on the solution

through local exchanges. First, it considers all pairs of items assigned to different knapsacks and, if possible, interchanges them should the insertion of a new item into the solution be allowed. When all pairs have been considered, the algorithm tries to exclude in turn each item currently in the solution and to replace it with one or more items not in the solution so that the total profit is increased.

Computational experiments (Martello and Toth, 1981b) indicated that the exchanges tend to be much more effective when, in the current solution, each knapsack contains items having dissimilar profit per unit weight. This, however, is not the case for the initial solution determined with GREEDYS. In fact, for the first knapsacks, the best items are initially inserted and, after the critical item has been encountered, generally other "good" items of smaller weight are selected. It follows that, for the last knapsacks, we can expect that only "bad" items are available. Hence, the exchange phases are preceded by a rearrangement of the initial solution. This is obtained by removing from the knapsacks all the items currently in the solution, and reconsidering them according to *increasing* profit per unit weight, by trying to assign each item to the next knapsack, in a cyclic manner. (In this way the items with small weight are considered when the residual capacities are small.)

The resulting procedure follows. It is assumed that items and knapsacks are sorted according to (6.9) and (6.29).

```
procedure MTHM:
input: n, m, (p_j), (w_j), (c_i);
output: z, (y_j);
begin
1. [initial solution]
   z := 0;
   for j := 1 to n do y_j := 0;
   for i := 1 to m do
       begin
           c̄_i := c_i;
           call GREEDYS
       end;
2. [rearrangement]
   z := 0;
   for i := 1 to m do c̄_i := c_i;
   i := 1;
   for j := n to 1 step-1 do if y_j > 0 then
       begin
           let l be the first index in {i, ..., m} ∪ {1, ..., i − 1} such that
             w_j ≤ c̄_l;
           if no such l then y_j := 0 else
               begin
                   y_j := l;
                   c̄_l := c̄_l − w_j;
                   z := z + p_j;
                   if l < m then i := l + 1 else i := 1
```

```
            end
        end;
    for i := 1 to m do call GREEDYS;
3. [first improvement]
    for j := 1 to n do if y_j > 0 then
        for k := j + 1 to n do if 0 < y_k ≠ y_j then
            begin
                h := arg max{w_j, w_k};
                l := arg min{w_j, w_k};
                d := w_h − w_l;
                if d ≤ c̄_{y_l} and c̄_{y_h} + d ≥ min{w_u : y_u = 0} then
                    begin
                        t := arg max{p_u : y_u = 0 and w_u ≤ c̄_{y_h} + d};
                        c̄_{y_h} := c̄_{y_h} + d − w_t;
                        c̄_{y_l} := c̄_{y_l} − d;
                        y_t := y_h;
                        y_h := y_l;
                        y_l := y_t;
                        z := z + p_t
                    end
            end;
4. [second improvement]
    for j := n to 1 step-1 do if y_j > 0 then
        begin
            c̄ := c̄_{y_j} + w_j;
            Y := Ø;
            for k := 1 to n do
                if y_k = 0 and w_k ≤ c̄ then
                    begin
                        Y := Y ∪ {k};
                        c̄ := c̄ − w_k
                    end;
            if Σ_{k∈Y} p_k > p_j then
                begin
                    for each k ∈ Y do y_k := y_j;
                    c̄_{y_j} := c̄;
                    y_j := 0;
                    z := z + Σ_{k∈Y} p_k − p_j
                end
        end
end.
```

No step of MTHM requires more than $O(n^2)$ time. This is obvious for Steps 1 and 2 (since GREEDYS takes $O(n)$ time) and for Step 4. As for Step 3, it is enough to observe that the updating of $\min\{w_u : y_u = 0\}$ and the search for t (in the inner loop) are executed only when a new item enters the solution, hence $O(n)$ times in total.

