7

Generalized assignment problem

7.1 INTRODUCTION

The *Generalized Assignment Problem* (GAP) can be described, using the terminology of knapsack problems, as follows. Given n *items* and m *knapsacks*, with

$$p_{ij} = \textit{profit} \text{ of item } j \text{ if assigned to knapsack } i,$$

$$w_{ij} = \textit{weight} \text{ of item } j \text{ if assigned to knapsack } i,$$

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

assign each item to exactly one knapsack so as to maximize the total profit assigned, without assigning to any knapsack a total weight greater than its capacity, i.e.

$$\text{maximize} \quad z = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \tag{7.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_{ij} \leq c_i, \qquad i \in M = \{1, \ldots, m\}, \tag{7.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j \in N = \{1, \ldots, n\}, \tag{7.3}$$

$$x_{ij} = 0 \text{ or } 1, \qquad i \in M, j \in N, \tag{7.4}$$

where

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

The problem is frequently described in the literature as that of optimally assigning n tasks to m processors (n jobs to m agents, and so on), given the profit p_{ij} and the amount of resource w_{ij} corresponding to the assignment of task j to processor i, and the total resource c_i available for each processor i.

The minimization version of the problem can also be encountered in the literature: by defining c_{ij} as the *cost* required to assign item j to knapsack i, MINGAP is

$$\text{minimize} \quad v = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{7.5}$$

$$\text{subject to} \quad (7.2), (7.3), (7.4).$$

GAP and MINGAP are equivalent. Setting $p_{ij} = -c_{ij}$ (or $c_{ij} = -p_{ij}$) for all $i \in M$ and $j \in N$ immediately transforms one version into the other. If the numerical data are restricted to positive integers (as frequently occurs), the transformation can be obtained as follows. Given an instance of MINGAP, define any integer value t such that

$$t > \max_{i \in M,\, j \in N} \{c_{ij}\} \tag{7.6}$$

and set

$$p_{ij} = t - c_{ij} \qquad \text{for } i \in M, j \in N. \tag{7.7}$$

From (7.5) we then have

$$v = t \sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} - \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij},$$

where, from (7.3), the first term is independent of (x_{ij}). Hence the solution (x_{ij}) of GAP also solves MINGAP. The same method transforms any instance of GAP into an equivalent instance of MINGAP (by setting $c_{ij} = \hat{t} - p_{ij}$ for $i \in M, j \in N$, with $\hat{t} > \max_{i \in M,\, j \in N} \{p_{ij}\}$).

Because of constraints (7.3), an instance of the generalized assignment problem does not necessarily have a feasible solution. Moreover, even the feasibility question is NP-complete. In fact, given an instance $(w_1, \ldots, w_n)$ of PARTITION (see Section 1.3), consider the instance of GAP (or MINGAP) having $m = 2$, $w_{1,j} = w_{2,j} = w_j$ and $p_{1,j} = p_{2,j} = 1$ for $j = 1, \ldots, n$, and $c_1 = c_2 = \frac{1}{2} \sum_{j=1}^{n} w_j$. Deciding whether a feasible solution (of value n) to such instance exists is an NP-complete problem, since the answer is yes if and only if the answer to the instance of PARTITION is yes.

The following version of the problem (LEGAP), instead, always admits a feasible solution.

$$\text{maximize} \quad \hat{z} = \sum_{i=1}^{m} \sum_{j=1}^{n} \hat{p}_{ij} x_{ij} \tag{7.8}$$

$$\text{subject to} \quad (7.2), (7.4) \text{ and}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \qquad j \in N. \tag{7.9}$$

LEGAP too is equivalent to GAP. Given any instance of LEGAP, an equivalent instance of GAP will have an additional knapsack of capacity $c_{m+1} = n$, with $p_{m+1,j} = 0$ and $w_{m+1,j} = 1$ for $j \in N$, while $p_{ij} = \hat{p}_{ij}$ for $i \in M$ and $j \in N$. (Knapsack $m + 1$ gives no extra profit and always allows a feasible solution, so $z = \hat{z}$.) Conversely, given any instance of GAP, we can define an integer constant q such that

$$q > \sum_{j \in N} \max_{i \in M} \{p_{ij}\},$$

and set

$$\hat{p}_{ij} = p_{ij} + q \qquad \text{for } i \in M, j \in N.$$

With these profits, any set of n items has a higher value than any set of $k < n$ items. Hence, by solving LEGAP we obtain the solution for GAP (of value $z = \hat{z} - nq$) if (7.3) is satisfied, or we know that the instance of GAP has no feasible solution if $\sum_{i=1}^{m} x_{ij} = 0$ for some j.

LEGAP is a generalization of the 0-1 multiple knapsack problem (Chapter 6), in which $p_{ij} = p_j$ and $w_{ij} = w_j$ for all $i \in M$ and $j \in N$ (i.e. the profit and weight of each item are independent of the knapsack it is assigned to). Lagrangian relaxations for LEGAP have been studied by Chalmet and Gelders (1977).

The best known special case of generalized assignment problem is the *Linear Min–Sum Assignment Problem* (or *Assignment Problem*), which is a MINGAP with $n = m$, $c_i = 1$ and $w_{ij} = 1$ for all $i \in M$ and $j \in N$ (so, because of (7.3), constraints (7.2) can be replaced by $\sum_{j=1}^{n} x_{ij} = 1$ for $i \in M$). The problem can be solved in $O(n^3)$ time through the classical Hungarian algorithm (Kuhn (1955), Lawler (1976); efficient Fortran codes can be found in Carpaneto, Martello and Toth (1988)). The assignment problem, however, is not used in general as a subproblem in algorithms for the generalized case.

Another special case arises when $w_{ij} = w_j$ for all $i \in M$ and $j \in N$. Implicit enumeration algorithms for this case have been presented by De Maio and Roveda (1971) and Srinivasan and Thompson (1973).

Facets of the GAP polytope have been studied by Gottlieb and Rao (1989a, 1989b).

We will suppose, as is usual, that the weights w_{ij} of any GAP instance are positive integers. Hence, without loss of generality, we will also assume that

$$p_{ij} \text{ and } c_i \text{ are positive integers,} \tag{7.10}$$

$$|\{i : w_{ij} \leq c_i\}| \geq 1 \qquad \text{for } j \in N, \tag{7.11}$$

$$c_i \geq \min_{j \in N} \{w_{ij}\} \qquad \text{for } i \in M. \tag{7.12}$$

If assumption (7.10) is violated, (a) fractions can be handled by multiplying through by a proper factor; (b) knapsacks with $c_i \leq 0$ can be eliminated; (c) for each item j having $\min_{i \in M} \{p_{ij}\} \leq 0$, we can set $p_{ij} = p_{ij} + |\min_{i \in M} \{p_{ij}\}| + 1$

for $i \in M$ and subtract $|\min_{i \in M}\{p_{ij}\}| + 1$ from the resulting objective function value. As is the case for the 0-1 multiple knapsack problem, there is no easy way of transforming an instance so as to handle negative weights, but all our considerations easily extend to this case too. If an item violates assumption (7.11) then it cannot be assigned, so the GAP instance is infeasible. Knapsacks violating assumption (7.12) can be eliminated from the instance.

In Section 7.2 we introduce various types of relaxations. Exact and approximate algorithms are described in Sections 7.3 and 7.4, reduction procedures in Section 7.5. Section 7.6 presents the results of computational experiments.

7.2 RELAXATIONS AND UPPER BOUNDS

The continuous relaxation of GAP, $C(GAP)$, given by (7.1)–(7.3) and

$$x_{ij} \geq 0, \qquad i \in M, j \in N, \tag{7.13}$$

is rarely used in the literature since it does not exploit the structure of the problem and tends to give solutions a long way from feasibility.

