5

Change-making problem

5.1 INTRODUCTION

In Chapter 1 the change-making problem has been presented, for the sake of uniformity, as a maximization problem with bounded variables. However, in the literature it is generally considered in minimization form and, furthermore, the main results have been obtained for the case in which the variables are unbounded. Hence we treat the bounded case in the final section of this chapter, the remaining ones being devoted to the *Change-Making Problem* (CMP) defined as follows. Given n *item types* and a *knapsack*, with

$$w_j = \textit{weight} \text{ of an item of type } j;$$

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

select a number x_j $(j = 1, \ldots, n)$ of items of each type so that the total weight is c and the total number of items is a minimum, i.e.

$$\text{minimize} \quad z = \sum_{j=1}^{n} x_j \tag{5.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j = c. \tag{5.2}$$

$$x_j \geq 0 \text{ and integer}, \quad j \in N = \{1 \ldots . n\}. \tag{5.3}$$

The problem is NP-hard also in this version, since Lueker (1975) has proved that even the feasibility question (5.2)–(5.3) is NP-complete. The problem is called "change-making" since it can be interpreted as that of a cashier having to assemble a given change, c, using the least number of coins of specified values w_j $(j = 1, \ldots, n)$ in the case where, for each value, an unlimited number of coins is available. CMP can also be viewed as an unbounded knapsack problem (Section 3.6) in which $p_j = -1$ for all j and, in the capacity constraint, strict equality is imposed. (On the other hand, imposing inequality $\sum_{j=1}^{n} w_j x_j \geq c$ gives rise to a trivial problem whose optimal solution is $x_l = \lceil c/w_l \rceil$ (where l is the item type of maximum weight) and $x_j = 0$ for $j \in N \setminus \{l\}$, since item type l "dominates" all the others in the sense of Theorem 3.2.) Note that, because of (5.2), a feasible

solution to the problem does not necessarily exist.

It is usual in the literature to consider positive weights w_j. Hence, we will also assume, without loss of generality, that

$$w_j \text{ and } c \text{ are integers;} \tag{5.4}$$

$$w_j < c \qquad \text{for } j \in N; \tag{5.5}$$

$$w_i \neq w_j \qquad \text{if } i \neq j. \tag{5.6}$$

Violation of assumption (5.4) can be handled by scaling. If assumption (5.5) is violated, then we can set $x_j = 0$ for all j such that $w_j > c$ and, if there is an item type (say k) with $w_k = c$, immediately determine an optimal solution ($x_k = 1$, $x_j = 0$ for $j \in N \setminus \{k\}$). If assumption (5.6) is violated then the two item types can be replaced by a single one. Note that, on the contrary, the assumption on the positivity of w_j ($j \in N$) produces a loss of generality, because of the equality constraint.

CMP can arise, in practice, in some classes of unidimensional cargo-loading and cutting stock problems. Consider, for example, a wall to be covered with panels: how is it possible, given the available panel lengths, to use the least possible number of panels?

In the following sections we examine lower bounds (Section 5.2), greedy solutions (Sections 5.3, 5.4), dynamic programming and branch-and-bound algorithms (Sections 5.5, 5.6), and the results of computational experiments (Section 5.7). Section 5.8 analyses the generalization of the problem to the case where, for each j, an upper bound on the availability of items of type j is given (*Bounded Change-Making Problem*).

5.2 LOWER BOUNDS

Assume, without loss of generality, that the item types satisfy

$$w_1 > w_2 > w_3 > w_j \qquad \text{for } j = 4, \ldots, n. \tag{5.7}$$

Let us consider the continuous relaxation of CMP, i.e. (5.1), (5.2) and

$$x_j \geq 0, \qquad j \in N.$$

From (5.7), its optimal solution $\bar{x}$ is straightforward ($\bar{x}_1 = c/w_1, \bar{x}_j = 0$ for $j = 2, \ldots, n$) and provides an immediate lower bound for CMP:

$$L_0 = \left\lceil \frac{c}{w_1} \right\rceil.$$

If we also impose, similarly to what has been done for the unbounded knapsack problem (Section 3.6.1), the obvious condition $\bar{x}_1 \leq \lfloor c/w_1 \rfloor$, which must hold in any integer solution, the continuous solution becomes

$$\bar{x}_1 = \left\lfloor \frac{c}{w_1} \right\rfloor,$$

$$\bar{x}_j = 0 \quad \text{for } j = 3, \ldots, n,$$

$$\bar{x}_2 = \frac{\bar{c}}{w_2},$$

where

$$\bar{c} = c(\bmod w_1). \tag{5.8}$$

This gives an improved lower bound:

$$L_1 = \left\lfloor \frac{c}{w_1} \right\rfloor + \left\lceil \frac{\bar{c}}{w_2} \right\rceil.$$

Suppose now that $\lfloor c/w_1 \rfloor$ items of type 1 and $\lfloor \bar{c}/w_2 \rfloor$ items of type 2 are initially selected, and let

$$z' = \left\lfloor \frac{c}{w_1} \right\rfloor + \left\lfloor \frac{\bar{c}}{w_2} \right\rfloor, \tag{5.9}$$

$$c' = \bar{c}(\bmod w_2). \tag{5.10}$$

In the optimal solution to CMP, either $x_2 \leq \lfloor \bar{c}/w_2 \rfloor$ or $x_2 > \lfloor \bar{c}/w_2 \rfloor$. In the former case the continuous relaxation gives a lower bound

$$L^0 = z' + \left\lceil \frac{c'}{w_3} \right\rceil, \tag{5.11}$$

while in the latter a valid lower bound is

$$L^1 = z' - 1 + \left\lceil \frac{c' + w_1}{w_2} \right\rceil, \tag{5.12}$$

since the condition implies $x_1 \leq \lfloor c/w_1 \rfloor - 1$. We have thus proved the following

Theorem 5.1 (Martello and Toth, 1980b). *The value*

$$L_2 = \min (L^0, L^1),$$

where L^0 and L^1 are defined by (5.8)–(5.12), *is a lower bound for CMP.*

Since L_1 can be re-written as $z' + \lceil c'/w_2 \rceil$, we have $L^0 \geq L_1$. Also, by noting that $L^1 = z' + \lceil (c' + w_1 - w_2)/w_2 \rceil$ and $w_1 - w_2 > 0$, we have $L^1 \geq L_1$. Hence L_1 is dominated by the new bound.