The Fortran implementation of MTHM is included in the present volume. With

respect to the above description: (a) at Step 1 it includes the possibility of using, for small-size problems, a more effective (and time consuming) way for determining the initial solution; (b) Step 3 incorporates additional tests to avoid the examination of hopeless pairs; (c) the execution of Step 4 is iterated until no further improvement is found. (More details can be found in Martello and Toth (1981b).)

Example 6.3

Consider the instance of MKP defined by

$$n = 9 ;$$

$$m = 2 ;$$

$$(p_j) = (80, 20, 60, 40, 60, 60, 65, 25, 30);$$

$$(w_j) = (40, 10, 40, 30, 50, 50, 55, 25, 40);$$

$$(c_i) = (100, 150).$$

After Step 1 we have

$$(y_j) = (1, 1, 1, 2, 2, 2, 0, 0, 0),$$

$$z = 320 .$$

Step 2 changes (y_j) to

$$(y_j) = (2, 1, 2, 1, 2, 1, 0, 0, 0), \text{ with } (\overline{c}_i) = (10, 20).$$

Step 3 interchanges items 1 and 4, and produces

$$(y_j) = (1, 1, 2, 2, 2, 1, 0, 2, 0), \text{ with } (\overline{c}_i) = (0, 5),$$

$$z = 345 .$$

Step 4 excludes item 5, and produces

$$(y_j) = (1, 1, 2, 2, 0, 1, 2, 2, 0), \text{ with } (\overline{c}_i) = (0, 0),$$

$$z = 350,$$

which is the optimal solution. □

6.7 COMPUTATIONAL EXPERIMENTS

Tables 6.1 and 6.2 compare the Fortran IV implementations of the exact algorithms of the previous sections on randomly generated test problems, using *uncorrelated items* with

Table 6.1 Uncorrelated items; dissimilar capacities. CDC-Cyber 730 in seconds. Average times over 20 problems

m	n	HF	MT	MTM	IKRM + MTM
2	25	0.221	0.143	0.076	0.119
	50	0.694	0.278	0.112	0.333
	100	1.614	1.351	0.159	1.297
	200	6.981	7.182	0.223	6.551
3	25	4.412	9.363	0.458	0.463
	50	54.625	17.141	0.271	0.472
	100	—	—	0.327	1.542
	200	—	—	0.244	6.913
4	25	time limit	time limit	1.027	0.921
	50	—	—	0.952	1.102
	100	—	—	0.675	1.892
	200	—	—	0.518	7.084

Table 6.2 Uncorrelated items; similar capacities. CDC-Cyber 730 in seconds. Average times over 20 problems

m	n	HF	MT	MTM	IKRM + MTM
2	25	0.280	0.141	0.191	0.215
	50	0.671	0.473	0.329	0.490
	100	1.666	0.810	0.152	1.295
	200	6.109	4.991	0.313	6.733
3	25	3.302	1.206	1.222	1.101
	50	44.100	2.362	0.561	0.757
	100	—	6.101	0.428	1.622
	200	—	39.809	0.585	7.190
4	25	13.712	6.341	3.690	3.351
	50	time limit	26.100	12.508	9.516
	100	—	—	3.936	3.064
	200	—	—	9.313	7.412

$$p_j \text{ and } w_j \text{ uniformly random in } [10, 100],$$

and two classes of capacities: *dissimilar capacities*, having

$$c_i \text{ uniformly random in } \left[0, \left(0.5 \sum_{j=1}^{n} w_j - \sum_{k=1}^{i-1} c_k\right)\right] \quad \text{for } i = 1, \ldots, m-1,$$

and *similar capacities*, having

$$c_i \text{ uniformly random in } \left[0.4 \sum_{j=1}^{n} w_j/m,\ 0.6 \sum_{j=1}^{n} w_j/m\right] \text{ for } i = 1, \ldots, m-1.$$

For both classes, the capacity of the mth knapsack was set to

$$c_m = 0.5 \sum_{j=1}^{n} w_j - \sum_{i=1}^{m-1} c_i .$$

Whenever an instance did not satisfy conditions (6.5)–(6.8), a new instance was generated. The entries in the tables give average running times, expressed in seconds, comprehensive of the sorting times.

