MANAGEMENT SCIENCE
Vol. 34, No. 5, May 1988
Printed in U.S.A.

A NEW ALGORITHM FOR THE 0-1 KNAPSACK PROBLEM*

SILVANO MARTELLO AND PAOLO TOTH
DEIS, University of Bologna, Italy

We present a new algorithm for the optimal solution of the 0-1 Knapsack problem, which is particularly effective for large-size problems. The algorithm is based on determination of an appropriate small subset of items and the solution of the corresponding "core problem": from this we derive a heuristic solution for the original problem which, with high probability, can be proved to be optimal. The algorithm incorporates a new method of computation of upper bounds and efficient implementations of reduction procedures. The corresponding Fortran code is available. We report computational experiments on small-size and large-size random problems, comparing the proposed code with all those available in the literature.
(KNAPSACK PROBLEM; BRANCH-AND-BOUND)

1. Introduction

Given n items, each having a *profit* p_j and a *weight* w_j ($j \in N = \{1, \ldots, n\}$), and a container of *capacity* c, the well-known 0-1 *single knapsack problem* (KP) is

$$\text{maximize} \quad z = \sum_{j \in N} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j \in N} w_j x_j \leqslant c, \tag{2}$$

$$x_j \in \{0, 1\} \qquad (j \in N). \tag{3}$$

We will suppose, without loss of generality,
(i) p_j, w_j and c positive integers;
(ii) $\max_j \{w_j\} \leqslant c$;
(iii) $\sum_{j \in N} w_j > c$.

KP is NP-hard (see Garey and Johnson 1979), although it can be solved in pseudo-polynomial time by dynamic programming. The problem has been intensively studied in the last decade both because of its theoretical interest and its wide applicability. As a matter of fact, many instances of the problem can be solved, within acceptable running times, even for very large values of n.

The best-known algorithms for KP are those of Horowitz and Sahni (1974), Nauss (1976), Martello and Toth (1977, 1978), all based on branch-and-bound techniques and requiring preliminary sorting of the items according to decreasing values of the profit per unit weight. For large values of n ($n \geqslant 2000$) it turns out that if an instance can be efficiently solved then most of the computing time (about 90%) is spent for the above sorting. To overcome this inconvenience, Balas and Zemel (1980) have proposed a particular approach which in many cases allows one to solve the problem by sorting only a small subset of the items. Fayard and Plateau (1982) have presented an algorithm based on the Balas-Zemel idea. Efficient Fortran implementations of algorithms for KP are included in the papers by Martello and Toth (1978) and Fayard and Plateau (1982), or can be obtained from Nauss (1976). For further details we refer the reader to a recent survey by Martello and Toth (1987).

* Accepted by Alexander H. G. Rinnooy Kan; received August 1986. This paper has been with the authors 5 months for 1 revision.

0025-1909/88/3403/633$01.25

The basic structure of the algorithm we propose arises from the following experimental result. When a large-size instance can be solved within acceptable computing time it turns out that, in general, there exist two values, α and β ($\alpha > \beta$), such that

(i) $x_j = 1$ for all j for which $p_j/w_j > \alpha$;

(ii) $x_j = 0$ for all j for which $p_j/w_j < \beta$.

Hence, knowing α and β in advance would allow one to solve KP by considering only the items in $C = \{j: \alpha \geqslant p_j/w_j \geqslant \beta\}$: solving (for example through branch-and-bound) the *core problem* defined by the items in C with reduced capacity $c - \sum_{j \in \{k: p_k/w_k > \alpha\}} w_j$ would require sorting of only $|C|$ items, with considerable computational advantage, since $|C|$ is usually a very small fraction of n. Balas and Zemel (1980) have proposed a partitioning technique from which it is possible to derive an algorithm for determining an approximate core problem in $O(n)$ time.

The algorithm we propose can be sketched as follows:

Step 1. Determine an approximate core $C' = \{j: \alpha' \geqslant w_j \geqslant \beta'\}$ through a modification of the Balas-Zemel method. Sort the items in C' and exactly solve the core problem. Derive an approximate solution for KP by setting $x_j = 1$ if $p_j/w_j > \alpha'$, $x_j = 0$ if $p_j w_j < \beta'$.