7.2.1 Relaxation of the capacity constraints

Ross and Soland (1975) have proposed the following upper bound for GAP. First, constraints (7.2) are relaxed to

$$w_{ij}x_{ij} \leq c_i, \quad i \in M, j \in N,$$

and the optimal solution $\hat{x}$ to the resulting problem is obtained by determining, for each $j \in N$,

$$i(j) = \arg\max\{p_{ij} : i \in M, w_{ij} \leq c_i\}$$

and setting $\hat{x}_{i(j),j} = 1$ and $\hat{x}_{ij} = 0$ for all $i \in M \setminus \{i(j)\}$. The resulting upper bound, of value

$$U_0 = \sum_{j=1}^{n} p_{i(j),j}, \tag{7.14}$$

is then improved as follows. Let

$$N_i = \{j \in N : \hat{x}_{ij} = 1\}, \qquad i \in M,$$

$$d_i = \sum_{j \in N_i} w_{ij} - c_i, \qquad i \in M,$$

$$M' = \{i \in M : d_i > 0\},$$

$$N' = \bigcup_{i \in M'} N_i.$$

Given a set S of numbers, we denote with $\max_2 S$ (resp. $\min_2 S$) the second maximum (resp. minimum) value in S, and with $\arg\max_2 S$ (resp. $\arg\min_2 S$) the corresponding index. Since M' is the set of those knapsacks for which the relaxed constraint (7.2) is violated,

$$q_j = p_{i(j),j} - \max_2\{p_{ij} : i \in M, w_{ij} \leq c_i\}, \quad j \in N'$$

gives the minimum penalty that will be incurred if an item j currently assigned to a knapsack in M' is reassigned. Hence, for each $i \in M'$, a lower bound on the loss of profit to be paid in order to satisfy constraint (7.2) is given by the solution to the 0-1 single knapsack problem in minimization form (see Section 2.1), KP_i^1 ($i \in M'$), defined by

$$\text{minimize} \quad v_i = \sum_{j \in N_i} q_j y_{ij}$$

$$\text{subject to} \quad \sum_{j \in N_i} w_{ij} y_{ij} \geq d_i,$$

$$y_{ij} = 0 \text{ or } 1, \qquad j \in N_i,$$

where $y_{ij} = 1$ if and only if item j is removed from knapsack i. The resulting Ross and Soland (1975) bound is thus

$$U_1 = U_0 - \sum_{i \in M'} v_i. \tag{7.15}$$

This bound can also be derived from the Lagrangian relaxation, $L(GAP, \lambda)$, of the problem, obtained by dualizing constraints (7.3) in much the same way as described in Section 6.2.2 for the 0-1 multiple knapsack problem. In this case too the relaxed problem,

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

$$\text{subject to} \quad (7.2), (7.4),$$

separates into m 0-1 single knapsack problems (KP_i^λ, $i = 1, \ldots, m$) of the form

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

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

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

where $\tilde{p}_{ij} = p_{ij} - \lambda_j$, and its solution value is

$$z(L(GAP, \lambda)) = \sum_{i=1}^{m} z_i + \sum_{j=1}^{n} \lambda_j. \tag{7.16}$$

It is now easy to see that, by choosing for λ_j the value

$$\bar{\lambda}_j = \max_2\{p_{ij} : i \in M, \ w_{ij} \leq c_i\}, \quad j \in N,$$

we have $z(L(GAP, \bar{\lambda})) = U_1$. In fact, by transforming each KP_i^1 into an equivalent maximization form (as described in Section 2.1), and noting that, in each KP_i^{λ}, $\tilde{p}_{ij} \leq 0$ if $j \notin N_i$ and $w_{ij} \leq c_i$, we have $v_i = \sum_{j \in N_i} q_j - z_i$ $(i \in M')$. Hence, from (7.14) and (7.15),

$$U_1 = \sum_{j \in N} p_{i(j),j} - \sum_{j \in N'} p_{i(j),j} + \sum_{j \in N'} \bar{\lambda}_j + \sum_{i \in M'} z_i;$$

observing that, for $i \notin M'$, by definition we have $\sum_{j \in N_i} w_{ij} \leq c_i$, hence $z_i = \sum_{j \in N_i} \tilde{p}_{ij}$, the Lagrangian solution value (7.16) can be written as

$$\begin{aligned} z(L(GAP, \bar{\lambda})) &= \sum_{i \in M'} z_i + \sum_{i \in M \setminus M'} \sum_{j \in N_i} (p_{ij} - \bar{\lambda}_j) + \sum_{j \in N} \bar{\lambda}_j \\ &= \sum_{i \in M'} z_i + \sum_{j \in N \setminus N'} p_{i(j),j} - \sum_{j \in N \setminus N'} \bar{\lambda}_j + \sum_{j \in N} \bar{\lambda}_j \\ &= U_1. \end{aligned}$$

Example 7.1

Consider the instance of GAP defined by

$$n = 7;$$

$$m = 2;$$

$$(p_{ij}) = \begin{pmatrix} 6 & 9 & 4 & 2 & 10 & 3 & 6 \\ 4 & 8 & 9 & 1 & 7 & 5 & 4 \end{pmatrix};$$

$$(w_{ij}) = \begin{pmatrix} 4 & 1 & 2 & 1 & 4 & 3 & 8 \\ 9 & 9 & 8 & 1 & 3 & 8 & 7 \end{pmatrix}; \ (c_i) = \begin{pmatrix} 11 \\ 22 \end{pmatrix}.$$

The initial bound is $U_0 = 47$. Then we have

$N_1 = \{1, 2, 4, 5, 7\}$, $N_2 = \{3, 6\}$, $(d_i) = (7, -6)$;

$M' = \{1\}$, $N' = \{1, 2, 4, 5, 7\}$;

$q_1 = 2$, $q_2 = 1$, $q_4 = 1$, $q_5 = 3$, $q_7 = 2$.

Solving KP_1^1 we obtain

$v_1 = 2$, $(y_{1j}) = (0, 0, -, 0, 0, -, 1)$,

so the resulting bound is

$U_1 = U_0 - v_1 = 45$. □

7.2.2 Relaxation of the semi-assignment constraints

Martello and Toth (1981c) have obtained an upper bound for GAP by removing constraints (7.3). It is immediate to see that the resulting relaxed problem coincides with $L(GAP\,.\,0)$, hence it decomposes into a series of 0-1 single knapsack problems, KP_i^2 $(i \in M)$, of the form

$$\text{maximize} \quad z_i = \sum_{j=1}^{n} p_{ij} x_{ij}$$

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

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

In this case too, the resulting upper bound, of value

$$\overline{U}_0 = \sum_{i=1}^{m} z_i, \tag{7.17}$$

can be improved by computing a lower bound on the penalty to be paid in order to satisfy the violated constraints. Let

$$N^0 = \left\{ j \in N : \sum_{i \in M} x_{ij} = 0 \right\},$$

$$N^{>} = \left\{ j \in N : \sum_{i \in M} x_{ij} > 1 \right\}$$

be the sets of those items for which (7.3) is violated, and define

$$M^{>}(j) = \{ i \in M : x_{ij} = 1 \} \qquad \text{for all } j \in N^{>};$$

we can compute, using any of the methods of Sections 2.2–2.3,

$$u_{ij}^{0} = \text{upper bound on } z_i \text{ if } x_{ij} = 0, \quad j \in N^{>},\ i \in M^{>}(j),$$

$$u_{ij}^{1} = \text{upper bound on } z_i \text{ if } x_{ij} = 1, \quad j \in N^{0},\ i \in M$$

and determine, for each item $j \in N^0 \cup N^{>}$, a lower bound l_j on the penalty to be paid for satisfying (7.3):

$$l_j = \begin{cases} \min_{i \in M} \{ z_i - \min (z_i, u_{ij}^{1}) \} & \text{if } j \in N^{0}; \\ \sum_{i \in M^{>}(j)} (z_i - \min (z_i, u_{ij}^{0})) & \\ \quad -\max_{i \in M^{>}(j)} \{ z_i - \min (z_i, u_{ij}^{0}) \} & \text{if } j \in N^{>}. \end{cases}$$

The improved upper bound is thus

$$U_2 = \overline{U}_0 - \max_{j \in N^0 \cup N^{>}} \{ l_j \}. \tag{7.18}$$

Example 7.2

Consider the instance of GAP defined by

$$n = 5;$$

$$m = 2;$$

$$(p_{ij}) = \begin{pmatrix} 7 & 3 & 3 & 8 & 7 \\ 5 & 3 & 8 & 4 & 1 \end{pmatrix};$$

$$(w_{ij}) = \begin{pmatrix} 8 & 2 & 8 & 9 & 1 \\ 2 & 2 & 6 & 4 & 4 \end{pmatrix};$$

$$(c_i) = \begin{pmatrix} 11 \\ 7 \end{pmatrix}.$$

The solutions to KP_1^2 and KP_2^2 are

$$z_1 = 17, \quad (x_{1,j}) = (1, 1, 0, 0, 1);$$

$$z_2 = 9, \quad (x_{2,j}) = (1, 0, 0, 1, 0),$$

so $\overline{U}_0 = 26$ and

$N^0 = \{3\}$, $N^> = \{1\}$, $M^>(1) = \{1, 2\}$.