The time complexity for the computation of the above bounds is clearly $O(n)$. No sorting is in fact needed, since only the three largest weights are required.

Example 5.1

Consider the instance of CMP defined by

$n = 5;$

$(w_j) = (11, 8, 5, 4, 1);$

$c = 29.$

We obtain

$L_0 = 3.$

$L_1 = 2 + \lceil \frac{7}{8} \rceil = 3.$

$L^0 = 2 + \lceil \frac{7}{5} \rceil = 4;$

$L^1 = 2 - 1 + \lceil \frac{18}{8} \rceil = 4;$

$L_2 = 4.$ □

Lower bounds L_0. L_1 and L_2 are the respective conceptual counterparts of upper bounds U_0. U_1 and U_3 introduced for the unbounded knapsack problem (Section 3.6.1), for which we have proved that the worst case performance ratio is 2. As often occurs, however, maximization and minimization problems do not have the same theoretical properties for upper and lower bounds (see, e.g., Section 2.4). For CMP, the worst-case performance of the lower bounds above is arbitrarily bad. Consider in fact the series of instances with $n = 3$, $w_1 = k$, $w_2 = k - 1$, $w_3 = 1$ and $c = 2k - 3$ $(k > 3)$: we have $L_0 = L_1 = L_2 = 2$, while the optimal solution has value $z = k - 2$, so the ratio L_i/z $(i = 1.2$ or $3)$ can be arbitrarily close to 0 for k sufficiently large.

5.3 GREEDY ALGORITHMS

In the present section we consider both the change-making problem (5.1)–(5.3) and the generalization we obtain by associating an integer non-negative *cost* q_j with each item type $j \in N$ and changing the objective function to

$$\text{minimize} \quad z = \sum_{j=1}^{n} q_j x_j . \tag{5.13}$$

We obtain an *Unbounded Equality Constrained Min-Knapsack Problem* (UEMK). UEMK contains, as special cases:

(i) CMP, when $q_j = 1$ for all $j \in N$;

(ii) the unbounded knapsack problem (Section 3.6) in minimization form, when an extra item type $n+1$ is added, with $q_{n+1} = 0$ and $w_{n+1} = -1$.

For convenience of notation, we assume that the item types are sorted according to *decreasing* values of the cost per unit weight, i.e.

$$\frac{q_1}{w_1} \geq \frac{q_2}{w_2} \geq \ldots \geq \frac{q_n}{w_n}, \tag{5.14}$$

and note that, for CMP, this implies

$$w_1 < w_2 < \ldots < w_n .$$

A greedy algorithm for UEMK can be derived immediately from procedure GREEDYU of Section 3.6.1 as follows.

```
procedure GREEDYUM:
input: n, c, (q_j), (w_j);
output: z^g, (x_j), c̄;
begin
    c̄ := c;
    z^g := 0;
    for j := n to 1 step −1 do
        begin
            x_j := ⌊c̄/w_j⌋;
            c̄ := c̄ − w_j x_j;
            z^g := z^g + q_j x_j
        end
end.
```

The time complexity of GREEDYUM is $O(n)$, plus $O(n \log n)$ for the preliminary sorting. By replacing the last statement with

$$z^g := z^g + x_j$$

we obtain a greedy algorithm for CMP.

On return from GREEDYUM, if $\bar{c} = 0$ then z^g and (x_j) give a feasible (not necessarily optimal) solution. If $\bar{c} > 0$ then no feasible solution has been found by the procedure.

Example 5.1 (continued)

Applying GREEDYUM we obtain $(x_j) = (2.\ 0.\ 1.\ 0.\ 2)$, $z^g = 5$ and $\bar{c} = 0$, while the optimal solution has value $z = 4$ (as will be seen in Section 5.5.2). □

The case in which at least one item type has weight of value 1, has interesting properties. First, a feasible solution to the problem always exists. Furthermore, GREEDYUM always returns a feasible solution, since the iteration in which this item is considered produces $\bar{c} = 0$. The worst-case behaviour of the greedy algorithm, however, is arbitrarily bad, also with this restriction. Consider in fact the series of instances (both of CMP and UEMK) with: $n = 3$. $q_j = 1$ for all j. $w_1 = 1$. $w_2 = k$. $w_3 = k+1$ and $c = 2k > 2$, for which $z = 2$ and $z^g = k$, so the ratio z^g/z goes to infinity with k. (The absolute error produced by GREEDYUM for UEMK has been investigated by Magazine, Nemhauser and Trotter (1975), the relative error produced for CMP by Tien and Hu (1977).)

Consider now a change-making problem for the US coins, i.e. with: $n = 6$. $w_1 = 1$. $w_2 = 5$. $w_3 = 10$. $w_4 = 25$. $w_5 = 50$. $w_6 = 100$. It is not difficult to be convinced that GREEDYUM gives the optimal solution for any possible value of c (expressed in cents). Situations of this kind are analysed in the next section.

