3

Bounded knapsack problem

3.1 INTRODUCTION

The *Bounded Knapsack Problem* (BKP) is: given n *item types* and a *knapsack*, with

p_j = *profit* of an item of type j;

w_j = *weight* of an item of type j;

b_j = *upper bound* on the availability of items of type j;

c = *capacity* of the knapsack,

select a number x_j $(j = 1, \dots, n)$ of items of each type so as to

$$\text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{3.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \le c, \tag{3.2}$$

$$0 \le x_j \le b_j \text{ and integer}, \quad j \in N = \{1, \dots, n\}. \tag{3.3}$$

BKP is a generalization of the 0-1 knapsack problem (Chapter 2), in which $b_j = 1$ for all $j \in N$.

We will assume, without loss of generality, that

$$p_j, w_j, b_j \text{ and } c \text{ are positive integers,} \tag{3.4}$$

$$\sum_{j=1}^{n} b_j w_j > c, \tag{3.5}$$

$$b_j w_j \le c \text{ for } j \in N. \tag{3.6}$$

Violation of assumption (3.4) can be handled through a straightforward adaptation of the Glover (1965) method used for the 0-1 knapsack problem

(Section 2.1). If assumption (3.5) is violated then we have the trivial solution $x_j = b_j$ for all $j \in N$, while for each j violating (3.6) we can replace b_j with $\lfloor c/w_j \rfloor$. Also, the way followed in Section 2.1 to transform minimization into maximization forms can be immediately extended to BKP.

Unless otherwise specified, we will suppose that the item types are ordered so that

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n} \tag{3.7}$$

A close connection between the bounded and the 0-1 knapsack problems is self-evident, so all the mathematical and algorithmic techniques analysed in Chapter 2 could be extended to the present case. The literature on BKP, however, is not comparable to that on the binary case, especially considering the last decade. The main reason for such a phenomenon is, in our opinion, the possibility of transforming BKP into an equivalent 0-1 form with a generally limited increase in the number of variables, and hence effectively solving BKP through algorithms for the 0-1 knapsack problem.

In the following sections we give the transformation technique (Section 3.2) and consider in detail some of the basic results concerning BKP (Section 3.3). The algorithmic aspects of the problem are briefly examined in Section 3.4. We do not give detailed descriptions of the algorithms since the computational results of Section 3.5 show that the last generation of algorithms for the 0-1 knapsack problem, when applied to transformed instances of BKP, outperforms the (older) specialized algorithms for the problem.

The final section is devoted to the special case of BKP in which $b_j = +\infty$ for all $j \in N$ *(Unbounded Knapsack Problem)*. For this case, interesting theoretical results have been obtained. In addition, contrary to what happens for BKP, specialized algorithms usually give the best results.

3.2 TRANSFORMATION INTO A 0-1 KNAPSACK PROBLEM

The following algorithm transforms a BKP, as defined by (3.1)–(3.3), into an equivalent 0-1 knapsack problem with

$$\begin{aligned} \hat{n} &= \text{ number of variables;} \\ (\hat{p}_j) &= \text{ profit vector;} \\ (\hat{w}_j) &= \text{ weight vector;} \\ \hat{c} = c &= \text{ capacity.} \end{aligned}$$

For each item-type j of BKP, we introduce a series of $\lfloor \log_2 b_j \rfloor$ items, whose profits and weights are, respectively, (p_j, w_j), $(2p_j, 2w_j)$, $(4p_j, 4w_j)$, ... , and one item such that the total weight (resp. profit) of the new items equals $b_j w_j$ (resp. $b_j p_j$).

```
procedure TB01:
input: $n, (p_j), (w_j), (b_j)$;
output: $\hat{n}, (\hat{p}_j), (\hat{w}_j)$;
begin
   $\hat{n} := 0$;
   for $j := 1$ to $n$ do
      begin
         $\beta := 0$;
         $k := 1$;
         repeat
            if $\beta + k > b_j$ then $k := b_j - \beta$;
            $\hat{n} := \hat{n} + 1$;
            $\hat{p}_{\hat{n}} := kp_j$;
            $\hat{w}_{\hat{n}} := kw_j$;
            $\beta := \beta + k$;
            $k := 2k$
         until $\beta = b_j$
      end
end.
```

The transformed problem has $\hat{n} = \sum_{j=1}^{n} \lceil \log_2(b_j + 1) \rceil$ binary variables, hence $O(\hat{n})$ gives the time complexity of the procedure. To see that the transformed problem is equivalent to the original one, let $\hat{x}_{j_1}, \ldots, \hat{x}_{j_q}$ ($q = \lceil \log_2(b_j + 1) \rceil$) be the binary variables introduced for x_j and notice that item j_h corresponds to n_h items of type j, where

$$n_h = \begin{cases} 2^{h-1} & \text{if } h < q; \\ b_j - \sum_{i=1}^{q-1} 2^{i-1} & \text{if } h = q. \end{cases}$$

Hence $x_j = \sum_{h=1}^{q} n_h \hat{x}_{j_h}$ can take any integer value between 0 and b_j.

Notice that the transformation introduces 2^q binary combinations, i.e. $2^q - (b_j + 1)$ redundant representations of possible x_j values (the values from n_q to $2^{q-1} - 1$ have a double representation). Since, however, q is the minimum number of binary variables needed to represent the integers from 0 to b_j, any alternative transformation must introduce the same number of redundancies.

Example 3.1

Consider the instance of BKP defined by

$n = 3$;

$(p_j) = (10,\ 15,\ 11)$;

$(w_j) = (\ 1,\ \ 3,\ \ 5)$;

$(b_j) = (\ 6,\ \ 4,\ \ 2)$;

$c = 10$.