For each value of m and n, 20 instances were generated and solved on a CDC-Cyber 730. Each algorithm had a time limit of 300 seconds to solve the 80 instances generated for each value of m. When this limit was reached, we give the average time only if the number of solved instances was significant.

Tables 6.1 and 6.2 compare, on small-size problems, the branch-and-bound algorithms of Hung and Fisk (1978) and Martello and Toth (1980a) (Section 6.4.1) and the bound-and-bound algorithm MTM (Section 6.4.3). Three implementations of the Hung and Fisk (1978) algorithm are possible, according to the relaxation used (Lagrangian, surrogate, or a combination of the two). In addition, the algorithm can be run with or without previous application of the Ingargiola and Korsh (1975) reduction procedure IKRM (Section 6.5). Each entry in columns HF gives the lowest of the six average times obtained. Similarly, columns MT give the lowest of the four times obtained for the Martello and Toth (1980a) algorithm (Lagrangian or combination of Lagrangian and surrogate relaxation, with or without the application of IKRM). The last two columns refer to algorithm MTM, without and with the application of IKRM, respectively. For all the algorithms, the solution of the 0-1 single knapsack problems was obtained using algorithm MT1 of Section 2.5.2.

The tables show that MTM is the fastest method, and that use of the reduction procedure generally produces a considerable increase in the total computing time (except for very difficult problems). MT is generally faster than HF. The different capacity generations have little effect on HF and MT. For MTM, instead, problems with dissimilar capacities are considerably easier. This can be explained by observing that the algorithm generates no decision nodes for the last knapsack, so it is at an advantage when one of the capacities is much greater than the others. We used problems with dissimilar capacities to test MTM on larger instances.

Table 6.3 compares the exact algorithm MTM with the approximate algorithm MTHM. In addition, we analyse the behaviour of MTM when used to produce approximate solutions, by halting execution after B backtrackings (with B = 10 or 50). For each approximate algorithm we give, in brackets, the average percentage error. The table shows that the time required to find the exact solution increases much more steeply with m than with n and tends to become impractical for $m > 10$.

Table 6.3 Uncorrelated items; dissimilar capacities. CDC-Cyber 730 in seconds. Average times (average percentage errors) over 20 problems

m	n	MTM exact time	MTHM time (% error)	MTM (B = 10) time (% error)	MTM (B = 50) time (% error)
2	50	0.082	0.013(0.170)	0.049(0.028)	0.070(0.004)
	100	0.129	0.031(0.147)	0.089(0.018)	0.127(0.000)
	200	0.153	0.057(0.049)	0.143(0.000)	0.152(0.000)
	500	0.243	0.132(0.020)	0.242(0.000)	0.242(0.000)
	1 000	0.503	0.266(0.003)	0.502(0.000)	0.502(0.000)
5	50	1.190	0.018(0.506)	0.157(0.344)	0.434(0.312)
	100	1.014	0.040(0.303)	0.268(0.076)	0.601(0.027)
	200	1.178	0.074(0.148)	0.327(0.018)	0.687(0.012)
	500	0.862	0.186(0.031)	0.659(0.001)	0.705(0.001)
	1 000	1.576	0.391(0.016)	1.231(0.001)	1.576(0.000)
10	50	3.852	0.035(0.832)	0.162(0.287)	0.477(0.211)
	100	7.610	0.057(0.437)	0.324(0.174)	0.950(0.092)
	200	32.439	0.106(0.219)	0.659(0.060)	1.385(0.039)
	500	5.198	0.535(0.078)	1.760(0.009)	3.836(0.003)
	1 000	9.729	0.870(0.031)	3.846(0.003)	7.623(0.001)

When used as a heuristic, MTM gives solutions very close to the optimum; the running times are reasonable and increase slowly with n and m. MTHM is faster than MTM but its solutions are clearly worse.