Step 2. Compute an upper bound for KP. If its value equals that of the approximate solution then this is clearly optimal: stop. Otherwise

Step 3. Reduce KP by determining, for some variables x_j, the value they must take in an optimal solution.

Step 4. If all variables x_j such that $p_j/w_j > \alpha'$ or $p_j/w_j < \beta'$ have been fixed, respectively to 1 and to 0, by reduction, then we have it that $C \subseteq C'$, so the approximate solution of *Step* 1 is optimal: stop. Otherwise

Step 5. Sort the items corresponding to variables not fixed by reduction and exactly solve the corresponding problem.

The algorithm improves upon previous works from the literature in three main respects:

(a) the approximate solution determined at Step 1 is more precise (often optimal); this is obtained through more careful definition of the approximate core and through exact (instead of heuristic) solution of the corresponding problem;

(b) there is a higher probability that such an approximate solution can be proved to be optimal either at Step 2 (because of a tighter upper bound computation) or at Step 4 (missing in previous works);

(c) the procedures for determining the approximate core (Step 1), reducing KP (Step 3) and exactly solving subproblems (Steps 1 and 5) have been implemented more efficiently.

In §2 we give a new method for the computation of upper bounds and in §3 reduction procedures. The algorithm is presented in §4 and experimentally compared with the other codes from the literature in §5. The Fortran code of the proposed algorithm is available on request from the authors.

2. Upper Bounds

The first upper bound for KP was derived by Dantzig (1957) from the continuous relaxation of the problem (i.e. (1), (2) and $0 \leqslant x_j \leqslant 1$ ($j \in N$)). Assume that the items are sorted so that $p_j/w_j \geqslant p_{j+1}/w_{j+1}$ for all j and define the *critical item* s as $s = \min\{j: \sum_{i=1}^{j} w_i > c\}$; the optimal solution of the relaxed problem is: $x_j = 1$ for $j = 1, \ldots, s-1$; $x_j = 0$ for $j = s+1, \ldots, n$; $x_s = (c - \sum_{j=1}^{s-1} w_j)/w_s$. Hence the Dantzig bound:

$$u = \sum_{j=1}^{s-1} p_j + \lfloor (c - \sum_{j=1}^{s-1} w_j) p_s/w_s \rfloor.$$

Many improved upper bounds have been proposed in the literature: we mention in particular those of Martello and Toth (1977), Müller-Merbach (1978), Fayard and

Plateau (1982), Dudzinski and Walukiewicz (1984). All these bounds can be computed in $O(n)$ time, once the critical item s in known. s can be determined either in $O(n \log n)$ time by sorting the p_j/w_j ratios, or in $O(n)$ time by modifying a partitioning technique proposed by Balas and Zemel (1980).

The upper bound presented in Martello and Toth (1977), which will be used in the following sections, is

$$u' = \sum_{j=1}^{s-1} p_j + \max \{[(c - \sum_{j=1}^{s-1} w_j)p_{s+1}/w_{s+1}], \quad [p_s - (w_s - (c - \sum_{j=1}^{s-1} w_j))p_{s-1}/w_{s-1}]\}.$$

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

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

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

For each $l \in L_1 \cup L_2$, let u_l be any upper bound on the problem defined by (1), (2) and

$$\begin{cases} x_j = x_j^l & \text{if} \quad j \in \{r, \ldots, f(l)\}, \\ x_j \in \{0, 1\} & \text{if} \quad j \in N \setminus \{r, \ldots, f(l)\}. \end{cases} \tag{4}$$

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

$$\bar{u} = \max \{u_l: l \in L_1 \cup L_2\}. \tag{5}$$

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

$$u_l = \begin{cases} [p^l - d^l p_{r-1}/w_{r-1}] & \text{if} \quad l \in L_1; \\ [p^l + d^l p_{t+1}/w_{t+1}] & \text{if} \quad l \in L_2, \end{cases}$$

which is clearly an upper bound on the continuous solution value for problem (1), (2), (4).