Applying TB01, we get the equivalent 0-1 form:

$$\hat{n} = 8;$$
$$(\hat{p}_j) = (10, 20, 30, 15, 30, 15, 11, 11);$$
$$(\hat{w}_j) = (1, 2, 3, 3, 6, 3, 5, 5).$$

Items 1 to 3 correspond to the first item type, with double representation of the value $x_1 = 3$. Items 4 to 6 correspond to the second item type, with double representation of the values $x_2 = 1$, $x_2 = 2$ and $x_2 = 3$. Items 7 and 8 correspond to the third item type, with double representation of the value $x_3 = 1$. □

3.3 UPPER BOUNDS AND APPROXIMATE ALGORITHMS

3.3.1 Upper bounds

The optimal solution $\overline{x}$ of the continuous relaxation of BKP, defined by (3.1), (3.2) and

$$0 \le x_j \le b_j, \qquad j \in N,$$

can be derived in a straightforward way from Theorem 2.1. Assume that the items are sorted according to (3.7) and let

$$s = \min \left\{ j : \sum_{i=1}^{j} b_i w_i > c \right\} \tag{3.8}$$

be the *critical item type*. Then

$$\overline{x}_j = b_j \qquad \text{for } j = 1, \dots, s-1,$$
$$\overline{x}_j = 0 \qquad \text{for } j = s+1, \dots, n.$$
$$\overline{x}_s = \frac{\overline{c}}{w_s},$$

where

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

Hence the optimal continuous solution value is

$$\sum_{j=1}^{s-1} b_j p_j + \overline{c}\frac{p_s}{w_s},$$

and an upper bound for BKP is

$$U_1 = \sum_{j=1}^{s-1} b_j p_j + \left\lfloor \bar{c} \frac{p_s}{w_s} \right\rfloor . \tag{3.9}$$

A tighter bound has been derived by Martello and Toth (1977d) from Theorem 2.2. Let

$$z' = \sum_{j=1}^{s-1} b_j p_j + \left\lfloor \frac{\bar{c}}{w_s} \right\rfloor p_s \tag{3.10}$$

be the total profit obtained by selecting b_j items of type j for $j = 1, \dots, s-1$, and $\lfloor \bar{x}_s \rfloor$ items of type s. The corresponding residual capacity is

$$c' = \bar{c} - \left\lfloor \frac{\bar{c}}{w_s} \right\rfloor w_s .$$

Then

$$U^0 = z' + \left\lfloor c' \frac{p_{s+1}}{w_{s+1}} \right\rfloor \tag{3.11}$$

is an upper bound on the solution value we can obtain if no further items of type s are selected, while selecting at least one additional item of this type produces upper bound

$$U^1 = z' + \left\lfloor p_s - (w_s - c') \frac{p_{s-1}}{w_{s-1}} \right\rfloor . \tag{3.12}$$

Hence

$$U_2 = \max\,(U^0 . U^1) \tag{3.13}$$

is an upper bound for BKP. Since from (3.9) we can write $U_1 = z' + \lfloor c' p_s / w_s \rfloor$, $U^0 \le U_1$ is immediate, while $U^1 \le U_1$ is proved by the same algebraic manipulations as those used in Theorem 2.2 (ii). $U_2 \le U_1$ then follows.

The time complexity for the computation of U_1 or U_2 is $O(n)$ if the item types are already sorted. If this is not the case, the computation can still be done in $O(n)$ time through an immediate adaptation of procedure CRITICAL_ ITEM of Section 2.2.2.

Determining the continuous solution of BKP in 0-1 form still produces bound U_1. The same does not hold for U_2, since (3.11) and (3.12) explicitly consider the nature of BKP hence U^0 and U^1 are tighter than the corresponding values obtainable from the 0-1 form.

Example 3.1 (continued)

The critical item type is $s = 2$. Hence

$$U_1 = 60 + \left\lfloor 4 \, \frac{15}{3} \right\rfloor = 80.$$

$$U^0 = 75 + \left\lfloor 1 \frac{11}{5} \right\rfloor = 77;$$

$$U^1 = 75 + \left\lfloor 15 - 2 \frac{10}{1} \right\rfloor = 70;$$

$$U_2 = 77.$$

Considering the problem in 0-1 form and applying (2.10) and (2.16), we would obtain $U_1 = U_2 = 80$. □

Since $U_2 \le U_1 \le z' + p_s \le 2z$, the worst-case performance ratio of U_1 and U_2 is at most 2. To see that $\rho(U_1) = \rho(U_2) = 2$, consider the series of problems with $n = 3$, $p_j = w_j = k$ and $b_j = 1$ for all j, and $c = 2k - 1$: we have $U_1 = U_2 = 2k - 1$ and $z = k$, so U_1/z and U_2/z can be arbitrarily close to 2 for k sufficiently large.

All the bounds introduced in Section 2.3 for the 0-1 knapsack problem can be generalized to obtain upper bounds for BKP. This could be done either in a straightforward way, by applying the formulae of Section 2.3 to BKP in 0-1 form (as was done for U_1) or, better, by exploiting the peculiar nature of the problem (as was done for U_2). This second approach, not yet dealt with in the literature, could be a promising direction of research.

3.3.2 Approximate algorithms

Value z' defined by (3.10) is an immediate feasible solution value for BKP. Let z be the optimal solution value. Then the absolute error $z - z'$ is bounded by p_s (since $z' \le z \le U_1 \le z' + p_s$), while the ratio z'/z can be arbitrarily close to 0 (consider, e.g., $n = 2$, $p_1 = w_1 = 1$, $p_2 = w_2 = k$, $b_1 = b_2 = 1$ and $c = k$, for k sufficiently large). The worst-case performance ratio, however, can be improved to $1/2$ by computing (still in $O(n)$ time)

$$z^h = \max\,(z', p_s)$$

as the approximate solution value. In fact, $z \le z' + p_s \le 2z^h$, and a tightness example is: $n = 2$, $p_1 = w_1 = 1$, $p_2 = w_2 = k$, $b_1 = 1$, $b_2 = 2$ and $c = 2k$, for k sufficiently large.

If the item types are sorted according to (3.7), a more effective *greedy algorithm* is the following:

```
procedure GREEDYB:
input: $n, c, (p_j), (w_j), (b_j)$;
output: $z^g, (x_j)$;
begin
    $\bar{c} := c$;
    $z^g := 0$;
    $j^* := 1$;
    for $j := 1$ to $n$ do
        begin
            $x_j :=$ min$(\lfloor \bar{c}/w_j \rfloor . b_j)$;
            $\bar{c} := \bar{c} - w_j x_j$;
            $z^g := z^g + p_j x_j$;
            if $b_j p_j > b_{j^*} p_{j^*}$ then $j^* := j$
        end;
    if $b_{j^*} p_{j^*} > z^g$ then
        begin
            $z^g := b_{j^*} p_{j^*}$;
            for $j := 1$ to $n$ do $x_j := 0$;
            $x_{j^*} := b_{j^*}$
        end
end.
```

The worst-case performance ratio is $\frac{1}{2}$, since trivially $z^g \geq z^h$ and the series of problems with $n = 3$, $p_1 = w_1 = 1, p_2 = w_2 = p_3 = w_3 = k$, $b_1 = b_2 = b_3 = 1$ and $c = 2k$ proves the tightness. The time complexity is clearly $O(n)$, plus $O(n\log n)$ for sorting.