We compute u^0_{ij} and u^1_{ij} through the Dantzig bound (Section 2.2.1), but, for u^1_{ij}, we skip those items k for which $w_{ik} > c_i - w_{ij}$. Hence

$$u^0_{1,1} = 7 + 3 + \left\lfloor \frac{64}{9} \right\rfloor = 17;$$

$$u^0_{2,1} = 3 + \left\lfloor \frac{40}{6} \right\rfloor = 9;$$

$$u^1_{1,3} = 3 + (7 + 3 + \lfloor 0 \rfloor) = 13;$$

$$u^1_{2,3} = 8.$$

It follows that $l_1 = 0$ and $l_3 = \min\{4, 1\} = 1$, so the resulting upper bound is

$$U_2 = \overline{U}_0 - l_3 = 25.$$

For this instance the Ross–Soland bound initially gives $U_0 = 33$, and, after the solution of KP^1_1, $U_1 = 31$. Hence $U_0 > U_1 > \overline{U}_0 > U_2$. On the other hand, computing the Martello–Toth bound for Example 7.1 gives $\overline{U}_0 = 54$, $l_2 = 2$, $l_3 = 1$, $l_5 = 5$, $l_6 = 1$, $l_7 = 2$, and $U_2 = 49$, i.e. $U_1 < U_0 < U_2 < \overline{U}_0$. Thus while, obviously, $U_0 \geq U_1$ and $\overline{U}_0 \geq U_2$, no dominance exists between the other pairs of these bounds. □

7.2.3 The multiplier adjustment method

Fisher, Jaikumar and Van Wassenhove (1986) have developed an upper bound, based on the Lagrangian relaxation $L(GAP, \lambda)$ and dominating the bound proposed by Ross and Soland (1975). Obviously, the continuous and integer solutions of a knapsack problem may differ; this implies (see Fisher (1981)) that, for the *optimal* Lagrangian multiplier λ^*,

$$z(L(GAP, \lambda^*)) \leq z(C(GAP));$$

there is no analytical way, however, to determine λ^*. One possibility is the classical subgradient optimization approach. The novelty of the Fisher–Jaikumar–Van Wassenhove bound consists of a new technique (*multiplier adjustment method*) for determining "good" multipliers. The method starts by setting

$$\lambda_j = \max_2 \{p_{ij} : i \in M, w_{ij} \leq c_i\}, \quad j \in N;$$

as shown in Section 7.2.1, the corresponding Lagrangian relaxation produces the value U_1 of the Ross–Soland bound. Note, in addition, that, with this choice, we

have, for each $j \in N$, $\tilde{p}_{ij}\ (= p_{ij} - \lambda_j) > 0$ for at most one $i \in M$, so there is an optimal Lagrangian solution for this λ which satisfies $\sum_{i=1}^{m} x_{ij} \leq 1$ for all $j \in N$. If some constraint (7.3) is not satisfied, it is, under certain conditions, possible to select a j^* for which $\sum_{i=1}^{m} x_{ij^*} = 0$ and decrease λ_{j^*} by an amount which ensures that in the new Lagrangian solution $\sum_{i=1}^{m} x_{ij^*} = 1$, while $\sum_{i=1}^{m} x_{ij} \leq 1$ continues to hold for all other j. This phase is iterated until either the solution becomes feasible or the required conditions fail.

The following procedure, ADJUST, is an efficient implementation of the multiplier adjustment method. After the initial solution has been determined, a heuristic phase attempts to satisfy violated constraints (7.3) through pairs (i, j) such that $p_{ij} - \lambda_j = 0$. The adjustment phase then considers items j^* violating (7.3) and computes, for $i \in M$, the least decrease Δ_{ij^*} required in λ_{j^*} for item j^* to be included in the optimal solution to KP_i^{λ}. If an item j^* is found for which

(a) $\min_2 \{\Delta_{1,j^*}, \ldots, \Delta_{mj^*}\} > 0$;

(b) decreasing λ_{j^*} by $\min_2 \{\Delta_{1,j^*}, \ldots, \Delta_{mj^*}\}$ the new Lagrangian solution satisfies $\sum_{i=1}^{m} x_{ij} \leq 1$ for all $j \in N$,

then such updating is performed (decreasing the current upper bound value by $\min\{\Delta_{1,j^*}, \ldots, \Delta_{mj^*}\}$) and a new heuristic phase is attempted. If no such j^* exists, the process terminates.

The output variables define the upper bound value

$$U_3 = \sum_{i=1}^{m} z_i + \sum_{j=1}^{n} \lambda_j ; \tag{7.19}$$

if *opt* = "yes", this value is optimal and (x_{ij}) gives the corresponding solution.

procedure ADJUST :
input: $n, m, (p_{ij}), (w_{ij}), (c_i)$;
output: (z_i), (λ_j), (x_{ij}), U_3, *opt*;
begin
  comment: initialization;
  $N := \{1, \ldots, n\}$;
  $M := \{1, \ldots, m\}$;
  for $i := 1$ **to** m **do for** $j := 1$ **to** n **do** $x_{ij} := 0$;
  for $j := 1$ **to** n **do** $\lambda_j := \max_2\{p_{ij} : i \in M, w_{ij} \leq c_i\}$;
  $U_3 := \sum_{j \in N} \lambda_j$;
  for $i := 1$ **to** m **do**
    begin
      $N_i := \{j \in N : p_{ij} - \lambda_j > 0\}$;
      set $x_{ij}(j \in N_i)$ to the solution to
        max $z_i = \sum_{j \in N_i} (p_{ij} - \lambda_j) x_{ij}$
        subject to $\sum_{j \in N_i} w_{ij} x_{ij} \leq c_i$,
                $x_{ij} = 0$ or 1, $j \in N_i$;

```
            U_3 := U_3 + z_i
        end;
    opt := "no";
    if Σ_{i∈M} x_ij = 1 for all j ∈ N then opt := "yes"
    else
        repeat
            comment: heuristic phase;
            IJ := {(i, j) : Σ_{k∈M} x_kj = 0, p_ij − λ_j = 0};
            for each (i, j) ∈ IJ, in order of decreasing p_ij, do
                if Σ_{k∈M} x_kj = 0 and w_ij + Σ_{l∈N} w_il x_il ≤ c_i then x_ij := 1;
            comment: adjustment ;
            if Σ_{i∈M} x_ij = 1 for all j ∈ N then opt := "yes"
            else
                begin
                J := {j ∈ N : Σ_{k∈M} x_kj = 0};
                found := "no";
                repeat
                    let j* be any index in J;
                    J := J \ {j*};
                    M_j* := {i ∈ M : w_ij* ≤ c_i};
                    for each i ∈ M_j* do
                        begin
                        N_i := {j ∈ N \ {j*} : p_ij − λ_j > 0, w_ij ≤ c_i − w_ij*};
                        determine the solution to
                        (KP_i) max z̃_i = Σ_{j∈N_i} (p_ij − λ_j) y_j
                                subject to Σ_{j∈N_i} w_ij y_j ≤ c_i − w_ij*,
                                           y_j = 0 or 1, j ∈ N_i;
                        Δ_ij* := z_i − (z̃_i + (p_ij* − λ_j*))
                        end;
                    if min_2{Δ_ij* : i ∈ M_j*} > 0 then
                        begin
                        i* := arg min {Δ_ij* : i ∈ M_j*};
                        let (y_j), j ∈ N_i*, be the solution found for KP_i*;
                        for each j ∈ N \ N_i* do y_j := 0;
                        y_j* := 1;
                        if y_j + Σ_{i∈M\{i*}} x_ij ≤ 1 for all j ∈ N then
                            begin
                            found := "yes";
                            λ_j* := λ_j* − min_2{Δ_ij* : i ∈ M_j*};
                            replace row i* of x with (y_j);
                            z_i* := z̃_i* + (p_i*j* − λ_j*);
                            U_3 := U_3 − Δ_i*j*
                            end
                        end
                until J = Ø or found = "yes"
                end
        until opt = "yes" or found = "no"
end.
```

Example 7.2 (continued)