Tables 6.4 and 6.5 show the behaviour of approximate algorithms (MTM halted after 10 backtrackings and MTHM) on very large-size instances. The Fisk and Hung (1979) algorithm is not considered, since extensive computational experiments (Martello and Toth, 1981b) showed that it is generally dominated by MTHM. All runs were executed on an HP 9000/840 with option "-o" for the Fortran compiler. We used the same capacity generations as in the previous tables. For all data generations, for $n \geq 5000$ the execution of MTHM was halted at the end of Step 3, so as to avoid the most time consuming phase (this is possible through an input parameter in the corresponding Fortran implementation).

Table 6.4 refers to uncorrelated items, obtained by generating

$$p_j \text{ and } w_j \text{ uniformly random in } [1, 1000].$$

The percentage errors were computed with respect to the optimal solution value for $m \leq 5$, with respect to the initial upper bound determined by MTM for larger values. With few exceptions in the case of very large problems, both algorithms require acceptable computing times. The approximation obtained is generally very good. The times of MTM (B = 10) are one order of magnitude larger than those of MTHM, but the errors produced are one order of magnitude smaller. Computational experiments on *weakly correlated* items (w_j uniformly random in [1, 1000], p_j uniformly random in $[w_j - 100, w_j + 100]$) gave similar results, both for computing times and percentage errors.

Table 6.4 Uncorrelated items. HP 9000/840 in seconds. Average times (average percentage errors) over 20 problems

		Dissimilar capacities		Similar capacities	
m	n	MTHM	MTM (B = 10)	MTHM	MTM (B = 10)
2	200	0.266(0.0694)	0.131(0.0049)	0.277(0.0441)	0.157(0.0081)
	500	0.085(0.0208)	0.382(0.0006)	0.086(0.0197)	0.387(0.0011)
	1 000	0.177(0.0048)	0.877(0.0001)	0.173(0.0059)	0.728(0.0002)
	2 000	0.359(0.0017)	1.354(0.0001)	0.392(0.0023)	1.638(0.0000)
	5 000	0.806(0.0009)	3.716(0.0000)	0.802(0.0007)	3.346(0.0000)
	10 000	1.730(0.0004)	4.962(0.0000)	1.691(0.0003)	5.250(0.0000)
5	200	0.418(0.1796)	0.283(0.0235)	0.529(0.2152)	0.328(0.0275)
	500	0.104(0.0278)	0.942(0.0037)	0.109(0.0408)	1.022(0.0069)
	1 000	0.214(0.0105)	2.009(0.0014)	0.203(0.0146)	1.976(0.0012)
	2 000	0.455(0.0038)	3.510(0.0003)	0.409(0.0048)	3.994(0.0003)
	5 000	0.968(0.0010)	7.348(0.0000)	0.888(0.0011)	9.849(0.0000)
	10 000	1.998(0.0004)	9.138(0.0000)	1.843(0.0005)	23.932(0.0000)
10	200	0.064(0.1826)	0.500(0.0582)	0.052(0.3051)	0.046(0.1024)
	500	0.154(0.0344)	1.172(0.0094)	0.132(0.0762)	1.373(0.0135)
	1 000	0.300(0.0143)	2.517(0.0022)	0.262(0.0189)	2.561(0.0032)
	2 000	0.685(0.0041)	6.608(0.0004)	0.531(0.0079)	7.030(0.0008)
	5 000	1.273(0.0009)	8.502(0.0000)	1.143(0.0022)	14.127(0.0001)
	10 000	2.527(0.0004)	15.773(0.0000)	2.294(0.0007)	45.760(0.0000)
20	200	0.100(0.1994)	0.706(0.0865)	0.088(0.9004)	0.614(0.2619)
	500	0.245(0.0471)	1.671(0.0181)	0.198(0.1393)	1.783(0.0327)
	1 000	0.426(0.0136)	4.285(0.0051)	0.403(0.0448)	4.065(0.0075)
	2 000	0.796(0.0059)	7.332(0.0012)	0.754(0.0113)	11.717(0.0016)
	5 000	1.676(0.0015)	17.980(0.0002)	1.659(0.0028)	27.829(0.0002)
	10 000	3.191(0.0005)	30.608(0.0000)	3.466(0.0010)	84.605(0.0000)
40	200	0.188(0.2865)	1.218(0.1923)	0.179(2.4654)	0.995(1.1246)
	500	0.446(0.0752)	3.501(0.0477)	0.378(0.4732)	2.748(0.0808)
	1 000	0.910(0.0255)	7.575(0.0137)	0.696(0.1219)	6.049(0.0173)
	2 000	1.411(0.0081)	12.689(0.0039)	1.289(0.0364)	13.608(0.0041)
	5 000	3.085(0.0022)	27.718(0.0009)	2.761(0.0065)	44.538(0.0004)
	10 000	5.733(0.0008)	37.310(0.0004)	5.364(0.0020)	124.637(0.0001)