Transforming BKP into an equivalent 0-1 problem and then applying any of the *polynomial-time*, or *fully polynomial-time approximation schemes* of Section 2.8, we obtain approximate solutions obeying the worst-case bounds defined for such schemes. In fact the two formulations of any instance have, of course, the same optimal value, and the solution determined by the scheme for the 0-1 formulation preserves feasibility and value for the bounded formulation. Hence the worst-case performance ratio is maintained. The time and space complexities of the resulting schemes are given by those in Section 2.8, with n replaced by $\hat{n} = \sum_{j=1}^{n} \lceil \log_2(b_j + 1) \rceil$.

In this case too, better results could be obtained by defining approximation schemes explicitly based on the specific structure of BKP.

3.4 EXACT ALGORITHMS

In this section we briefly outline the most important algorithms from the literature for the exact solution of BKP. The reason for not giving a detailed description of these methods is the fact that they are generally useless for effective solution of the problem. In fact, the high level of sophistication of the algorithms for the 0-1 knapsack problem has not been followed in the algorithmic approach to BKP, so the most effective way to solve bounded knapsack problems nowadays is to transform them into 0-1 form and then apply one of the algorithms of Section 2.9. (This is confirmed by the experimental results we present in the next section.) Of course, a possible direction of research could be the definition of more effective specific algorithms for BKP through adaptation of the results of Chapter 2.

3.4.1 Dynamic programming

Let $f_m(\hat{c})$ denote the optimal solution value of the sub-instance of BKP defined by item types $1, \dots, m$ and capacity $\hat{c}$ $(1 \le m \le n.\ 0 \le \hat{c} \le c)$. Clearly

$$f_1(\hat{c}) = \begin{cases} 0 & \text{for } \hat{c} = 0, \dots, w_1 - 1; \\ p_1 & \text{for } \hat{c} = w_1 . \dots, 2w_1 - 1; \\ \cdots & \\ (b_1 - 1)p_1 & \text{for } \hat{c} = (b_1 - 1)w_1, \dots . b_1 w_1 - 1; \\ b_1 p_1 & \text{for } \hat{c} = b_1 w_1 . \dots, c. \end{cases}$$

$f_m(\hat{c})$ can then 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

$$f_m(\hat{c}) = \max\{ f_{m-1}(\hat{c} - lw_m) + lp_m : l \text{ integer},\ 0 \le l \le \min(b_m, \lfloor \hat{c}/w_m \rfloor)\}.$$

The optimal solution value of BKP is given by $f_n(c)$. For each m, $O(cb_m)$ operations are necessary to compute $f_m(\hat{c})$ $(\hat{c} = 0, \dots, c)$. Hence the overall time complexity for solving BKP is $O(c \sum_{m=1}^{n} b_m)$, i.e. $O(nc^2)$ in the worst case. The space complexity is $O(nc)$, since the solution vector corresponding to each $f_m(\hat{c})$ must also be stored.

The basic recursion above has been improved on, among others, by Gilmore and Gomory (1966) and Nemhauser and Ullmann (1969). Dynamic programming, however, can only solve problems of very limited size. (Nemhauser and Ullmann (1969) report that their algorithm required 74 seconds to solve, on an IBM-7094, a problem instance with $n = 50$ and $b_j = 2$ for each j.)

3.4.2 Branch-and-bound

Martello and Toth (1977d) adapted procedure MT1 of Section 2.5.2 to BKP. The resulting depth-first branch-and-bound algorithm, which incorporates upper bound U_2 of Section 3.3.1, is not described here, but could easily be derived from procedure MTU1 presented in Section 3.6.2 for the unbounded knapsack problem. (See also a note by Aittoniemi and Oehlandt (1985).)

Ingargiola and Korsh (1977) presented a reduction algorithm related to the one in Ingargiola and Korsh (1973) (Section 2.7) and imbedded it into a branch-search algorithm related to the one in Greenberg and Hegerich (1970) (Section 2.5). (See also a note by Martello and Toth (1980c).)

Bulfin, Parker and Shetty (1979) have proposed a different branch-and-bound strategy, incorporating penalties in order to improve the bounding phase.

Aittoniemi (1982) gives an experimental comparison of the above algorithms, indicating the Martello and Toth (1977d) one as the most effective. As already mentioned, however, all these methods are generally outperformed by algorithm MT2 (Section 2.9.3) applied to the transformed 0-1 instance. The Fortran implementation of this algorithm (MTB2) is included in the present volume.

3.5 COMPUTATIONAL EXPERIMENTS

In Tables 3.1, 3.2 and 3.3 we analyse the experimental behaviour of exact and approximate algorithms for BKP through data sets similar to those used for the 0-1 knapsack problem, i.e.:

uncorrelated: p_j and w_j uniformly random in [1,1000];

weakly correlated: w_j uniformly random in [1,1000],
p_j uniformly random in $[w_j - 100, w_j + 100]$;

strongly correlated: w_j uniformly random in [1,1000],
$p_j = w_j + 100$.

For all data sets, the values b_j are uniformly random in [5,10], and c is set to $0.5 \sum_{j=1}^{n} b_j w_j$ (so about half of the items are in the optimal solution).

The tables compare the Fortran IV implementations of the following methods:

Table 3.1 Uncorrelated problems: p_j and w_j uniformly random in [1,1000], b_j uniformly random in [5,10]; $c = 0.5 \sum_{j=1}^{n} b_j w_j$. HP 9000/840 in seconds. Average times (average percentage errors) over 20 problems

	MTB	IK	MTB2	MTB2 approximate	GREEDYB
n	time	time	time	time (% error)	time (% error)
25	0.034	0.022	0.023	0.011(0.09851)	0.001(0.09721)
50	0.121	0.115	0.049	0.020(0.04506)	0.005(0.04775)
100	0.464	0.149	0.084	0.031(0.02271)	0.012(0.01354)
200	1.761	0.462	0.143	0.061(0.01166)	0.023(0.00809)
500	9.705	5.220	0.395	0.158(0.00446)	0.065(0.00246)
1 000	36.270	11.288	0.583	0.324(0.00079)	0.138(0.00071)
2 000	88.201	33.490	1.107	0.649(0.00097)	0.272(0.00033)
5 000	159.213	106.550	2.272	1.585(0.00028)	0.745(0.00008)
10 000	—	—	3.599	3.055(0.00031)	1.568(0.00003)
20 000	—	—	6.689	6.195(0.00011)	3.332(0.00001)
30 000	—	—	9.445	9.692(0.00010)	5.144(0.00000)
40 000	—	—	14.119	13.443(0.00003)	7.080(0.00000)
50 000	—	—	14.836	15.298(0.00005)	8.942(0.00000)