The initial solution is obtained by setting

$(\lambda_j) = (5, 3, 3, 4, 1)$:

$z_1 = 10, (x_{1,j}) = (0, 0, 0, 1, 1);$

$z_2 = 5, (x_{2,j}) = (0, 0, 1, 0, 0);$

$U_3 = 16 + (10 + 5) = 31 = U_1.$

The heuristic phase has no effect, hence $J = \{1, 2\}$. For $j^* = 1$ we obtain

$M_1 = \{1, 2\};$

$N_1 = \{5\}, \tilde{z}_1 = 6, y_5 = 1, \Delta_{1,1} = 2;$

$N_2 = \emptyset, \tilde{z}_2 = 0, \Delta_{2,1} = 5,$

hence $i^* = 1$. Replacing $(x_{1,j})$ with $(1, 0, 0, 0, 1)$, condition $\sum_{i \in M} x_{ij} \leq 1$ continues to hold for all $j \in N$, so we have

$(\lambda_j) = (0, 3, 3, 4, 1);$

$z_1 = 13, (x_{1,j}) = (1, 0, 0, 0, 1);$

$U_3 = 29.$

The heuristic phase sets $x_{1,2} = 1$, hence $J = \{4\}$. For $j^* = 4$ we have

$M_4 = \{1, 2\};$

$N_1 = \{5\}, \tilde{z}_1 = 6, y_5 = 1, \Delta_{1,4} = 3;$

$N_2 = \{1\}, \tilde{z}_2 = 5, y_1 = 1, \Delta_{2,4} = 0,$

so the execution terminates with $U_3 = 29$. For this instance we have $U_3 < U_1 (= 31)$, but $U_3 > \overline{U}_0 (= 26) > U_2 (= 25)$. On the other hand, applying procedure ADJUST to the instance of Example 7.1, we initially have $U_3 = 45$, then the first adjustment improves it to 43 and the second to 42 (with two further adjustments producing no improvement). Hence $U_3 = 42 < U_2 (= 49) < \overline{U}_0 (= 54)$. □

Examples 7.1 and 7.2 prove that no dominance exists between the Fisher–Jaikumar–Van Wassenhove bound (U_3) and the Martello–Toth bounds ($\overline{U}_0$ and

U_2), nor between the Ross–Soland (U_0 and U_1) and the Martello–Toth bounds. As already shown, the only dominances among these bounds are $U_3 \leq U_1 \leq U_0$ and $U_2 \leq \overline{U}_0$.

7.2.4 The variable splitting method

Jörnsten and Näsberg (1986) have introduced a new way of relaxing GAP in a Lagrangian fashion. (A general discussion on this kind of relaxation can be found in Guignard and Kim (1987).) By introducing extra binary variables y_{ij} ($i \in M$, $j \in N$) and two positive parameters α and β, the problem is formulated, through *variable splitting*, as

$$\text{maximize} \quad \alpha \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} + \beta \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} y_{ij} \tag{7.20}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_{ij} \leq c_i, \qquad i \in M, \tag{7.21}$$

$$\sum_{i=1}^{m} y_{ij} = 1, \qquad j \in N, \tag{7.22}$$

$$x_{ij} = y_{ij}, \qquad i \in M, j \in N, \tag{7.23}$$

$$x_{ij} = 0 \text{ or } 1, \qquad i \in M, j \in N, \tag{7.24}$$

$$y_{ij} = 0 \text{ or } 1, \qquad i \in M, j \in N. \tag{7.25}$$

We denote problem (7.20)–(7.25) by XYGAP. It is immediate that XYGAP is equivalent to GAP in the sense that the corresponding optimal solution values, $z(XYGAP)$ and $z(GAP)$, satisfy

$$z(XYGAP) = (\alpha + \beta)\, z(GAP). \tag{7.26}$$

The new formulation appears less natural than the original one, but it allows a relaxation of constraints (7.23) through Lagrangian multipliers (μ_{ij}). The resulting problem, $L(XYGAP, \mu)$,

$$\text{maximize} \quad \alpha \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} + \beta \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} y_{ij} + \sum_{i=1}^{m} \sum_{j=1}^{n} \mu_{ij} (x_{ij} - y_{ij}) \tag{7.27}$$

$$\text{subject to} \quad (7.21), (7.22), (7.24), (7.25),$$

keeps both sets of GAP constraints, and immediately separates into two problems, one, $XGAP(\mu)$, in the x variables and one, $YGAP(\mu)$, in the y variables. The former,

$$\text{maximize} \quad z(XGAP(\mu)) = \sum_{i=1}^{m}\sum_{j=1}^{n}(\alpha p_{ij} + \mu_{ij})x_{ij}$$

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

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

has the same structure as $L(GAP, \lambda)$ (Section 7.2.1), hence separates into m 0-1 single knapsack problems (KP_i^{μ}, $i = 1, \ldots, m$) of the form

$$\text{maximize} \quad \hat{z}_i = \sum_{j=1}^{n}(\alpha p_{ij} + \mu_{ij})x_{ij}$$

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

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

the latter

$$\text{maximize} \quad z(YGAP(\mu)) = \sum_{i=1}^{m}\sum_{j=1}^{n}(\beta p_{ij} - \mu_{ij})y_{ij}$$

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

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

has the same structure as the initial Ross–Soland relaxation (Section 7.2.1), hence its optimal solution is

$$y_{ij} = \begin{cases} 1 & \text{if } i = i(j); \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } j \in N,$$

where

$$i(j) = \arg\max \{\beta p_{ij} - \mu_{ij} : i \in M, w_{ij} \leq c_i\}.$$

By solving problems KP_i^{μ} ($i \in M$), we obtain the solution to $L(XYGAP, \mu)$, of value

$$z(L(XYGAP, \mu)) = \sum_{i=1}^{m} \hat{z}_i + \sum_{j=1}^{n} (\beta p_{i(j),j} - \mu_{i(j),j}), \tag{7.28}$$

hence the upper bound

$$U_4 = \lfloor z(L(XYGAP, \mu))/(\alpha + \beta) \rfloor. \tag{7.29}$$

Jörnsten and Näsberg (1986) have proved that, for $\alpha + \beta = 1$ and for the *optimal* Lagrangian multipliers λ^*, μ^*,

$$z(L(XYGAP, \mu^*)) \leq z(L(GAP, \lambda^*)).$$

However, there is no analytical way to determine μ^*, and the multiplier adjustment method of Section 7.2.3 does not appear adaptable to XYGAP. Jörnsten and Näsberg have proposed using a subgradient optimization algorithm to determine a "good" μ. At each iteration, the current μ is updated by setting $\mu_{ij} = \mu_{ij} + t(y_{ij} - x_{ij})$ $(i \in M, j \in N)$, where t is an appropriate positive step.

Example 7.2 (continued)

Using $\alpha = \beta = \frac{1}{2}$ and starting with $\mu_{ij} = 0$ for all $i \in M$, $j \in N$, we obtain

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

i.e., the same solution found for $\overline{U}_0$ (Section 7.2.2), and

$$(y_{ij}) = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix},$$

i.e., the same solution found for U_0. The initial upper bound value is thus

$$U_4 = \lfloor 13 + 16.5 \rfloor = 29 \ (= U_3).$$

Assuming that the initial step is $t = 1$, we then have

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

$$(\alpha p_{ij} + \mu_{ij}) = \begin{pmatrix} \frac{7}{2} & \frac{3}{2} & \frac{3}{2} & 5 & \frac{7}{2} \\ \frac{3}{2} & \frac{3}{2} & 5 & 1 & \frac{1}{2} \end{pmatrix};$$

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

$$(\beta p_{ij} - \mu_{ij}) = \begin{pmatrix} \frac{7}{2} & \frac{3}{2} & \frac{3}{2} & 3 & \frac{7}{2} \\ \frac{7}{2} & \frac{3}{2} & 3 & 3 & \frac{1}{2} \end{pmatrix};$$

$$(y_{ij}) = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix},$$

and the upper bound becomes

$$U_4 = \lfloor 13.5 + 14.5 \rfloor = 28.$$

Further improvements could be obtained by iterating the procedure. □

7.3 EXACT ALGORITHMS

The most commonly used methods in the literature for the exact solution of GAP are depth-first branch-and-bound algorithms.