Table 6.5 shows the behaviour of MTHM on *strongly correlated* items, obtained with

$$w_j \text{ uniformly random in } [1, 1000],$$

$$p_j = w_j + 100.$$

MTM was not run since it requires the exact solution of 0-1 single knapsack problems, which is practically impossible for this data generation (see Section 2.10.1). The percentage errors were computed with respect to an upper

bound on the solution value of the surrogate relaxation of the problem (we used upper bound U_2 of Section 2.3.1). The computing times are slightly higher than for uncorrelated items; the percentage errors are higher for large values of n.

Table 6.5 Algorithm MTHM. Strongly correlated items. HP 9000/840 in seconds. Average times (average percentage errors) over 20 problems

m	n	Dissimilar capacities	Similar capacities
2	200	0.124(0.0871)	0.114(0.0803)
	500	0.829(0.0422)	0.460(0.0278)
	1 000	1.546(0.0157)	1.078(0.0138)
	2 000	5.333(0.0069)	7.498(0.0083)
	5 000	0.823(0.0236)	0.805(0.0191)
	10 000	1.618(0.0144)	1.571(0.0110)
5	200	0.165(0.1085)	0.130(0.1061)
	500	0.683(0.0364)	0.373(0.0313)
	1 000	1.832(0.0155)	1.214(0.0133)
	2 000	3.500(0.0072)	6.662(0.0076)
	5 000	1.068(0.0272)	0.917(0.0245)
	10 000	2.173(0.0142)	1.919(0.0097)
10	200	0.158(0.1466)	0.091(0.1498)
	500	0.636(0.0383)	0.668(0.0443)
	1 000	1.583(0.0167)	1.217(0.0132)
	2 000	9.943(0.0090)	7.862(0.0079)
	5 000	1.697(0.0278)	1.214(0.0255)
	10 000	3.246(0.0134)	2.507(0.0112)
20	200	0.154(0.6698)	0.194(0.3539)
	500	0.491(0.0624)	0.480(0.0558)
	1 000	1.172(0.0187)	1.833(0.0195)
	2 000	7.293(0.0091)	5.728(0.0082)
	5 000	2.624(0.0237)	1.802(0.0285)
	10 000	5.307(0.0096)	3.686(0.0179)
40	200	0.249(4.2143)	0.446(2.3671)
	500	0.807(0.4680)	1.369(0.1365)
	1 000	1.460(0.0491)	3.477(0.0302)
	2 000	6.481(0.0137)	9.776(0.0108)
	5 000	4.799(0.0241)	2.986(0.0432)
	10 000	9.695(0.0141)	6.031(0.0186)