5.4 WHEN THE GREEDY ALGORITHM SOLVES CLASSES OF KNAPSACK PROBLEMS

We consider instances of CMP and UEMK in which at least one item type has weight of value 1.

For CMP, this implies that, after sorting,

$$1 = w_1 < w_2 < \ldots < w_n. \tag{5.15}$$

A weight vector $(w_1, \ldots, w_n)$ is called *canonical* if the greedy algorithm exactly solves all instances of CMP defined by the vector and any positive integer c. Chang and Gill have given the following necessary and sufficient condition for testing whether a vector is canonical.

Theorem 5.2 (Chang and Gill, 1970a) *A vector* $(w_1, \ldots, w_n)$ *satisfying* (5.15) *is canonical if and only if for all integers* c *in the range*

$$w_3 \leq c < \frac{w_n(w_n w_{n-1} + w_n - 3w_{n-1})}{w_n - w_{n-1}}$$

the greedy solution is optimal.

The proof is quite involved and will not be reported here. Furthermore, application of the theorem is very onerous, calling for optimality testing of a usually high number of greedy solutions.

Example 5.2

Consider the instance of CMP defined by

$$n = 7;$$

$$(w_j) = (1, 2, 4, 8, 10, 16).$$

This vector can be proved to be canonical. However, application of the Chang and Gill (1970a) test requires us to solve, both in an exact and greedy way, 386 instances. □

We now turn to instances of UEMK. Let j^* denote an item type such that $w_{j^*} = 1$ and $q_{j^*} = \min \{q_j : w_j = 1\}$ and note that all item types k for which $q_k/w_k \geq q_{j^*}/w_{j^*}$ are "dominated" by j^* so they can be eliminated from the instance. Hence we assume, without loss of generality, that the item types, sorted according to (5.14), satisfy

$$1 = w_1 < w_j \quad \text{for } j = 2, \ldots, n. \tag{5.16}$$

For $k = 1, \ldots, n$ and for any positive integer c, let $f_k(c)$ and $g_k(c)$ denote, respectively, the optimal and greedy solution value to

$$\text{minimize} \quad \sum_{j=1}^{k} q_j x_j$$

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

$$x_j \geq 0 \quad \text{and integer,} \quad j = 1, \ldots, k.$$

When $f_k(c) = g_k(c)$ for all c—or, more concisely, $f_k = g_k$—we say that the pair of vectors $((q_1, \ldots, q_k), (w_1, \ldots, w_k))$ is *canonical*. The following theorem provides a recursive sufficient condition for checking whether a pair of vectors is so.

Theorem 5.3 (Magazine, Nemhauser and Trotter, 1975) *Assume that* $(q_1, \ldots, q_n)$ *and* $(w_1, \ldots, w_n)$ *satisfy* (5.14) *and* (5.16). *If, for fixed* k $(1 \leq k \leq n)$, $f_k = g_k$ *and* $w_{k+1} > w_k$, *then, by defining* $m = \lceil w_{k+1}/w_k \rceil$ *and* $\gamma = mw_k - w_{k+1}$, *the following are equivalent:*

(i) $f_{k+1} = g_{k+1}$,

(ii) $f_{k+1}(mw_k) = g_{k+1}(mw_k)$,

(iii) $q_{k+1} + g_k(\gamma) \leq mq_k$.

Proof. It is obvious that (ii) follows from (i). Since $g_{k+1}(mw_k) = q_{k+1} + g_k(\gamma)$ and $f_{k+1}(mw_k) \leq mq_k$, (iii) follows from (ii). To prove that (i) follows from (iii), suppose, by absurdity, that (iii) holds but there exists a value c for which $f_{k+1}(c) < g_{k+1}(c)$. It must be $c > w_{k+1}$, since for $c < w_{k+1}$ we have $f_{k+1}(c) = f_k(c) = g_k(c) = g_{k+1}(c)$ while for $c = w_{k+1}$ we have $f_{k+1}(w_{k+1}) = g_{k+1}(w_{k+1}) = q_{k+1}$. Let $p = \lfloor c/w_k \rfloor$ and $\delta = c - pw_k$, and note that $p \geq m - 1$.

If $x_{k+1} > 0$ in an optimal solution, then $f_{k+1}(c) - g_{k+1}(c) = f_{k+1}(c - x_{k+1}w_{k+1}) - g_{k+1}(c - x_{k+1}w_{k+1})$ (since the greedy solution includes at least x_{k+1} items of type $k+1$), so we can assume that c is such that $x_{k+1} = 0$ in any optimal solution. Hence

$$f_{k+1}(c) = f_k(c) = g_k(c) = pq_k + f_k(\delta) = mq_k + (p - m)q_k + f_k(\delta). \tag{5.17}$$

From the definition of δ, by algebraic manipulation we can write $c = w_{k+1} + (p - m)w_k + \gamma + \delta$. Hence: (a) if $p \geq m$ then

$$f_{k+1}(c) < q_{k+1} + (p - m)q_k + g_k(\gamma) + f_k(\delta); \tag{5.18}$$

(b) if $p = m - 1$ then $\gamma + \delta - w_k = c - w_{k+1} > 0$ and $f_k(\gamma + \delta) = f_k(\gamma + \delta - w_k) + q_k$, so

$$f_{k+1}(c) < q_{k+1} + f_k(\gamma + \delta - w_k) \leq q_{k+1} - q_k + g_k(\gamma) + f_k(\delta),$$

showing that (5.18) holds for all $p \geq m - 1$. Combining (5.17) and (5.18) we obtain $mq_k < q_{k+1} + g_k(\gamma)$, a contradiction. □ (An alternative proof has been given by Hu and Lenard (1976).)