Table 3.2 Weakly correlated problems: w_j uniformly random in [1,1000], p_j in $[w_j - 100, w_j + 100]$, b_j uniformly random in [5,10]; $c = 0.5 \sum_{j=1}^{n} b_j w_j$. HP 9000/840 in seconds. Average times (average percentage errors) over 20 problems

	MTB	IK	MTB2	MTB2 approximate	GREEDYB
n	time	time	time	time (% error)	time (% error)
25	0.051	0.206	0.075	0.012(0.08072)	0.001(0.13047)
50	0.150	0.855	0.199	0.019(0.03975)	0.007(0.04214)
100	0.478	3.425	0.207	0.037(0.01384)	0.014(0.01374)
200	1.350	8.795	0.354	0.061(0.00901)	0.021(0.00461)
500	6.232	25.840	0.532	0.147(0.00414)	0.057(0.00126)
1 000	16.697	59.182	0.574	0.292(0.00228)	0.125(0.00054)
2 000	39.707	57.566	0.810	0.568(0.00242)	0.265(0.00015)
5 000	131.670	131.212	1.829	1.572(0.00062)	0.725(0.00004)
10 000	—	—	3.359	3.052(0.00037)	1.572(0.00001)
20 000	—	—	6.973	6.633(0.00021)	3.293(0.00000)
30 000	—	—	9.785	9.326(0.00016)	5.089(0.00000)
40 000	—	—	6435.178	12.182(0.00017)	6.966(0.00000)
50 000	—	—	—	15.473(0.00010)	8.533(0.00000)

Table 3.3 Strongly correlated problems: w_j uniformly random in [1,1000], $p_j = w_j + 100$, b_j uniformly random in [5,10]; $c = 0.5 \sum_{j=1}^{n} b_j w_j$. HP 9000/840 in seconds. Average times (average percentage errors) over 20 problems

	MTB	IK	MTB2	MTB2 approximate	GREEDYB
n	time	time	time	time (% error)	time (% error)
25	3.319	216.864	23.091	0.012(0.36225)	0.002(0.62104)
50	279.782	—	4513.810	0.018(0.14509)	0.005(0.22967)
100	—	—	—	0.037(0.14295)	0.010(0.16482)
200	—	—	—	0.066(0.07570)	0.023(0.08262)
500	—	—	—	0.139(0.03866)	0.059(0.03919)
1 000	—	—	—	0.283(0.01688)	0.123(0.01701)
2 000	—	—	—	0.589(0.00818)	0.265(0.00822)
5 000	—	—	—	1.529(0.00352)	0.756(0.00352)
10 000	—	—	—	3.133(0.00181)	1.558(0.00181)
20 000	—	—	—	5.794(0.00064)	3.169(0.00064)
30 000	—	—	—	9.847(0.00054)	5.065(0.00054)
40 000	—	—	—	12.058(0.00042)	6.705(0.00042)
50 000	—	—	—	15.265(0.00034)	8.603(0.00034)

exact algorithms:

MTB = Martello and Toth (1977d);

IK = Ingargiola and Korsh (1977);

MTB2 = Transformation through procedure TB01 (Section 3.2) and solution through algorithm MT2 (Section 2.9.3);

approximate algorithms:

MTB2 approximate = MTB2 with heuristic version of MT2 (Section 2.10.2);

GREEDYB = greedy algorithm (Section 3.3.2).

All runs were executed on an HP 9000/840 (with option "-o" for the Fortran compiler), with values of n ranging from 25 to 50 000 (for $n > 50\,000$, the size of the transformed instances could exceed the memory limit). The tables give average times and percentage errors computed over sets of 20 instances each. The errors are computed as $100(z - z^a)/z$, where z^a is the approximate solution value, and z either the optimal solution value (when available) or upper bound U_2 introduced in Section 3.3.1. The execution of each algorithm was halted as soon as the average time exceeded 100 seconds.

MTB2 is clearly the most efficient exact algorithm for uncorrelated and weakly correlated problems. Optimal solution of strongly correlated problems appears to be practically impossible. As for the heuristic algorithms, GREEDYB dominates the approximate version of MTB2 for uncorrelated and weakly correlated problems, but produces higher errors for strongly correlated problems with $n \leq 2\,000$. The anomalous entry in Table 3.2 (MTB2 exact, $n = 40\,000$) was produced by an instance requiring more than 34 hours!

3.6 A SPECIAL CASE: THE UNBOUNDED KNAPSACK PROBLEM

In this section we consider the problem arising from BKP when an unlimited number of items of each type is available, i.e. the *Unbounded Knapsack Problem* (UKP)

$$\text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{3.14}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c, \tag{3.15}$$

$$x_j \geq 0 \text{ and integer}, \quad j \in N = \{1, \dots, n\}. \tag{3.16}$$

The problem remains NP-hard, as proved in Lueker (1975) by transformation from subset-sum. However, it can be solved in polynomial time in the $n = 2$ case (Hirschberg and Wong (1976), Kannan (1980)). Notice that the result is not trivial, since a naive algorithm, testing $x_1 = i$, $x_2 = \lfloor (c - iw_1)/w_2 \rfloor$ for i taking on integer values from 0 to $\lfloor c/w_1 \rfloor$, would require a time $O(c)$, exponential in the input length.

UKP can clearly be formulated (and solved) by defining an equivalent BKP with $b_j = \lfloor c/w_j \rfloor$ for $j = 1, \dots, n$, but algorithms for BKP generally perform rather poorly in instances of this kind. Also transformation into an equivalent 0-1 knapsack problem is possible (through a straightforward adaptation of the method of Section 3.2), but usually impractical since the number of resulting binary variables $(\sum_{j=1}^{n} \lceil \log_2(\lfloor c/w_j \rfloor + 1) \rceil)$ is generally too elevated for practical solution of the problem.

We maintain assumptions (3.4) and (3.7), while (3.6) transforms into

$$w_j \le c \qquad \text{for } j \in N \tag{3.17}$$

and (3.5) is satisfied by any instance of UKP.