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

We illustrate the above concepts with an example. Let $c = 100$, $n = 7$ and

$$(p_j) = (100, 90, 60, 40, 15, 10, 10); \qquad (w_j) = (20, 20, 30, 40, 30, 60, 70).$$

We have $s = 4$, so the Dantzig bound is $u = 280$. Assume $r = 3$, $t = 5$: the reduced capacity is $c(r) = 60$, the binary tree is given in Figure 1. The leaf sets are $L_1 = \{3, 9\}$, $L_2 = \{5, 6, 10, 12, 13\}$ and the corresponding bounds are $u_3 = 245$, $u_9 = 200$; $u_5 = 265$, u_6

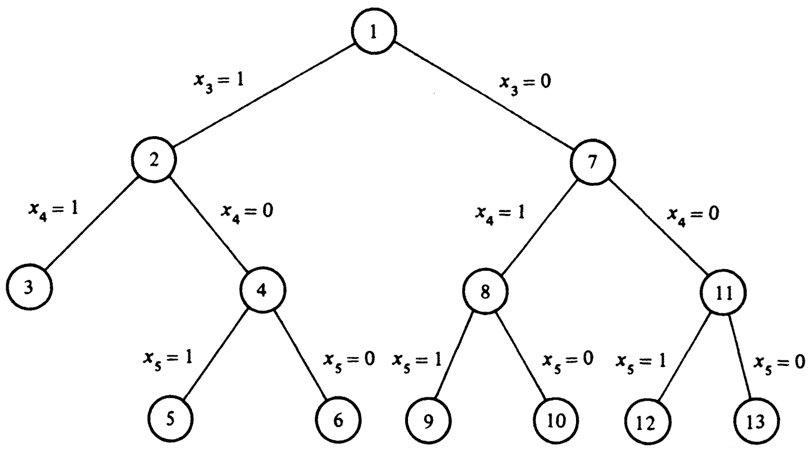

FIGURE 1

$= 255$, $u_{10} = 233$, $u_{12} = 210$, $u_{13} = 200$. It follows that $\bar{u} = 265$, which is the optimal solution.

A more accurate way to compute $\bar{u}$ could consist of evaluating the upper bounds at the leaves through any method from the literature (Dantzig 1957, Martello and Toth, 1977, Müller-Merback 1978, Fayard and Plateau 1982, Dudzinski and Walukiewicz 1984). By means of computational experiments, however, we have found that the increased computational effort is not rewarded by a significant decrease in the value of $\bar{u}$ (for the above example, use of the Dantzig bound gives the same values of u_l for all $l \in L_1 \cup L_2$).

A sharper bound can be obtained—with very small extra computational effort—by evaluating $w_m = \min\{w_j: j > t\}$: it is not difficult to see that, when $l \in L_2$ and $d^l < w_m$, u_l can be computed as $\max\{p^l, [p^l + w_m p_{t+1}/w_{t+1} - (w_m - d^l)p_{r-1}/w_{r-1}]\}$. For the previous example, this would give $w_m = 60$, $u_5 = 265$, $u_6 = 250$, $u_{10} = 230$, $u_{12} = 205$.

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

$$\tilde{u}_k = \sum_{j=r}^{f(k)} p_j x_j^k + \sum_{j=f(k)+1}^{s(k)-1} p_j + \Big[\Big(c(r) - \Big(\sum_{j=r}^{f(k)} w_j x_j^k + \sum_{j=f(k)+1}^{s(k)-1} w_j\Big)\Big)p_{s(k)}/w_{s(k)}\Big],$$

where $s(k) = \min\{t + 1, \min\{i: \sum_{j=r}^{f(k)} w_j x_j^k + \sum_{j=f(k)+1}^{i} w_j > c(r)\}\}$. If we have $\tilde{u}_k \leqslant \bar{z}(r, t)$, the nodes descending from k need not be generated. In fact, for any leaf l descending from k, it would result that $u_l \leqslant \sum_{j=1}^{r-1} p_j + \tilde{u}_k \leqslant \sum_{j=1}^{r-1} p_j + z(r, t) \leqslant \bar{u}$. In the example of Figure 1, after node 5, the value of the best current solution is $\bar{z}(r, t) = 75$: at node 7 we compute $\tilde{u}_7 = 50$, so nodes 8 to 13 need not be generated.