Theorem 5.3 is known as the "one-point theorem" since, given a canonical pair $((q_1, \ldots, q_k), (w_1, \ldots, w_k))$ and a further item, $k+1$, satisfying $q_{k+1}/w_{k+1} \geq q_k/w_k$ and $w_{k+1} > w_k$, the canonicity of $((q_1, \ldots, q_{k+1}), (w_1, \ldots, w_{k+1}))$ is guaranteed by optimality of the greedy solution for the single value $c = mw_k$. Moreover, this check does not require exact solution of the instance, since execution of the greedy algorithm for the value $c = \gamma$ (requiring $O(k)$ time) is sufficient to test condition (iii).

Given q_j, w_j $(j = 1, \ldots, n)$ satisfying (5.14) and (5.15), the following procedure checks condition (iii) for $k = 1, \ldots, n - 1$. Note that condition (5.15), always satisfied by CMP, is not necessarily satisfied by an instance of UEMK.

```
procedure MNT:
input: n.(q_j).(w_j);
output: opt;
begin
    ñ := n;
    n := 1;
    optimal := "yes";
    while n < ñ and optimal = "yes" do
        begin
            m := ⌈w_{n+1}/w_n⌉;
```

```
            c := mw_n − w_{n+1};
            call GREEDYUM;
            if q_{n+1} + z^g ≤ mq_n then n := n + 1 else optimal := "no"
        end;
    opt := n;
    n := ñ
end.
```

The time complexity of MNT is clearly $O(n^2)$. On return, we know that the pairs $((q_1,\ldots,q_k),(w_1,\ldots,w_k))$ are canonical for all $k \leq opt$. If $opt < n$, then the pair with $k = opt + 1$ is not canonical, while, for the pairs with $k > opt + 1$ the situation is undecided.

Example 5.2 (continued)

By setting $q_j = 1$ for $j = 1, \ldots, 6$, and applying MNT, we have

$n = 1, 2, 3 : m = 2, c = 0, z^g = 0;$

$n = 4 \quad : m = 2, c = 6, z^g = 2, opt = 4.$

Hence the greedy algorithm exactly solves all the instances induced by items $(1,\ldots,k)$ with $k \leq 4$, while it fails for at least one instance with $k = 5$ (see, e.g., the case with $c = 16$). The situation for $(w_1,\ldots,w_6)$ cannot be decided through procedure MNT, although the vector is canonical, as can be proved using Theorem 5.2 . □

Further characterizations of instances for which the greedy solution is optimal have been given by Chang and Korsh (1976) and Tien and Hu (1977).

5.5 EXACT ALGORITHMS

Chang and Gill (1970a) have presented a recursive procedure for the exact solution of those instances of CMP in which one item type has weight of value 1. An Algol implementation of this method has been given by Chang and Gill (1970b) and corrected by Johnson and Kernighan (1972). The resulting code is, however, highly inefficient, as shown in Martello and Toth (1977c, 1980b) through computational experiments, so no account of it will be taken in the following.

In the following sections we consider algorithms for the exact solution of CMP with no special assumption.

5.5.1 Dynamic programming

Given a pair of integers m $(1 \leq m \leq n)$ and $\hat{c}$ $(0 \leq \hat{c} \leq c)$, consider the sub-instance of CMP consisting of item types $1,\ldots,m$ and capacity $\hat{c}$, and denote

with $f_m(\hat{c})$ the corresponding optimal solution value ($f_m(\hat{c}) = \infty$ if no solution of value $\hat{c}$ exists). Then, clearly,

$$f_1(\hat{c}) = \begin{cases} \infty & \text{for all positive } \hat{c} \leq c \text{ such that } \hat{c} (\bmod w_1) \neq 0; \\ l & \text{for } \hat{c} = lw_1, \text{ with } l = 0, \ldots, \left\lfloor \dfrac{c}{w_1} \right\rfloor. \end{cases}$$

$f_m(\hat{c})$ can be computed, by considering increasing values of m from 2 to n and, for each m, increasing values of $\hat{c}$ from 0 to c, as

$$f_m(\hat{c}) = \min \left\{ f_{m-1}(\hat{c} - lw_m) + l : l \text{ integer. } 0 \leq l \leq \left\lfloor \frac{\hat{c}}{w_m} \right\rfloor \right\}.$$

The optimal solution value of CMP is then given by $f_n(c)$. The time complexity for this computation is $O(nc^2)$, the space complexity $O(nc)$.

By adapting to CMP the improved recursion proposed by Gilmore and Gomory (1965) for the unbounded knapsack problem (Section 3.6.2), we obtain

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

which reduces the time complexity to $O(nc)$. Wright (1975) has further noted that, if the items are sorted according to increasing weights, only values of $\hat{c}$ not greater than $w_m w_{m+1}$ need be considered at each stage m. In fact, w_m items of type $m + 1$ give the same weight as w_{m+1} items of type m and a better value for the objective function. A specialized dynamic programming algorithm for CMP can be found in Wright (1975).

5.5.2 Branch-and-bound

In the present section we assume that the item types are sorted so that

$$w_1 > w_2 > \ldots > w_n. \tag{5.19}$$

To our knowledge, the only branch-and-bound algorithms for CMP are those in Martello and Toth (1977c, 1980b). We give a description of the latter, which has a structure similar to that of MTU1 (Section 3.6.2), with one important difference.