In the Ross and Soland (1975) scheme, upper bound U_1 (see Section 7.2.1) is computed at each node of the branch-decision tree. The branching variable is selected through the information determined for computing U_1. In fact, the variable chosen to separate, $x_{i^*j^*}$, is the one, among those with $y_{ij} = 0$ ($i \in M'$, $j \in N'$) in the optimal solution to problems KP_i^1 ($i \in M'$), for which the quantity

$$\frac{q_j}{w_{ij} / \left(c_i - \sum_{k=1}^{n} w_{ik} x_{ik}\right)}$$

is a maximum. This variable represents an item j^* which is "well fit" into knapsack i^*, considering both the penalty for re-assigning the item and the residual capacity of the knapsack. Two branches are then generated by imposing $x_{i^*j^*} = 1$ and $x_{i^*j^*} = 0$.

In the Martello and Toth (1981c) scheme, upper bound $\min(U_1, U_2)$ (see Sections 7.2.1, 7.2.2) is computed at each node of the branch-decision tree. In addition, at the root node, a tighter upper bound on the global solution is determined by computing $\min(U_3, U_2)$ (see Section 7.2.3). The information computed for U_2 determines the branching as follows. The separation is performed on item

$$j^* = \arg\max \{l_j : j \in N^0 \cup N^>\},$$

i.e. on the item whose re-assignment is likely to produce the maximum decrease of the objective function. If $j^* \in N^0$, m nodes are generated by assigning j^* to each knapsack in turn (as shown in Figure 7.1(a)); if $j^* \in N^>$, with $M^>(j^*) = \{i_1, i_2, \ldots, i_{\overline{m}}\}$, $\overline{m} - 1$ nodes are generated by assigning j^* to knapsacks $i_1, \ldots, i_{\overline{m}-1}$ in turn, and another node by excluding j^* from knapsacks $i_1, \ldots, i_{\overline{m}-1}$ (as shown in Figure 7.1(b)). With this branching strategy, m single knapsack

problems KP_i^2 must be solved to compute the upper bound associated with the root node, but only one new KP_i^2 for each other node of the tree. In fact if $j^* \in N^0$, imposing $x_{kj^*} = 1$ requires only the solution of problem KP_k^2, the solutions to problems KP_i^2 $(i \neq k)$ being unchanged with respect to the generating node; if $j^* \in N^>$, the strategy is the same as that used in the Martello and Toth (1980a) algorithm for the 0-1 multiple knapsack problem (see Section 6.4.1), for which we have shown that the solution of $\overline{m}$ problems KP_i^2 produces the upper bounds corresponding to the $\overline{m}$ generated nodes.

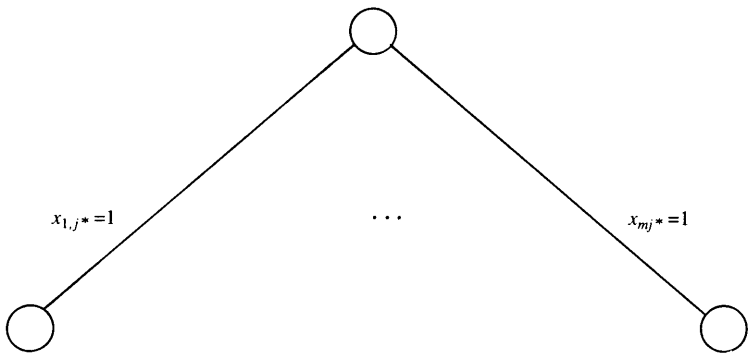

Figure 7.1(a) Branching strategy when $j^* \in N^0$

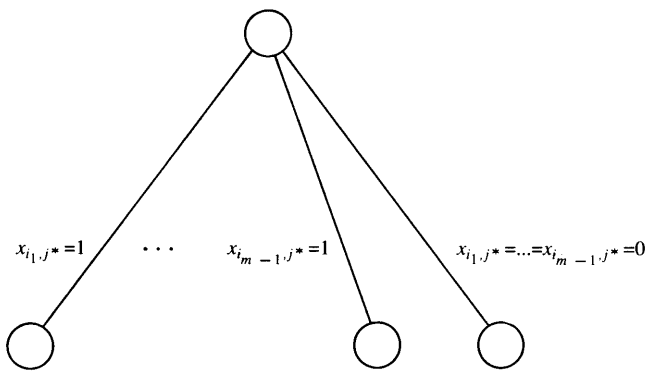

Figure 7.1(b) Branching strategy when $j^* \in N^<$

The execution of the above scheme is preceded by a preprocessing which: (a) determines an approximate solution through a procedure, MTHG, described in the next section; (b) reduces the size of the instance, through two procedures, MTRG1 and MTRG2, described in Section 7.5. (Example 7.3 of Section 7.5 illustrates the branching scheme.) At each decision node, a partial reduction is performed,

by searching for unassigned items which can currently be assigned to only one knapsack. The Fortran implementation of the resulting algorithm (MTG) is included in the present volume.

In the Fisher, Jakumar and Van Wassenhove (1986) scheme, upper bound U_3 (Section 7.2.3) is computed at each node of the branch-decision tree. The branching variable is an $x_{i^*j^*}$ corresponding to a $w_{i^*j^*}$ which is maximum over all variables that have not been fixed to 0 or 1 at previous branches. Two nodes are then generated by fixing $x_{i^*j^*} = 1$ and $x_{i^*j^*} = 0$.

No scheme has been proposed by Jörnsten and Näsberg (1986).

7.4 APPROXIMATE ALGORITHMS

As seen in Section 7.1, determining whether an instance of GAP (or MINGAP) has a feasible solution is an NP-complete problem. It follows that, unless $\mathcal{P} = \mathcal{NP}$, these problems admit no polynomial-time approximate algorithm with fixed worst-case performance ratio, hence also no polynomial-time approximation scheme.

The following polynomial-time algorithm (Martello and Toth, 1981c) provides approximate solutions to GAP. Let f_{ij} be a measure of the "desirability" of assigning item j to knapsack i. We iteratively consider all the unassigned items, and determine the item j^* having the maximum difference between the largest and the second largest f_{ij} ($i \in M$); j^* is then assigned to the knapsack for which f_{ij^*} is a maximum. In the second part of the algorithm the current solution is improved through local exchanges. On output, if *feas* = "no", no feasible solution has been determined; otherwise the solution found, of value z^h, is stored in y_j = (knapsack to which item j is assigned), $j = 1, \ldots, n$.

```
procedure MTHG;
input: n, m, (p_ij), (w_ij), (c_i), (f_ij);
output: z^h, (y_j), feas;
begin
    M := {1, ..., m};
    U := {1, ..., n};
    comment: initial solution;
    feas := "yes"
    for i := 1 to m do c̄_i := c_i;
    z^h := 0;
    while U ≠ Ø and feas = "yes" do
        begin
            d* := −∞;
            for each j ∈ U do
                begin
                    F_j := {i ∈ M : w_ij ≤ c̄_i};
                    if F_j = Ø then feas := "no"
                    else
```

```
                    begin
                        i' := arg max { f_ij : i ∈ F_j };
                        if F_j \ {i'} = Ø then d := +∞
                        else d := f_i'j − max_2{ f_ij : i ∈ F_j };
                        if d > d* then
                            begin
                                d* := d;
                                i* := i';
                                j* := j
                            end
                    end
                end;
            if feas = "yes" then
                begin
                    y_j• := i*;
                    z^h := z^h + p_i•j•;
                    c̄_i• := c̄_i• − w_i•j•;
                    U := U \ {j*}
                end
        end;
    comment: improvement;
    if feas = "yes" then
        for j := 1 to n do
            begin
                i' := y_j;
                A := { p_ij : i ∈ M \ {i'}, w_ij ≤ c̄_i };
                if A ≠ Ø then
                    begin
                        let p_i''j = max A;
                        if p_i''j > p_i'j then
                            begin
                                y_j := i'';
                                z^h := z^h − p_i'j + p_i''j;
                                c̄_i' := c̄_i' + w_i'j;
                                c̄_i'' := c̄_i'' − w_i''j
                            end
                    end
            end
end.
```

Procedure MTHG can be implemented efficiently by initially sorting in decreasing order, for each item j, the values f_{ij} $(i \in M)$ such that $w_{ij} \leq \bar{c}_i$ $(= c_i)$. This requires $O(nm \log m)$ time, and makes immediately available, at each iteration in the inner loop, the pointers to the maximum and the second maximum element of $\{f_{ij} : i \in F_j\}$. Hence the main while loop performs the $O(n)$ assignments within a total of $O(n^2)$ time. Whenever an item is assigned, the decrease in $\bar{c}_{i\bullet}$ can make it necessary to update the pointers. Since, however, the above maxima can only decrease during execution, a total of $O(n^2)$ operations is required by the

algorithm for these checks and updatings. By finally observing that the exchange phase clearly takes $O(nm)$ time, we conclude that the overall time complexity of MTHG is $O(nm\log m + n^2)$.