3.6.1 Upper bounds and approximate algorithms

The optimal solution of the continuous relaxation of UKP, defined by (3.14), (3.15) and

$$x_j \ge 0, \qquad j \in N,$$

is $\overline{x}_1 = c/w_1$, $\overline{x}_j = 0$ for $j = 2, \dots, n$, and provides the trivial upper bound

$$U_0 = \left\lfloor c\frac{p_1}{w_1} \right\rfloor.$$

By also imposing $\overline{x}_1 \le \lfloor c/w_1 \rfloor$, which must hold in any integer solution, the continuous solution is

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

$$\overline{x}_j = 0 \qquad \text{for } j = 3, \dots, n,$$

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

where

$$\overline{c} = c (\text{mod } w_1). \tag{3.18}$$

This provides the counterpart of upper bound U_1 of Section 3.3.1, i.e.

$$U_1 = \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + \left\lfloor \bar{c}\frac{p_2}{w_2} \right\rfloor. \tag{3.19}$$

(Note that the critical item type is always $s = 2$.)

The counterpart of the improved upper bound U_2 is

$$U_2 = \max(U^0, U^1). \tag{3.20}$$

where

$$z' = \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + \left\lfloor \frac{\bar{c}}{w_2} \right\rfloor p_2, \tag{3.21}$$

$$c' = \bar{c} (\text{mod } w_2), \tag{3.22}$$

$$U^0 = z' + \left\lfloor c'\frac{p_3}{w_3} \right\rfloor, \tag{3.23}$$

$$U^1 = z' + \left\lfloor p_2 - (w_2 - c')\frac{p_1}{w_1} \right\rfloor. \tag{3.24}$$

In this case, however, we can exploit the fact that $s = 2$ to obtain a better bound. Remember (see Section 3.3.1) that U^1 is an upper bound on the solution value we can obtain if at least $\lfloor \bar{c}/w_2 \rfloor + 1$ items of type 2 are selected. Notice now that this can be done only if at least $\lceil (w_2 - c')/w_1 \rceil$ items of type 1 are removed from the solution corresponding to z', and that $c' + \lceil (w_2 - c')/w_1 \rceil w_1$ units of capacity are then available for the items of type 2. Hence, a valid upper bound can be obtained by replacing U^1 with

$$\overline{U}^1 = z' + \left\lfloor \left(c' + \left\lceil \frac{w_2 - c'}{w_1} \right\rceil w_1 \right) \frac{p_2}{w_2} - \left\lceil \frac{w_2 - c'}{w_1} \right\rceil p_1 \right\rfloor. \tag{3.25}$$

Furthermore, $\overline{U}^1 \leq U^1$ since, with $c' + \lceil (w_2 - c')/w_1 \rceil w_1 \geq w_2$, $\overline{U}^1$ is obtained by "moving" a greater number of capacity units from items of type 1 to (worse) items of type 2. We have thus proved the following

Theorem 3.1 (Martello and Toth, 1990a)

$$U_3 = \max(U^0, \overline{U}^1). \tag{3.26}$$

where U^0 and $\overline{U}^1$ are defined by (3.18), (3.21)–(3.23) *and* (3.25), *is an upper bound for UKP and, for any instance, $U_3 \leq U_2$.*

The time complexity for the computation of U_0, U_1, U_2 and U_3 is $O(n)$, since only the three largest ratios p_j/w_j are needed.

Example 3.2

Consider the instance of UKP defined by

$$\begin{aligned} n &= 3\,; \\ (p_j) &= (20,\ 5,\ 1); \\ (w_j) &= (10,\ 5,\ 3); \\ c &= 39. \end{aligned}$$

The upper bounds are

$$U_0 = 78.$$

$$U_1 = 60 + \left\lfloor 9\frac{5}{5} \right\rfloor = 69.$$

$$U^0 = 65 + \left\lfloor 4\frac{1}{3} \right\rfloor = 66;$$

$$U^1 = 65 + \left\lfloor 5 - 1\frac{20}{10} \right\rfloor = 68;$$

$$U_2 = 68.$$

$$\overline{U}^1 = 65 + \left\lfloor \left(4 + \left\lceil \frac{1}{10} \right\rceil 10\right) \frac{5}{5} - \left\lceil \frac{1}{10} \right\rceil 20 \right\rfloor = 59;$$

$$U_3 = 66. \ \square$$

Since $U_3 \le U_2 \le U_1 \le U_0 \le z' + p_1 \le 2z$, the worst-case performance ratio of all bounds is at most 2. To see that $\rho(U_0) = \rho(U_1) = \rho(U_2) = \rho(U_3) = 2$, consider the series of problems with $n = 3$, $p_j = w_j = k$ for all j, and $c = 2k - 1$: we have $U_0 = U_1 = U_2 = U_3 = 2k - 1$ and $z = k$, so the ratio (upper bound)/z can be arbitrarily close to 2 for k sufficiently large.

The heuristic solution value defined by (3.21) has an interesting property. Remember that the analogous values z' defined for BKP (Section 3.3.2) and for the 0-1 knapsack problem (Section 2.4) can provide an arbitrarily bad approximation of the optimal value z. For any instance of UKP, instead, we have it that $z'/z \ge \frac{1}{2}$. The proof is immediate by observing that $z - z' \le p_1$ and, from (3.17), $z' \ge p_1$. The series of problems with $n = 2$, $p_1 = w_1 = k + 1$, $p_2 = w_2 = k$ and $c = 2k$ shows that $\frac{1}{2}$ is tight, since $z'/z = (k + 1)/(2k)$ can be arbitrarily close to $\frac{1}{2}$ for k sufficiently large. Also notice that the same property holds for the simpler heuristic value $z'' = \lfloor c/w_1 \rfloor p_1$.

The greedy algorithm of Section 3.3.2 can now be simplified as follows. (We assume that the item types are sorted according to (3.7).)

```
procedure GREEDYU:
input: $n, c, (p_j), (w_j)$;
output: $z^g, (x_j)$;
begin
    $\bar{c} := c$;
    $z^g := 0$;
    for $j := 1$ to $n$ do
        begin
            $x_j := \lfloor \bar{c}/w_j \rfloor$;
            $\bar{c} := \bar{c} - w_j x_j$;
            $z^g := z^g + p_j x_j$
        end
end.
```

The time complexity of GREEDYU is $O(n)$, plus $O(n\log n)$ for the preliminary sorting.