As in MTU1, at a forward move associated with the jth item type, if a lower bound on the best solution obtainable is less than the best solution so far, the largest possible number of items of type j is selected. As usual, a backtracking move consists in removing one item of the last inserted type. Before backtracking on item type i, let $\hat{x}_j$ ($j = 1, \ldots, i$) be the current solution, $\hat{c} = c - \sum_{j=1}^{i} w_j \hat{x}_j$ and $\hat{z} = \sum_{j=1}^{i} \hat{x}_j$ the corresponding residual capacity and value, and z the best solution value so far. The following is then true (Martello and Toth, 1980b):

if $\hat{c} < w_i$, *the value* $l_{\hat{c}} = z - \hat{z}$ *is a lower bound on* $f_n(\hat{c})$ (= number of items needed to obtain a change $\hat{c}$ with item weights $(w_1, \ldots, w_n)$, see Section 5.5.1). In fact: (a) only item types $i+1, \ldots, n$ can produce $\hat{c}(< w_i)$, so (b) if the solution of value z has been obtained at a decision node descending from the current one, then, clearly, $l_{\hat{c}} = f_n(\hat{c})$; otherwise, at each leaf λ of the decision sub-tree descending from the current node, the lower bound, say $\hat{z} + L_\lambda$, allowed the search to be stopped, so a valid lower bound on $f_n(\hat{c})$ is $\min_\lambda \{L_\lambda\} \geq z - \hat{z} = l_{\hat{c}}$.

The consideration above leads to a dominance criterion which is particularly strong, since it allows one to fathom nodes of the decision tree basing oneself on a value depending only on the current residual capacity, independently of the variables currently fixed. In the resulting algorithm, $l_{\hat{c}}$ is defined at Step 5, and tested at Steps 2a and 5.

Also, it is useful to initially define a vector $(m_{\hat{c}})$ such that $m_{\hat{c}} = \min\{j : w_j \leq \hat{c}\}$, so that, for each residual capacity $\hat{c}$ produced by branch-and-bound, the next feasible item type is immediately known. Vectors $(l_{\hat{c}})$ and $(m_{\hat{c}})$ clearly require pseudo-polynomial space, hence, in the following implementation, their size is limited by a constant parameter γ. It is assumed that the item types are sorted according to (5.19), and that $\gamma < w_1$. (Note that vector $(m_{\hat{c}})$ can be used only after a forward move, i.e. when $\hat{c} < w_1$, while after a backtracking, say on item type i, the next feasible item type is $i+1$.)

procedure MTC1:
input: $n, c, (w_j), \gamma$;
output: $z, (x_j)$;
begin
1. [initialize]
    $z := c + 1$;
    $w_{n+1} := 1$;
    for $k := 1$ **to** n **do** $\hat{x}_k := 0$;
    compute the lower bound $L = L_2$ (Section 5.2) on the optimal solution value;
    $j := n$;
    while $j > 1$ and $w_j \leq \gamma$ **do**
        begin
            for $h := w_j$ **to** $\min(\gamma, w_{j-1} - 1)$ **do** $m_h := j$;
            $j := j - 1$
        end;
    for $h := 1$ **to** $\min(\gamma, w_n - 1)$ **do** $l_h := \infty$;
    for $h := w_n$ **to** γ **do** $l_h := 0$;
    $\hat{x}_1 := \lfloor c/w_1 \rfloor$;
    $\hat{z} := \hat{x}_1$;
    $\hat{c} := c - w_1 \hat{x}_1$;
    $j := 2$;
    if $\hat{c} > 0$ **then go to** 2a;
    $z := \hat{x}_1$;
    for $j := 1$ **to** n **do** $x_j := \hat{x}_j$;
    return ;
2a. [dominance criterion]

if $\hat{c} \leq \gamma$ **then**
  if $l_{\hat{c}} \geq z - \hat{z}$ **then go to** 5 **else** $j := m_{\hat{c}}$
else
  if $\hat{c} < w_n$ **then go to** 5 **else** find $j = \min\{k \geq j : w_k \leq \hat{c}\}$;

2. [build a new current solution]
  $y := \lfloor \hat{c}/w_j \rfloor$;
  $\tilde{c} := \hat{c} - yw_j$;
  if $z \leq \hat{z} + y + \lceil \tilde{c}/w_{j+1} \rceil$ **then go to** 5;
  if $\tilde{c} = 0$ **then go to** 4;
  if $j = n$ **then go to** 5;
3. [save the current solution]
  $\hat{c} := \tilde{c}$;
  $\hat{z} := \hat{z} + y$;
  $\hat{x}_j := y$;
  $j := j + 1$;
  go to 2a;
4. [update the best solution so far]
  $z := \hat{z} + y$;
  for $k := 1$ **to** $j - 1$ **do** $x_k := \hat{x}_k$;
  $x_j := y$;
  for $k := j + 1$ **to** n **do** $x_k := 0$;
  if $z = L$ **then return**;
5. [backtrack]
  find $i = \max\{k < j : \hat{x}_k > 0\}$;
  if no such i **then return** ;
  if $\hat{c} \leq \min(\gamma, w_i - 1)$ **then** $l_{\hat{c}} := \max(l_{\hat{c}}, z - \hat{z})$;
  $\hat{c} := \hat{c} + w_i$;
  $\hat{z} := \hat{z} - 1$;
  $\hat{x}_i := \hat{x}_i - 1$;
  if $z \leq \hat{z} + \lceil \hat{c}/w_{i+1} \rceil$ **then** (**comment**: remove all items of type i)
    begin
      $\hat{c} := \hat{c} + w_i \hat{x}_i$;
      $\hat{z} := \hat{z} - \hat{x}_i$;
      $\hat{x}_i := 0$;
      $j := i$;
      go to 5
    end;
  $j := i + 1$;
  if $\hat{c} < \gamma$ and $l_{\hat{c}} \geq z - \hat{z}$ **then go to** 5;
  if $\hat{c} - w_i \geq w_n$ **then go to** 2;
  $h := i$;
6. [try to replace one item of type i with items of type h]
  $h := h + 1$;
  if $z \leq \hat{z} + \lceil \hat{c}/w_h \rceil$ or $h > n$ **then go to** 5;
  if $\hat{c} - w_h < w_n$ **then go to** 6;
  $j := h$;
  go to 2

end.