Computational experiments have shown that good results can be obtained using the following choices for f_{ij}:

(a) $f_{ij} = p_{ij}$ (with this choice the improvement phase can be skipped);
(b) $f_{ij} = p_{ij}/w_{ij}$;
(c) $f_{ij} = -w_{ij}$;
(d) $f_{ij} = -w_{ij}/c_i$.

Example 7.3

Consider the instance of GAP defined by

$$n = 8;$$

$$m = 3;$$

$$(p_{ij}) = \begin{pmatrix} 27 & 12 & 12 & 16 & 24 & 31 & 41 & 13 \\ 14 & 5 & 37 & 9 & 36 & 25 & 1 & 34 \\ 34 & 34 & 20 & 9 & 19 & 19 & 3 & 34 \end{pmatrix};$$

$$(w_{ij}) = \begin{pmatrix} 21 & 13 & 9 & 5 & 7 & 15 & 5 & 24 \\ 20 & 8 & 18 & 25 & 6 & 6 & 9 & 6 \\ 16 & 16 & 18 & 24 & 11 & 11 & 16 & 18 \end{pmatrix}; (c_i) = \begin{pmatrix} 26 \\ 25 \\ 34 \end{pmatrix}.$$

Let us consider execution of MTHG with $f_{ij} = -w_{ij}$. The first phase of the algorithm gives

$$j^* = 4 : d^* = 19, \quad y_4 = 1, \bar{c}_1 = 21;$$

$$j^* = 8 : d^* = 12, \quad y_8 = 2, \bar{c}_2 = 19;$$

$$j^* = 3 : d^* = 9, \quad y_3 = 1, \bar{c}_1 = 12;$$

$$j^* = 1 : d^* = +\infty, \quad y_1 = 3, \bar{c}_3 = 18;$$

$$j^* = 2 : d^* = 8, \quad y_2 = 2, \bar{c}_2 = 11;$$

$$j^* = 6 : d^* = 5, \quad y_6 = 2, \bar{c}_2 = 5;$$

$$j^* = 7 : d^* = 11, \quad y_7 = 1, \bar{c}_1 = 7;$$

$$j^* = 5 : d^* = 4, \quad y_5 = 1, \bar{c}_1 = 0;$$

hence

$$z^h = 191, (y_j) = (3, 2, 1, 1, 1, 2, 1, 2), (\bar{c}_i) = (0, 5, 18).$$

The second phase performs the exchanges

$$j = 2 : y_2 = 3, \bar{c}_2 = 13, \bar{c}_3 = 2;$$

$$j = 5 : y_5 = 2, \bar{c}_1 = 7, \bar{c}_2 = 7;$$

so the solution found is

$$z^h = 232, (y_j) = (3, 3, 1, 1, 2, 2, 1, 2). \square$$

A Fortran implementation of MTHG, which determines the best solution obtainable with choices (a)–(d) for f_{ij}, is included in the present volume. A more complex approximate algorithm, involving a modified subgradient optimization approach and branch-and-bound, can be found in Klastorin (1979).

Mazzola (1989) has derived from MTHG an approximate algorithm for the generalization of GAP arising when the capacity constraints (7.2) are non-linear.

7.5 REDUCTION ALGORITHMS

The following algorithms (Martello and Toth, 1981c) can be used to reduce the size of an instance of GAP. Let (y_j) define a feasible solution (determined, for example, by procedure MTHG of the previous section) of value $z^h = \sum_{j=1}^{n} p_{y_j, j}$.

The first reduction algorithm receives in input the upper bound value U_0 of Section 7.2.1 and the corresponding values $i(j) = \arg\max \{p_{ij} : i \in M, w_{ij} \leq c_i\}$ $(j \in N)$. The algorithm fixes to 0 those variables x_{ij} which, if set to 1, would decrease the bound to a value not greater than z^h. (We obviously assume $z^h < U_0$.) If, for some j, all x_{ij} but one, say $x_{i \bullet j}$, are fixed to 0, then $x_{i \bullet j}$ is fixed to 1. We assume that, on input, all entries of (x_{ij}) are preset to a dummy value other than 0 or 1. On output, k_j^0 $(j \in N)$ has the value $|\{x_{ij} : i \in M, x_{ij} = 0\}|$, and $\bar{c}_i$ gives the residual capacity of knapsack i $(i \in M)$; these values are used by the second reduction algorithm. We also assume that, initially, $\bar{c}_i = c_i$ for all $i \in M$. If, for some j, all x_{ij} are fixed to 0, the feasible solution (y_j) is optimal, hence the output variable *opt* takes the value "yes".

```
procedure MTRG1:
input: n, m, (p_ij), (w_ij), (c̄_i), z^h, U_0, (i(j)), (x_ij);
output: (x_ij), (k_j^0), (c̄_i), opt;
begin
    opt := "no";
    j := 0;
    while j < n and opt = "no" do
        begin
            j := j + 1;
            k_j^0 := 0;
            for i := 1 to m do
                if z^h ≥ U_0 − p_{i(j),j} + p_ij  or  w_ij > c̄_i then
```

```
                    begin
                        x_ij := 0;
                        k_j^0 := k_j^0 + 1
                    end
                else i* := i;
            if k_j^0 = m − 1 then
                begin
                    x_i*j := 1;
                    c̄_i* := c̄_i* − w_i*j
                end
            else if k_j^0 = m then opt := "yes"
        end
end.
```

The time complexity of MTRG1 is clearly $O(nm)$. When the execution fixes some variable to 1, hence decreasing some capacity $\overline{c}_i$, further reductions can be obtained by reapplying the procedure. Since n variables at most can be fixed to 1, the resulting time complexity is $O(n^2m)$.

The second reduction algorithm receives in input (x_{ij}), (k_j^0), $(\overline{c}_i)$, the upper bound value $\overline{U}_0$ of Section 7.2.2 and, for each problem KP_i^2 $(i \in M)$, the corresponding solution $(\hat{x}_{i1}, \ldots, \hat{x}_{in})$ and optimal value z_i. Computation of the upper bounds of Section 7.2.2,

$u_{ij}^0 =$ current upper bound on the solution value of KP_i^2 if $x_{ij} = 0$;

$u_{ij}^1 =$ current upper bound on the solution value of KP_i^2 if $x_{ij} = 1$,

is then used to fix to $\hat{x}_{ij}$ variables x_{ij} which, if set to $1 - \hat{x}_{ij}$, would give an upper bound not greater than z^h. We assume that MTRG1 is first iteratively run, then $\overline{U}_0$ and the solutions to problems KP_i^2 are determined using the reductions obtained. Consequently, the new algorithm cannot take decisions contradicting those of MTRG1. It can, however, fix to 1 more than one variable in a column, or to 0 all the variables in a column. Such situations imply that the current approximate solution is optimal, hence the output variable *opt* takes the value "yes".

```
procedure MTRG2:
input: n, m, (p_ij), (w_ij), (c̄_i), z^h, Ū_0, (z_i), (x̂_ij), (x_ij), (k_j^0);
output: (x_ij), opt;
begin
    opt := "no";
    j := 1;
    repeat
        if k_j^0 < m − 1 then
            begin
                k1 := 0;
                for i := 1 to m do
                    if x_ij ≠ 0 then
```

```
                    if w_ij > c̄_i then
                        begin
                            x_ij := 0;
                            k_j^0 := k_j^0 + 1
                        end
                    else
                        if x̂_ij = 0 and z^h ≥ Ū_0 − z_i + u_ij^1 then
                            begin
                                x_ij := 0;
                                k_j^0 := k_j^0 + 1
                            end
                        else
                            begin
                                if k1 = 0 then i* := i;
                                if x̂_ij = 1 and z^h ≥ Ū_0 − z_i + u_ij^0 then
                                    if k1 = 0 then k1 := 1
                                    else opt := "yes"
                            end;
            if opt = "no" then
                if k_j^0 = m − 1 or k1 = 1 then
                    begin
                        for i := 1 to m do x_ij := 0;
                        x_i*j := 1;
                        c̄_i* := c̄_i* − w_i*j;
                        k_j^0 := m − 1
                    end
                else
                    if k_j^0 = m then opt = "yes"
        end;
    j := j + 1
until j > n or opt = "yes"
end.
```

If u_{ij}^0 and u_{ij}^1 are computed through any of the $O(n)$ methods of Sections 2.2 and 2.3, the time complexity of MTRG2 is $O(mn^2)$. In this case too, when a variable has been fixed to 1, a new execution can produce further reductions.