Magazine, Nemhauser and Trotter (1975) studied theoretical properties of the greedy algorithm when applied to the minimization version of UKP. In particular, they determined necessary and sufficient conditions for the optimality of the greedy solution (see also Hu and Lenard (1976) for a simplified proof), and analysed the worst-case absolute error produced by the algorithm. Ibarra and Kim (1975) adapted their fully polynomial-time approximation scheme for the 0-1 knapsack problem (Section 2.8.2) to UKP. The resulting scheme produces, for any fixed $\varepsilon > 0$, a solution having worst-case relative error not greater than ε in time $O(n + (1/\varepsilon^4)\log(1/\varepsilon))$ and space $O(n + (1/\varepsilon^3))$. Also Lawler (1979) derived from his algorithm for the 0-1 knapsack problem (Section 2.8.2) a fully polynomial-time approximation scheme for UKP, obtaining time and space complexity $O(n+(1/\varepsilon^3))$.

3.6.2 Exact algorithms

An immediate recursion for computing the dynamic programming function $f_m(\hat{c})$ (see Section 3.4.1), is

$$f_1(\hat{c}) = \left\lfloor \frac{\hat{c}}{w_1} \right\rfloor p_1 \qquad \text{for } \hat{c} = 0, \ldots, c;$$

$$f_m(\hat{c}) = \max\left\{ f_{m-1}(\hat{c} - lw_m) + lp_m : l \text{ integer}, \ 0 \le l \le \left\lfloor \frac{\hat{c}}{w_m} \right\rfloor \right\}$$
$$\text{for } m = 2, \ldots, n \text{ and } \hat{c} = 0, c.$$

The time complexity for determining $z = f_n(c)$ is $O(nc^2)$.

Gilmore and Gomory (1965) have observed that a better recursion for computing $f_m(\hat{c})$, for $m = 2, \ldots, n$, is

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

which reduces the overall time complexity to $O(nc)$.

Specialized dynamic programming algorithms for UKP have been given by Gilmore and Gomory (1966), Hu (1969), Garfinkel and Nemhauser (1972), Greenberg and Feldman (1980), Greenberg (1985, 1986). Dynamic programming, however, is usually capable of solving only instances of limited size.

More effective algorithms, based on branch-and-bound, have been proposed by Gilmore and Gomory (1963), Cabot (1970) and Martello and Toth (1977d). The last one has proved to be experimentally the most effective (Martello and Toth, 1977d), and derives from algorithm MT1 for the 0-1 knapsack problem, described in Section 2.5.2. Considerations (i) to (iii) of that section easily extend to this algorithm, while parametric computation of upper bounds (consideration (iv)) is no longer needed, since the current critical item type is always the next item type to be considered. The general structure of the algorithm and the variable names used in the following detailed description are close to those in MT1. It is assumed that the item types are sorted according to (3.7).

```
procedure MTU1:
input: $n, c, (p_j), (w_j)$;
output: $z, (x_j)$;
begin
1. [initialize]
    $z := 0$;
    $\hat{z} := 0$;
    $\hat{c} := c$;
    $p_{n+1} := 0$;
    $w_{n+1} := +\infty$;
    for $k := 1$ to $n$ do $\hat{x}_k := 0$;
    compute the upper bound $U = U_3$ on the optimal solution value;
    for $k := n$ to $1$ step $-1$ do compute $m_k = \min\{w_i : i > k\}$;
    $j := 1$;
2. [build a new current solution]
    while $w_j > \hat{c}$ do
        if $z \geq \hat{z} + \lfloor \hat{c} p_{j+1} / w_{j+1} \rfloor$ then go to 5 else $j := j + 1$;
    $y := \lfloor \hat{c} / w_j \rfloor$;
    $u := \lfloor (\hat{c} - y w_j) p_{j+1} / w_{j+1} \rfloor$;
    if $z \geq \hat{z} + y p_j + u$ then go to 5;
    if $u = 0$ then go to 4;
3. [save the current solution]
    $\hat{c} := \hat{c} - y w_j$;
    $\hat{z} := \hat{z} + y p_j$;
    $\hat{x}_j := y$;
    $j := j + 1$;
    if $\hat{c} \geq m_{j-1}$ then go to 2;
    if $z \geq \hat{z}$ then go to 5;
    $y := 0$;
4. [update the best solution so far]
    $z := \hat{z} + yp_j$;
    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 $z = U$ then return ;
5. [backtrack]
    find $i = \max\{k < j : \hat{x}_k > 0\}$;
    if no such $i$ then return ;
    $\hat{c} := \hat{c} + w_i$;
    $\hat{z} := \hat{z} - p_i$;
    $\hat{x}_i := \hat{x}_i - 1$;
    if $z \geq \hat{z} + \lfloor \hat{c} p_{i+1} / w_{i+1} \rfloor$ then
        begin
            comment: remove all items of type $i$;
            $\hat{c} := \hat{c} + w_i \hat{x}_i$;
            $\hat{z} := \hat{z} - p_i \hat{x}_i$;
            $\hat{x}_i := 0$;
            $j := i$;
            go to 5
        end;
    $j := i + 1$;
    if $\hat{c} - w_i \geq m_i$ then go to 2;
    $h := i$;
6. [try to replace one item of type $i$ with items of type $h$]
    $h := h + 1$;
    if $z \geq \hat{z} + \lfloor \hat{c} p_h / w_h \rfloor$ then go to 5;
    if $w_h = w_i$ then go to 6;
    if $w_h > w_i$ then
        begin
            if $w_h > \hat{c}$ or $z \geq \hat{z} + p_h$ then go to 6;
            $z := \hat{z} + p_h$;
            for $k := 1$ to $n$ do $x_k := \hat{x}_k$;
            $x_h := 1$;
            if $z = U$ then return;
            $i := h$;
            go to 6
        end
    else
        begin
            if $\hat{c} - w_h < m_{h-1}$ then go to 6;
            $j := h$;
            go to 2
        end
end.
```

Example 3.3

Consider the instance of UKP defined by

$$\begin{aligned} n &= 7\,; \\ (p_j) &= (20, 39, 52, 58, 31, 4, 5); \\ (w_j) &= (15, 30, 41, 46, 25, 4, 5); \\ c &= 101. \end{aligned}$$

Figure 3.1 gives the decision-tree produced by algorithm MTU1. □

The Fortran implementation of procedure MTU1 is included in that of procedure MTU2, which is described in the next section.