Example 5.1 (continued)

Recall the instance

$$n = 5,$$
$$(w_j) = (11, 8, 5, 4, 1),$$
$$c = 29.$$

Figure 5.1 gives the decision-tree produced by MTC1 (with $\gamma = 10$). □

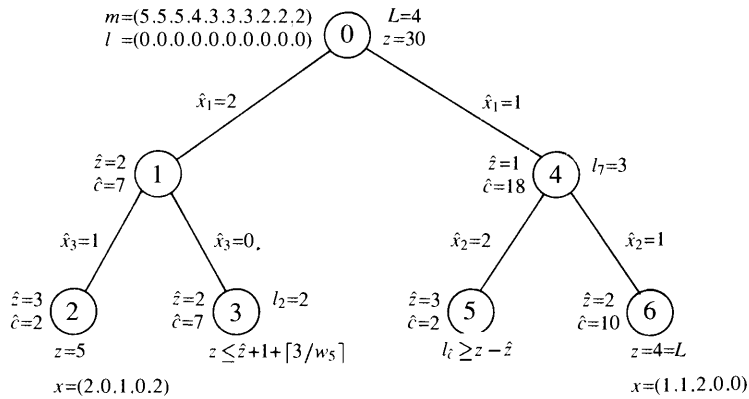

Figure 5.1 Decision-tree of procedure MTC1 for Example 5.1

The Fortran implementation of procedure MTC1 is included in that of procedure MTC2, which is described in the next section.

5.6 AN EXACT ALGORITHM FOR LARGE-SIZE PROBLEMS

Computational experiments with algorithm MTC1 (Martello and Toth, 1980b) show that, similarly to what happens for other knapsack-type problems (see Sections 2.9, 3.6.3, 4.2.3), many instances of CMP can be solved efficiently even for high values of n and, in such cases, the number of item types used for the solution is considerably smaller than n.

For CMP, however, the *core problem* does not consist of the most "favourable" item types (those with highest weight), since the equality constraint often forces the optimal solution to include also some items with medium and small weight.

An experimental analysis of the structure of the solutions found by MTC1 shows two facts: (a) the optimal solution value is often equal to the value of bound L_2 (Section 5.2); (b) many equivalent solutions usually exist. Hence we define the *core* as

$$C = \{j_1, \ldots, j_{\bar{n}}\},$$

with

j_1, j_2, j_3 = the three item types of maximum weight,

$j_4, \ldots, j_{\bar{n}}$ = any $\bar{n} - 3$ other item types,

and the *core problem* as

$$\text{minimize} \quad z = \sum_{j \in C} x_j$$

$$\text{subject to} \quad \sum_{j \in C} w_j x_j = c,$$

$$x_j \geq 0 \text{ and integer}, \qquad j \in C.$$

Noting that j_1, j_2 and j_3 are the only item types needed to compute L_2, the following procedure attempts to solve CMP by sorting only a subset of the item types. In input, $\bar{n} \leq n - 3$ is the expected size of the core, γ the parameter needed by MTC1.

```
procedure MTC2:
input: n, c, (w_j), γ, n̄;
output: z, (x_j);
begin
    determine the three item types (j1, j2, j3) of maximum weight;
    compute lower bound L2;
    C := {1, ..., n̄} ∪ {j1, j2, j3};
    sort the item types in C in order of decreasing weights;
    solve the core problem exactly, using MTC1, and let z and (x_j) define the
      solution;
    if z = L2 then for each j ∈ {1, ..., n}\C do x_j := 0
    else
        begin
            sort item types 1, ..., n in order of decreasing weights;
            solve CMP exactly, using MTC1, and let z and (x_j) define the
              solution
        end
end.
```

"Good" values for $\bar{n}$ and γ were experimentally determined as

$$\bar{n} = \min\left(n, \max\left(500, \left\lfloor \frac{n}{20} \right\rfloor\right)\right),$$

$$\gamma = \min(10\,000, w_1 - 1).$$

The Fortran implementation of algorithm MTC2 is included in the present volume.

5.7 COMPUTATIONAL EXPERIMENTS

In the present section we analyse the computational behaviour of the Fortran IV implementations of algorithms for CMP on data sets having

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

In Table 5.1 we compare the dynamic programming approach of Wright (Section 5.5.1), algorithm MTC1 (Section 5.5.2) and the approximate algorithm GREEDYUM (Section 5.3) on problems with $M = 4n$, for two values of c. The recursive approach of Chang and Gill (Section 5.5) is not considered since computational experiments (Martello and Toth, 1980b) showed that it is very much slower than the Wright algorithm. For each value of n and c, we generated 100 problems admitting of a feasible solution. Each algorithm had a time limit of 250 seconds to solve them. The entries give the average running times (including sorting times) obtained by the three algorithms on a CDC Cyber 730, the percentages of approximate solutions which are sub-optimal, infeasible and optimal, respectively, and the average running time of MTC1 when GREEDYUM finds the optimum. The table shows that MTC1 clearly dominates the Wright algorithm. The greedy algorithm is about twice as fast as MTC1, but the quality of its solutions is rather poor. In addition, for the instances in which it gives the optimal solution, the running