Example 7.3 (continued)

Using the solution value $z^h = 232$ found by MTHG and the upper bound value $U_0 = 263$, MTRG1 gives

$j = 7 : x_{2,7} = 0, x_{3,7} = 0$, hence $k_7^0 = 2$, so

$x_{1,7} = 1, \bar{c}_1 = 21;$

$j = 8 : x_{1,8} = 0.$

Solving KP_i^2 $(i = 1, 2, 3)$ for the reduced problem, we get

$$(\hat{x}_{ij}) = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & \textcircled{1} & \textcircled{0} \\ 0 & 0 & 0 & 0 & 1 & 1 & \textcircled{0} & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & \textcircled{0} & 0 \end{pmatrix}, \quad (z_i) = \begin{pmatrix} 93 \\ 95 \\ 68 \end{pmatrix}, \quad \overline{U}_0 = 256,$$

where fixed x_{ij} values are circled. Executing MTRG2 we have

$j = 1 : x_{1,1} = 0, x_{2,1} = 0$, hence $x_{3,1} = 1, \overline{c}_3 = 18;$

$j = 4 : x_{2,4} = 0, x_{3,4} = 0$, hence $x_{1,4} = 1, \overline{c}_1 = 16.$

The execution of the Martello and Toth (1981c) branch-and-bound algorithm (see Section 7.3) follows. Computation of U_2 and U_3 for the root node gives

$N^0 = \emptyset, \; N^> = \{5\}, M^>(5) = \{1, 2\};$

$u_{1,5}^0 = 89, \; u_{2,5}^0 = 85; \; l_5 = 4, \; U_2 = 252;$

$U_3 = 245.$

The branching scheme is shown in Figure 7.2. Since $j^* = 5$, we generate nodes 1 and 2, and compute the corresponding bounds.

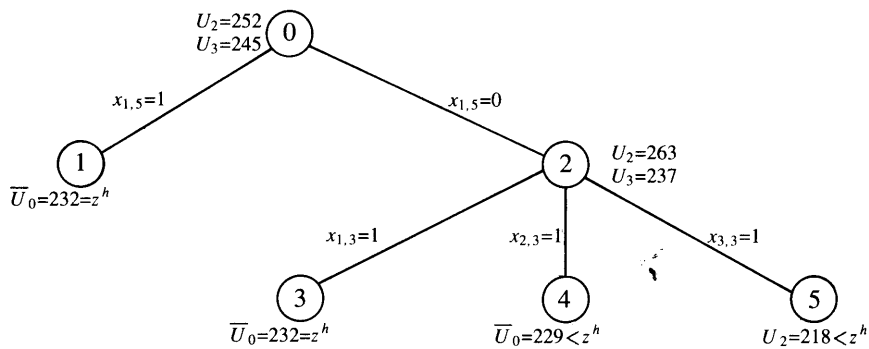

Figure 7.2 Decision-tree for Example 7.3

Node 1 : $(\hat{x}_{2,j}) = (0, 0, 1, 0, 0, 0, 0, 1), z_2 = 71, \; \overline{U}_0 = 232 = z^h.$

Node 2 : $(\hat{x}_{1,j}) = (0, 0, 0, 1, 0, 1, 1, 0), z_1 = 88, \; \overline{U}_0 = 251;$

$N^0 = \{3\}, \; N^> = \{6\}, \; M^>(6) = \{1, \; 2\};$

$u_{1,3}^1 = 69, \; u_{2,3}^1 = 78, \; u_{3,3}^1 = 54, \; l_3 = 14;$

$u_{1,6}^0 = 75, \; u_{2,6}^0 = 96, \; l_6 = 0;$

$$U_2 = 237;$$

$$U_0 = U_1 = 263 \text{ (unchanged).}$$

The highest penalty is $l_3 = 14$, hence $j^* = 3$ and nodes 3, 4, 5 are generated.

Node 3 : $(\hat{x}_{1,j}) = (0, 0, 1, 1, 0, 0, 1, 0)$, $z_1 = 69$, $\overline{U}_0 = 232 = z^h$.

Node 4 : $(\hat{x}_{2,j}) = (0, 0, 1, 0, 1, 0, 0, 0)$, $z_2 = 73$, $\overline{U}_0 = 229 < z^h$.

Node 5 : $(\hat{x}_{3,j}) = (1, 0, 1, 0, 0, 0, 0, 0)$, $z_3 = 54$, $\overline{U}_0 = 237$;

$$N^0 = \{2\}, \quad N^> = \{6\}, \quad M^>(6) = \{1, 2\};$$

$$u^1_{1,2} = 69, \quad u^1_{2,2} = 95, \quad u^1_{3,2} = -\infty, \quad l_2 = 0;$$

$$u^0_{1,6} = 69, \quad u^0_{2,6} = 75, \quad l_6 = 19;$$

$$U_2 = 218 < z^h.$$

The approximate solution $(y_1) = (3, 3, 1, 1, 2, 2, 1, 2)$, of value $z^h = 232$, is thus optimal. □

7.6 COMPUTATIONAL EXPERIMENTS

Tables 7.1 to 7.4 compare the exact algorithms of Section 7.3 on four classes of randomly generated problems. For the sake of uniformity with the literature (Ross and Soland (1975), Martello and Toth (1981c), Fisher, Jaikumar and Van Wassenhove (1986)), all generated instances are *minimization* problems of the form (7.5), (7.2), (7.3), (7.4). All the algorithms we consider except the Ross and Soland (1975) one, solve maximization problems, so the generated instances are transformed through (7.7), using for t the value

$$t = \max_{i \in M,\, j \in N} \{c_{ij}\} + 1.$$

The classes are

(a) w_{ij} uniformly random in [5, 25],
c_{ij} uniformly random in [1, 40],
$c_i = 9(n/m) + 0.4 \max_{i \in M} \{\sum_{j \in N_i} w_{ij}\}$ for $i = 1, \ldots, m$
(where N_i is defined as in Section 7.2.1);

(b) w_{ij} and c_{ij} as for class (a),
$c_i = 0.7(9(n/m) + 0.4 \max_{i \in M} \{\sum_{j \in N_i} w_{ij}\})$ for $i = 1, \ldots, m$;

(c) w_{ij} and c_{ij} as for class (a),
$c_i = 0.8 \sum_{j=1}^{n} w_{ij}/m$ for $i = 1, \ldots, m$;

Table 7.1 Data set (a). HP 9000/840 in seconds. Average times/Average numbers of nodes over 10 problems

		RS		MTG		FJV		MTGFJV		MTGJN	
m	n	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
	10	0.003	3	0.014	1	0.010	2	0.022	1	0.054	1
2	20	0.020	10	0.038	2	1.792	113	0.042	2	0.152	2
	30	0.018	5	0.060	2	10.089(9)	54	0.086	2	0.283	2
	10	0.006	2	0.016	1	0.005	1	0.018	1	0.019	1
3	20	0.035	18	0.124	16	5.927	293	0.100	5	1.004	15
	30	0.045	15	0.075	1	0.124	6	0.071	1	0.069	1
	10	0.017	10	0.067	7	0.056	3	0.074	3	0.387	7
5	20	0.057	19	0.258	41	6.799	320	0.190	7	1.837	41
	30	0.044	10	0.224	12	0.117	6	0.150	3	1.108	12

Table 7.2 Data set (b). HP 9000/840 in seconds. Average times/Average numbers of nodes over 10 problems