3.6.3 An exact algorithm for large-size problems

Experimental results with algorithm MTU1, reported in Martello and Toth (1977b), show a behaviour close to that of analogous algorithms for the 0-1 knapsack problem, i.e.: (i) in spite of its worst-case complexity, many instances of UKP can be exactly solved within reasonable computing times, even for very large values of n; (ii) when this is possible, the sorting time is usually a very large fraction of the total time; however, (iii) only the item types with the highest values of the ratio p_j/w_j are selected for the solution, i.e. $\max\{j : x_j > 0\} \ll n$.

The concept of core problem (Section 2.9) can be extended to UKP by recalling that, in this case, the critical item type is always the second one. Hence, given a UKP and supposing, without loss of generality, that $p_j/w_j > p_{j+1}/w_{j+1}$ for $j = 1, \dots, n-1$, we define the *core* as

$$C = \{1, 2, \dots, \overline{n} \equiv \max\{j : x_j > 0\}\}.$$

and the *core problem* as

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

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

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

If we knew "a priori" the value of $\overline{n}$, we could solve UKP by setting $x_j = 0$ for all j such that $p_j/w_j < p_{\overline{n}}/w_{\overline{n}}$, determining C as $\{j : p_j/w_j \ge p_{\overline{n}}/w_{\overline{n}}\}$ and solving the resulting core problem by sorting only the items in C. $\overline{n}$ cannot, of course, be "a priori" identified, but we can determine an approximate core without sorting as follows.

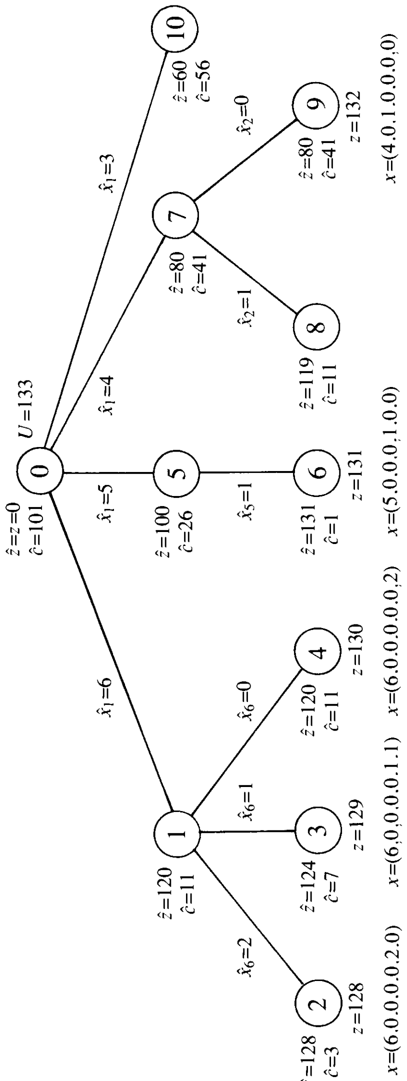

Figure 3.1 Decision-tree of procedure MTU1 for Example 3.3

Assuming no condition on the ratios p_j/w_j, we select a tentative value for $p_{\overline{n}}/w_{\overline{n}}$ and solve the corresponding core problem: if the solution value equals that of an upper bound, then we have the optimum; otherwise, we reduce the variables not in the core and, if any variables are left, we try again with a decreased tentative value. Reduction is based on the following criterion. Let $U_q(j)$ denote upper bound U_q ($q = 1, 2$ or 3) of Section 3.6.1 for UKP, with the additional constraint $x_j = 1$, i.e. an upper bound on the solution value that UKP can have if item type j is used for the solution. If, for j not in the approximate core, we have $U_q(j) \leq z$ (where z denotes the solution value of the approximate core problem), then we know that x_j must take the value 0 in any solution better than the current one. Given a tentative value ϑ for the initial core problem size, the resulting algorithm is thus (Martello and Toth, 1990a) the following.

```
procedure MTU2:
input: $n, c, (p_j), (w_j), \vartheta$;
output: $z, (x_j)$;
begin
    $k := 0$;
    $\overline{N} := \{1, 2, \dots, n\}$;
    repeat
        $k := \min(k + \vartheta, |\overline{N}|)$;
        find  the $k$th largest value $r$ in $\{p_j/w_j : j \in \overline{N}\}$;
        $G := \{j \in \overline{N} : p_j/w_j > r\}$;
        $E := \{j \in \overline{N} : p_j/w_j = r\}$;
        $\overline{E}$ := any subset of $E$ such that $|\overline{E}| = k - |G|$;
        $C := G \cup \overline{E}$;
        sort the item types in $C$ according to decreasing $p_j/w_j$ ratios;
        exactly solve the core problem, using MTU1, and let $z$ and $(x_j)$ define
          the solution;
        if $k = \vartheta$ (comment: first iteration) then
            compute upper bound $U_3$ of Section 3.6.1;
        if $z < U_3$ then (comment: reduction)
            for each $j \in \overline{N} \backslash C$ do
                begin
                    $u := U_1(j)$;
                    if $u > z$ then $u := U_3(j)$;
                    if $u \leq z$ then $\overline{N} := \overline{N} \backslash \{j\}$
                end
    until $z = U_3$ or $\overline{N} = C$;
    for each $j \in \{1, \dots, n\} \backslash C$ do $x_j := 0$
end.
```

At each iteration, the exact solution of the core problem is obtained by first identifying *dominated* item types in C, then applying algorithm MTU1 to the undominated item types. Dominances are identified as follows.

Definition 3.1 *Given an instance of UKP, relative to item types set N, item type*

$k \in N$ is dominated if the optimal solution value does not change when k is removed from N.

Theorem 3.2 (Martello and Toth, 1990a) *Given any instance of UKP and an item type k, if there exists an item type j such that*

$$\left\lfloor \frac{w_k}{w_j} \right\rfloor p_j \geq p_k \tag{3.27}$$

then k is dominated.

Proof. Given a feasible solution in which $x_k = \alpha > 0$ and $x_j = \beta$, a better solution can be obtained by setting $x_k = 0$ and $x_j = \beta + \lfloor w_k/w_j \rfloor \alpha$. In fact: (i) the new solution is feasible, since $\lfloor w_k/w_j \rfloor \alpha w_j \leq \alpha w_k$; (ii) the profit produced by item type j in the new solution is no less than that produced by item types j and k in the given solution, since, from (3.27), $\lfloor w_k/w_j \rfloor \alpha p_j \geq \alpha p_k$. □

Corollary 3.1 *All dominated item types can be efficiently eliminated from the core as follows:*

```
1. sort the item types according to (3.7), breaking ties so that $w_j \leq w_{j+1}$;
2. for $j := 1$ to $|C| - 1$ do
       for $k := j + 1$ to $|C|$ do if (3.27) holds then $C := C \setminus \{k\}$.
```

Proof. Condition (3.27) never holds if either $p_j/w_j < p_k/w_k$ or $w_k < w_j$. □

Hence the time complexity to eliminate the dominated item types is $O(|C|^2)$ (or $O(n^2)$, if the original UKP is considered).