3. Reduction Procedures

The size of an instance of KP can be reduced through procedures (see Ingargiola and Korsh 1973, Toth 1976, Dembo and Hammer 1980, Fayard and Plateau 1982) which partition set N into (N_0, N_1, F) such that KP can have an optimal solution (x_j^*) better than a given feasible solution only if

$$x_j^* = 0 \quad \text{for all} \quad j \in N_0,$$
$$x_j^* = 1 \quad \text{for all} \quad j \in N_1$$

(we will call *free* the items in F). The optimal solution value for KP is then obtained as $z = \sum_{j\in N_1} p_j + z(F)$, where $z(F)$ is the optimal solution value for the reduced problem defined by the free items and by capacity $c(F) = c - \sum_{j\in N_1} w_j$.

Let $\bar{z}$ be the value of any feasible solution to KP and s the critical item. Define $G = \{j \in N: p_j/w_j \geqslant p_s/w_s\}$, $L = \{j \in N: p_j/w_j \leqslant p_s/w_s\}$ and let u_j^0 (resp. u_j^1) be an upper bound on KP with the additional constraint $x_j = 0$ (resp. $x_j = 1$). N_1 (resp. N_0) is then determined as $N_1 = \{j \in G: u_j^0 \leqslant \bar{z}\}$ (resp. $N_0 = \{j \in L: u_j^1 \leqslant \bar{z}\}$). The effectiveness and computational complexity of a reduction procedure clearly depend on the techniques used in computing $\bar{z}$, u_j^0 and u_j^1 (the time complexity of procedures from the literature ranges between $O(n)$ and $O(n^2)$).

The algorithm for KP presented in §4.2 makes use of two reduction procedures. The first, RCS (Reduction with Complete Sorting), determines N_0 and N_1 by assuming that all items are sorted according to decreasing p_j/w_j ratios. Instead of sets G and L above, RCS explores, respectively, sets $\{1, \ldots, s\}$ and $\{s, \ldots, n\}$. u_j^0 and u_j^1 are computed through the Martello-Toth (1977) bound (lines 11 and 16), $\bar{z}$ is initially defined as the *greedy solution* of KP (lines 3–7) and improved at each u_j^0 or u_j^1 computation (lines 12 and 17).

```
Procedure RCS ($N_0$, $N_1$, $\bar{z}$):
comment It is assumed that $p_j/w_j \geqslant p_{j+1}/w_{j+1}$ for all $j \in N$;
 1. for $j$ := 1 to $n$ do compute $\bar{w}_j = \sum_{i=1}^{j} w_i$, $\bar{p}_j = \sum_{i=1}^{j} p_i$;
 2. find, through binary search, $s$ such that $\bar{w}_{s-1} \leqslant c < \bar{w}_s$;
 3. $\bar{z} := \bar{p}_{s-1}$;
 4. $\bar{c} := c - \bar{w}_{s-1}$;
 5. for $j := s + 1$ to $n$ do if $w_j \leqslant \bar{c}$ then
        begin
 6.         $\bar{z} := \bar{z} + p_j$;
 7.         $\bar{c} := \bar{c} - w_j$
        end;
 8. for $j$ := 1 to $s$ do
        begin
 9.         find, through binary search, $\bar{s}$ such that $\bar{w}_{\bar{s}-1} \leqslant c + w_j < \bar{w}_{\bar{s}}$ (current critical
            item);
10.         $\bar{c} := c + w_j - \bar{w}_{\bar{s}-1}$;
11.         $u_j^0 := \bar{p}_{\bar{s}-1} - p_j + \max\{[\bar{c}p_{\bar{s}+1}/w_{\bar{s}+1}], [p_{\bar{s}} - (w_{\bar{s}} - \bar{c})p_{\bar{s}-1}/w_{\bar{s}-1}]\}$;
12.         $\bar{z} := \max\{\bar{z}, \bar{p}_{\bar{s}-1} - p_j\}$
        end;
13. for $j := s$ to $n$ do
        begin
14.         find, through binary search, $\bar{s}$ such that $\bar{w}_{\bar{s}-1} \leqslant c - w_j < \bar{w}_{\bar{s}}$;
15.         $\bar{c} := c - w_j - \bar{w}_{\bar{s}-1}$;
16.         $u_j^1 := \bar{p}_{\bar{s}-1} + p_j + \max\{[\bar{c}p_{\bar{s}+1}/w_{\bar{s}+1}], [p_{\bar{s}} - (w_{\bar{s}} - \bar{c})p_{\bar{s}-1}/w_{\bar{s}-1}]\}$;
17.         $\bar{z} := \max\{\bar{z}, \bar{p}_{\bar{s}-1} + p_j\}$
        end;
18. $N_0 := \{j \geqslant s: u_j^1 \leqslant \bar{z}\}$;
19. $N_1 := \{j \leqslant s: u_j^0 \leqslant \bar{z}\}$.
```