		RS		MTG		FJV		MTGFJV		MTGJN	
m	n	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
2	10	0.089	93	0.123	32	0.416	15	0.312	14	0.793	26
	20	13.906	9907	14.827	4012	79.951(5)	714	49.628(8)	630	69.573(6)	2231
	30	time limit		time limit		time limit		time limit		time limit	
3	10	0.157	165	0.228	54	0.925	69	0.539	31	1.312	41
	20	48.804(6)	32264	32.487(7)	5982	time limit		42.530(7)	489	39.231(7)	853
	30	93.695(1)	56292	63.842(6)	6824	time limit		90.755(2)	1003	82.991(3)	1108
5	10	0.314	259	0.454	94	2.094	110	0.565	33	2.739	74
	20	17.036(9)	9194	19.536	2311	92.831(3)	2043	24.755	551	75.723(3)	944
	30	74.580(3)	33854	68.075(4)	5950	81.932(2)	438	80.615(2)	925	80.587(2)	693

Table 7.3 Data set (c). HP 9000/840 in seconds. Average times/Average numbers of nodes over 10 problems

		RS		MTG		FJV		MTGFJV		MTGJN	
m	n	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
2	10	0.110	121	0.176	50	0.575	22	0.507	25	1.156	44
	20	17.608	13637	18.717	4788	96.525(4)	1242	48.471(7)	760	44.267(7)	1614
	30	98.142(1)	68620	83.363(2)	13362	time limit		96.024(1)	853	86.580(2)	1985
3	10	0.245	240	0.243	43	1.648	92	0.576	28	0.959	19
	20	47.476(6)	32727	18.215(9)	4106	time limit		33.769(8)	534	31.446(8)	826
	30	92.670(1)	52612	47.820(7)	4924	time limit		91.459(2)	951	67.374(5)	830
5	10	0.474	393	0.794	226	4.028	195	1.708	70	6.881	216
	20	61.614(6)	31328	15.698	1332	time limit		47.882	514	59.436(7)	524
	30	time limit		time limit		time limit		time limit		time limit	

Table 7.4 Data set (d). HP 9000/840 in seconds. Average times/Average numbers of nodes over 10 problems

		RS		MTG		FJV		MTGFJV		MTGJN	
m	n	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
	10	0.150	162	0.168	30	0.254	30	0.364	25	0.760	14
2	20	50.575	42321	14.344	2275	86.809(5)	3362	62.123(7)	1405	21.322	290
	30	time limit		79.582(3)	8230	time limit		97.681(1)	1475	95.702(1)	1076
	10	0.541	575	0.350	48	0.966	80	0.870	41	2.244	38
3	20	time limit		16.890	1587	time limit		95.024(2)	876	91.181(3)	801
	30	time limit		97.000(1)	6267	time limit		time limit		time limit	
	10	0.810	697	0.498	73	1.677	108	1.244	55	3.481	63
5	20	time limit		21.203(3)	7722	time limit		time limit		time limit	
	30	time limit		time limit		time limit		time limit		time limit	

(d) w_{ij} uniformly random in [1, 100],
c_{ij} uniformly random in $[w_{ij}, w_{ij} + 20]$,
$c_i = 0.8 \sum_{j=1}^{n} w_{ij}/m$ for $i = 1, \ldots, m$.

Problems of class (a) have been proposed by Ross and Soland (1975) and generally admit many feasible solutions. Problems of classes (b), (c) and (d) have tighter capacity constraints; in addition, in problems of class (d) a correlation between profits and weights (often found in real-world applications) has been introduced.

The entries in the tables give average running times (expressed in seconds) and average numbers of nodes generated in the branch-decision tree. A time limit of 100 seconds was imposed on the running time spent by each algorithm for the solution of a single instance. For data sets for which the time limit occurred, the corresponding entry gives, in brackets, the number of instances solved within 100 seconds (the average values are computed by also considering the interrupted instances). The cases where the time limit occurred for all the instances are denoted as "time limit". The following algorithms have been coded in Fortran IV and run on an HP 9000/840 computer, using option "-o" for the Fortran compiler:

RS = Algorithm of Ross and Soland (1975);

MTG = Algorithm of Martello and Toth (1981c) as described in Section 7.3;

FJV = Algorithm of Fisher, Jaikumar and Van Wassenhove (1986);

MTGFJV = Algorithm MTG with upper bound min (U_2, U_3) (see Sections 7.2.2, 7.2.3) computed at each node of the branch-decision tree;

MTGJN = Algorithm MTG with upper bound min (U_1, U_2, U_4) (see Sections 7.2.1, 7.2.2, 7.2.4) computed at each node of the branch-decision tree.

For all the algorithms, the solution of the 0-1 single knapsack problems was obtained using algorithm MT1 of Section 2.5.2.

For the computation of U_4, needed by MTGJN, the number of iterations in the subgradient optimization procedure was limited to 50—as suggested by the authors (Jörnsten and Näsberg, 1986)—for the root node, and to 10 for the other nodes. The Lagrangian multipliers were initially set to

$$\mu_{ij} = \overline{p} = \frac{1}{m\ n} \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}, \quad i \in M, j \in N$$

(as suggested by the authors) for the root node, and to the corresponding values obtained at the end of the previous computation for the other nodes. (Different choices of the number of iterations and of the initial values of the multipliers produced worse computational results.) The step used, at iteration k of the subgradient optimization procedure, to modify the current μ_{ij} values was that proposed by the authors, i.e.

$$t^k = \frac{\overline{p}}{k+1}.$$

The tables show that the fastest algorithms are RS for the "easy" instances of class (a), and MTG for the harder instances (b), (c), (d). Algorithms MTGFJV and MTGJN generate fewer nodes than MTG, but the global running times are larger (the computation of U_3 and U_4 being much heavier than that of U_1 and U_2), mainly for problems of classes (b), (c) and (d).

Algorithm FJV is much worse than the other algorithms for all data sets, contradicting, to a certain extent, the results presented for the same classes of test problems in Fisher, Jaikumar and Van Wassenhove (1986). This could be explained by observing that such results were obtained by comparing executions on different computers and using different random instances. In addition, the current implementation of MTG incorporates, for the root node, the computation of upper bound U_3.

Table 7.5 gives the performance of the Fortran IV implementation of approximate algorithm MTHG (Section 7.4) on large-size instances. The entries give average running times (expressed in seconds) and, in brackets, upper bounds on the average percentage errors. The percentage errors were computed as $100\,(U - z^h)/U$, where $U = \min(U_1, U_2, U_3, U_4)$. Only data sets (a), (b) and (c) are considered, since the computation of U for data set (d) required excessive running times. Errors of value 0.000 indicate that all the solutions found were exact. The table shows that the running times are quite small and, with few exceptions, practically independent

Table 7.5 Algorithm MTHG. HP 9000/840 in seconds. Average times (average percentage errors) over 10 problems

m	n	Data set (a)	Data set (b)	Data set (c)
5	50	0.121(0.184)	0.140(5.434)	0.136(6.822)
	100	0.287(0.063)	0.325(4.750)	0.318(5.731)
	200	0.887(0.029)	0.869(4.547)	0.852(6.150)
	500	2.654(0.012)	3.860(5.681)	3.887(6.145)
10	50	0.192(0.016)	0.225(3.425)	0.240(6.243)
	100	0.457(0.019)	0.521(5.160)	0.550(5.908)
	200	1.148(0.004)	1.271(4.799)	1.334(5.190)
	500	3.888(0.006)	5.139(5.704)	5.175(5.553)
20	50	0.393(0.062)	0.399(1.228)	0.438(6.479)
	100	0.743(0.002)	0.866(1.189)	0.888(5.187)
	200	1.693(0.008)	2.011(2.140)	2.035(4.544)
	500	2.967(0.000)	7.442(3.453)	7.351(4.367)
50	50	0.938(0.000)	0.832(0.125)	0.876(2.024)
	100	0.728(0.005)	1.792(0.175)	2.016(4.041)
	200	3.456(0.002)	3.849(0.296)	4.131(3.248)
	500	2.879(0.000)	12.613(0.517)	12.647(3.198)

of the data set. For $n = 500$ and data set (a), the first execution of MTHG (with $f_{ij} = p_{ij}$) almost always produced an optimal solution of value $z^h = U_0$, so the computing times are considerably smaller than for the other data sets. The quality of the solutions found by MTHG is very good for data set (a) and clearly worse for the other data sets, especially for small values of m. However, it is not possible to decide whether these high errors depend only on the approximate solution or also on the upper bound values. Limited experiments indicated that the error computed with respect to the optimal solution value tends to be about half that computed with respect to U.