Example 3.3 (continued)

Taking $\vartheta = 4$, the core problem is defined by:

$(p_j) = (20,\ 39,\ 52,\ 58)$;

$(w_j) = (15,\ 30,\ 41,\ 46)$.

Applying Corollary 3.1, we find that item type 1 dominates item types 2 and 4. Applying MTU1 to the resulting problem, defined by

$(p_j) = (20,\ 52)$;

$(w_j) = (15,\ 41)$.

we obtain the branch-decision tree of Figure 3.2.

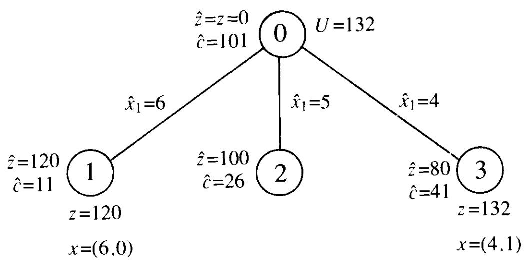

Figure 3.2 Decision-tree of procedure MTU2 for Example 3.3

The core problem solution value ($z = 132$) is not equal to upper bound U_3 relative to the original instance without the dominated item types ($U_3 = \max(120 + \lfloor 11\ \frac{31}{25} \rfloor .\ 120 + \lfloor (11 + \lceil \frac{30}{15} \rceil 15)\frac{52}{41} - \lceil \frac{30}{15} \rceil 20 \rfloor) = 133$). Hence we apply the reduction phase:

$$j = 5 : U_1(5) = 31 + \left(100 + \left\lfloor 1\frac{52}{41} \right\rfloor\right) = 132 \leq z;$$

$$j = 6 : U_1(6) = 4 + \left(120 + \left\lfloor 7\frac{52}{41} \right\rfloor\right) = 132 \leq z;$$

$$j = 7 : U_1(7) = 5 + \left(120 + \left\lfloor 6\frac{52}{41} \right\rfloor\right) = 132 \leq z.$$

Since all the item types not in core are reduced, we conclude that the core problem has produced the optimal solution $z = 132$. $(x_j) = (4, 0, 1, 0, 0, 0, 0)$. □

The initial tentative value ϑ was experimentally determined as

$$\vartheta = \max\left(100.\ \left\lfloor \frac{n}{100} \right\rfloor\right).$$

The Fortran implementation of algorithm MTU2 is included in the present volume.

3.6.4 Computational experiments

Table 3.4 compares the algorithms for UKP on the same data sets of Section 3.5, but with w_j uniformly randomly generated in the range [10,1000], so as to avoid the occurrence of trivial instances in which the item type with largest p_j/w_j ratio has $w_j = 1$ (so $x_1 = c$ is the optimal solution).

For all problems, c was set to $0.5\sum_{j=1}^{n} w_j$ for $n \leq 100\,000$, to $0.1\sum_{j=1}^{n} w_j$ (in order to avoid integer overflows) for $n > 100\,000$.

We compare the Fortran IV implementations of algorithms MTU1 and MTU2. The kth largest ratio p_j/w_j was determined through the algorithm given in Fischetti and Martello (1988) (including Fortran implementation). All runs have been

Table 3.4 w_j uniformly random in [10,1000]; $c = 0.5\sum_{j=1}^{n} w_j$ for $n \leq 100\,000$, $c = 0.1\sum_{j=1}^{n} w_j$ for $n > 100\,000$. HP 9000/840 in seconds. Average times over 20 problems

		Uncorrelated: p_j unif. random in [1,1000]		Weakly correlated: p_j unif. random in $[w_j - 100, w_j + 100]$		Strongly correlated: $p_j = w_j + 100$	
n	Sorting	MTU1	MTU2	MTU1	MTU2	MTU1	MTU2
50	0.01	0.01	0.01	0.01	0.01	0.01	0.01
100	0.01	0.01	0.01	0.01	0.01	0.01	0.01
200	0.02	0.02	0.02	0.02	0.03	0.06	0.02
500	0.05	0.05	0.04	0.05	0.05	131.70	0.04
1 000	0.11	0.11	0.07	0.11	0.08	—	0.08
2 000	0.24	0.24	0.13	0.24	0.14	—	0.14
5 000	0.60	0.68	0.32	0.62	0.29	—	0.35
10 000	1.35	1.44	0.60	1.37	0.66	—	0.62
20 000	3.21	3.31	1.23	494.93	1.18	—	1.39
30 000	4.71	5.38	1.82	—	1.94	—	1.91
40 000	6.12	9.67	2.73	—	2.48	—	2.66
50 000	8.04	21.91	3.25	—	3.30	—	3.34
60 000	10.53	41.11	3.90	—	3.71	—	4.10
70 000	12.50	17.63	4.89	—	4.50	—	4.81
80 000	13.86	172.00	5.28	—	5.00	—	5.12
90 000	15.56	—	5.88	—	5.41	—	5.68
100 000	17.96	—	5.83	—	6.22	—	5.81
150 000	27.41	—	10.05	—	10.14	—	9.95
200 000	37.56	—	13.08	—	11.98	—	13.26
250 000	48.55	—	17.35	—	17.52	—	17.94

executed on an HP 9000/840 with option "-o" for the Fortran compiler. For each data set and value of n, Table 3.4 gives the average running times (including sorting), expressed in seconds, computed over 20 problem instances. Sorting times are also separately shown. Execution of an algorithm was halted as soon as the average running time exceeded 100 seconds.

The table shows that MTU2 always dominates MTU1, and can solve very large problems with reasonable computing time also in the case of strongly correlated data sets. The initial value of ϑ always produced the optimal solution. With the exception of strongly correlated data sets, MTU1 requires negligible extra computational effort after sorting, when $n \leq 10\,000$. For larger values of n, the branch-and-bound phase can become impractical. This shows that the superiority of MTU2 (particularly evident for very large instances and for strongly correlated problems) derives not only from the avoided sorting phase but also from application of the dominance criterion. In fact, the number of undominated item types was always very small and almost independent of n.