Table 5.1 w_j uniformly random in $[1, 4n]$. CDC-Cyber 730 in seconds. Average values over 100 feasible problems

					Greedy solutions			
c	n	Wright (time)	MTC1 (time)	Greedy (time)	Sub-optimal (%)	Not feasible (%)	Optimal (%)	MTC1 When Greedy is optimal (time)
	25	0.135	0.006	0.002	35	43	22	0.004
	50	0.451	0.011	0.006	35	38	27	0.009
$10n$	100	1.612	0.022	0.013	35	47	18	0.018
	200	time	0.045	0.029	40	42	18	0.035
	500	—	0.119	0.078	42	35	23	0.098
	1 000	—	0.241	0.155	33	40	27	0.202
	25	0.166	0.006	0.002	35	43	22	0.004
	50	0.518	0.011	0.006	32	45	23	0.009
$\sum_{j=1}^{n} w_j$	100	1.768	0.022	0.013	29	44	27	0.018
	200	time	0.045	0.030	40	37	27	0.035
	500	—	0.121	0.078	29	43	28	0.096
	1 000	—	0.240	0.154	24	41	35	0.204

time of MTC1 is only slightly higher. The running times of all the algorithms are practically independent of the value of c.

Table 5.2 gives the average times obtained, for larger problems, by MTC1 and MTC2 (Section 5.6) on an HP 9000/840 (with option "-o" for the Fortran compiler). The problems, not necessarily feasible, were generated with $M = 4n$ and $c = 0.5 \sum_{j=1}^{n} w_j$. The sorting times, included in the entries, are also separately given. The table shows that, for $n \geq 1\,000$, i.e. when the core problem is introduced, MTC2 is considerably faster than MTC1.

Table 5.3 analyses the behaviour of MTC2 when M varies, and shows that higher values of M tend to produce harder problems. This can be explained by noting that increasing the data scattering makes it more difficult to satisfy the equality constraint. Hence, in order to evaluate MTC1 and MTC2 on difficult problems, we set $M = 10^5$ for all values of n. Table 5.4 confirms the superiority of MTC2 also

Table 5.2 w_j uniformly random in $[1, 4n]$; $c = 0.5 \sum_{j=1}^{n} w_j$. HP 9000/840 in seconds. Average times over 20 problems

n	Sorting	MTC1	MTC2
50	0.003	0.007	0.003
100	0.006	0.010	0.008
200	0.013	0.021	0.019
500	0.036	0.054	0.051
1 000	0.079	0.114	0.064
2 000	0.165	0.237	0.087
5 000	0.468	0.595	0.121
10 000	0.963	1.198	0.179
20 000	2.073	12.860	0.370

Table 5.3 Algorithm MTC2. w_j uniformly random in $[1, M]$; $c = 0.5 \sum_{j=1}^{n} w_j$. HP 9000/840 in seconds. Average times over 20 problems

n	$M = 4n$	$M = 8n$	$M = 12n$
100	0.008	0.013	0.017
1 000	0.064	0.081	0.096
10 000	0.179	0.309	4.743

Table 5.4 w_j uniformly random in $[1, 10^5]$; $c = 0.5 \sum_{j=1}^{n} w_j$. HP 9000/840 in seconds. Average times over 20 problems

n	Sorting	MTC1	MTC2
50	0.004	1.205	1.172
100	0.007	0.754	0.744
200	0.012	0.862	0.855
500	0.037	2.321	2.306
1 000	0.082	3.078	1.098
2 000	0.171	7.778	1.654
5 000	0.456	11.141	0.810
10 000	0.948	time	1.939
20 000	2.124	—	5.480

for this generation. Note that, for $n \geq 1\,000$, the time difference is considerably higher than the sorting time, indicating that MTC2 takes advantage of the lesser number of item types also in the branch-and-bound phase. For $n = 10\,000$, MTC1 could not solve the generated problems within 250 seconds.

5.8 THE BOUNDED CHANGE-MAKING PROBLEM

The *Bounded Change-Making Problem* (BCMP) is the generalization of CMP we obtain by introducing

$$b_j = \text{upper bound on the availability of items of type } j,$$

and formulating the problem as

$$\text{minimize} \quad z = \sum_{j=1}^{n} x_j \tag{5.20}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j = c, \tag{5.21}$$

$$0 \leq x_j \leq b_j \text{ and integer}, \qquad j = 1, \ldots, n. \tag{5.22}$$

We will maintain the assumptions made in Section 5.1. In addition we will assume, without loss of generality, that values b_j (positive integer for all j) satisfy

$$\sum_{j=1}^{n} b_j w_j > c, \tag{5.23}$$

$$b_j w_j \leq c, \qquad j = 1, \ldots, n. \tag{5.24}$$

Violation of assumption (5.23) produces an infeasible or trivial problem, while for each j not satisfying (5.24), we can replace b_j with $\lfloor c/w_j \rfloor$.

By assuming that the item types are sorted so that

$$w_1 > w_2 > \ldots > w_n, \tag{5.25}$$

the continuous relaxation of BCMP can easily be solved, as for the bounded knapsack problem, through a straightforward adaptation of Theorem 2.1. Define the *critical item type* s as

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

and

$$\overline{c} = c - \sum_{j=1}^{s-1} b_j w_j. \tag{5.26}$$

Then the optimal solution $\overline{x}$ of the continuous relaxation is

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

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

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

and the corresponding value produces a lower bound for BCMP:

$$LB_1 = \sum_{j=1}^{s-1} b_j + \left\lceil \frac{\overline{c}}{w_s} \right\rceil.$$

A tighter bound, conceptually close to bound L_2 of Section 5.2, can be obtained by noting that, in the optimal solution, either $x_s \leq \lfloor \overline{c}/w_s \rfloor$ or $x_s > \lfloor \overline{c}/w_s \rfloor$. By defining

$$z' = \sum_{j=1}^{s-1} b_j + \left\lfloor \frac{\overline{c}}{w_s} \right\rfloor, \tag{5.27}$$

$$c' = \overline{c} \pmod{w_s}, \tag{5.28}$$

the respective lower bounds are

$$LB^0 = z' + \left\lceil \frac{c'}{w_{s+1}} \right\rceil, \tag{5.29}$$

$$LB^1 = z' - 1 + \left\lceil \frac{c' + w_{s-1}}{w_s} \right\rceil. \tag{5.30}$$

Hence,

Theorem 5.4 (Martello and Toth, 1977c) *The value*

$$LB_2 = \min (LB^0, LB^1),$$

where LB^0 and LB^1 are defined by (5.26)–(5.30), *is a lower bound for BCMP.*

LB_2 clearly dominates LB_1, since $LB_1 = z' + \lceil c'/w_s \rceil \leq LB^0$ and $LB^1 = z' + \lceil (c' + w_{s-1} - w_s)/w_s \rceil \geq LB_1$. The time complexity for the computation of LB_1

or LB_2 is $O(n)$ if the item types are already sorted according to (5.25). If this is not the case, the computation can still be done in $O(n)$ time, through an adaptation of procedure CRITICAL_ITEM of Section 2.2.2.

A greedy algorithm for BCMP immediately derives from procedure GREEDYUM of Section 5.3, by replacing the definition of x_j and z^g with $x_j := \min(\lfloor \bar{c}/w_j \rfloor, b_j)$ and $z^g := z^g + x_j$, respectively. In this case too, the worst-case behaviour is arbitrarily bad, as shown by the counterexample of Section 5.3 with $b_1 = k$, $b_2 = 2$, $b_3 = 1$. To our knowledge, no further theoretical result on the behaviour of greedy algorithms for BCMP is known.

Exact algorithms for BCMP can also be obtained easily from those for CMP. In particular, a branch-and-bound algorithm MTCB derives from procedure MTC1 of Section 5.5.2 as follows. Apart from straightforward modifications at Step 1 ($b_{n+1} := +\infty$; $L = LB_2$; $\hat{x}_1 := \min(\lfloor c/w_1 \rfloor, b_1)$; **if** $\hat{c} > 0$ **then go to** 2), the main modification concerns Steps 2 and 4. For BCMP it is worth building a new current solution by inserting as many items of types $j, j+1, \ldots$ as allowed by their bounds, until the first is found whose bound cannot be reached. In order to avoid useless backtrackings, this solution is saved only if its value does not represent the minimum which can be obtained with item type j. The altered steps are then as follows.

2. [build a new current solution]
   $y' := 0$;
   $\tilde{c} := \hat{c}$;
   $i := j - 1$;
   repeat
      $i := i + 1$;
      $y := \min(\lfloor \tilde{c}/w_i \rfloor, b_i)$;
      $y' := y' + y$;
      $\tilde{c} := \tilde{c} - y w_i$;
      if $y = b_i$ **then** $\overline{w} := w_{i+1}$ **else** $\overline{w} := w_i$;
      if $z \leq \hat{z} + y' + \lceil \tilde{c}/\overline{w} \rceil$ **then go to** 5;
      if $\tilde{c} = 0$ **then go to** 4;
      if $i = n$ **then go to** 5
   until $y < b_i$;
   $\hat{z} := \hat{z} + (y' - y)$;
   for $k := j$ **to** $i - 1$ **do** $\hat{x}_k := b_k$;
   $j := i$;

4. [update the best solution so far]
   $z := \hat{z} + y'$;
   for $k := 1$ **to** $j - 1$ **do** $x_k := \hat{x}_k$;
   for $k := j$ **to** $i - 1$ **do** $x_k := b_j$;
   $x_i := y$;
   for $k := i + 1$ **to** n **do** $x_k := 0$;
   if $z = L$ **then return**;

The Fortran code of algorithm MTCB is included in the present volume.

Table 5.5 Algorithm MTCB. $c = 0.5\sum_{j=1}^{n} w_j$. HP 9000/840 in seconds. Average times over 20 problems

	w_j uniformly random in $[1, 4n]$		w_j uniformly random in $[1, 10^5]$	
n	b_j in [1, 5]	b_j in [1, 10]	b_j in [1, 5]	b_j in [1, 10]
50	0.009	0.010	1.646	1.442
100	0.016	0.019	1.230	1.033
200	0.038	0.036	1.073	0.934
500	0.100	0.099	1.233	2.051
1 000	0.213	0.210	9.894	11.377
2 000	0.453	0.449	6.145	20.145
5 000	1.207	1.201	18.622	35.799
10 000	2.429	2.377	—	—

Table 5.5 gives the results of computational experiments performed on the data generation of Tables 5.2 and 5.4, with values b_j uniformly random in ranges [1, 5] and [1, 10]. The solution of BCMP appears harder than that of CMP. The times in the first two columns are approximately twice as high as those obtained by MTC1 in Table 5.2. Increasing the range of values b_j did not alter the difficulty. The times in the third column are higher than those of MTC1 in Table 5.4. Larger values of b_j considerably increased in this case the difficulty, especially for large values of n.