The time complexity of line 1 is $O(n)$, that of line 2 $O(\log n)$. The loop of lines 5–7 has time complexity $O(n)$, that of lines 8–12 $O(n \log n)$, as well as that of lines 13–17. Lines 18–19 require $O(n)$ time. The overall time complexity of RCS is thus $O(n \log n)$ and is not affected by the preliminary sorting needed by the procedure.

The second procedure used in the algorithm of §4.2, RPS (Reduction with Partial Sorting), assumes that only a subset $\tilde{F} \subseteq N$ of items is sorted. RPS receives in input a feasible solution value $\tilde{z}$ and a partition $(\tilde{F}, \tilde{L}, \tilde{G})$ of N such that

(i) $\tilde{L} \subset L$, $\tilde{G} \subset G$;

(ii) $\tilde{c} = c - \sum_{j \in \tilde{G}} w_j > 0$ and $\sum_{j \in \tilde{F}} w_j > \tilde{c}$;

(iii) $\max\{p_k/w_k: k \in \tilde{L}\} \leqslant p_j/w_j \leqslant \min\{p_k/w_k: k \in \tilde{G}\}$ for all $j \in \tilde{F}$.

N_0 and N_1 are determined according to the same rules as in procedure RCS (but weaker bounds u_j^0 and u_j^1 are computed when (lines 13 and 20) the current critical item $\bar{s}$ is not in $\tilde{F}$).

```
Procedure RPS ($\tilde{z}, \tilde{F}, \tilde{L}, \tilde{G}, N_0, N_1$):
comment It is assumed that the items in $\tilde{F}$ are $1, 2, \ldots, f (f = |\tilde{F}|)$, sorted so that $p_j/w_j \geqslant p_{j+1}/w_{j+1}$ for all $j \in \tilde{F}$;
 1. $\tilde{c} := c - \sum_{j \in \tilde{G}} w_j$;
 2. $\tilde{p} := \sum_{j \in \tilde{G}} p_j$;
 3. for $j = 1$ to $f$ do compute $\bar{w}_j = \sum_{i=1}^{j} w_i$, $\bar{p}_j = \sum_{i=1}^{j} p_i$;
 4. find, through binary search, $s \in \tilde{F}$ such that $\bar{w}_{s-1} \leqslant \tilde{c} < \bar{w}_s$;
 5. for each $j \in \tilde{G} \cup \{1, \ldots, s\}$ do
      begin
 6.     if $\tilde{c} + w_j < \bar{w}_f$ then
          begin
 7.         find, through binary search, $\bar{s} \in \tilde{F}$ such that $\bar{w}_{\bar{s}-1} \leqslant \tilde{c} + w_j < \bar{w}_{\bar{s}}$;
 8.         $\bar{c} := \tilde{c} + w_j - \bar{w}_{\bar{s}-1}$;
 9.         $u_j^0 := \tilde{p} - p_j + \bar{p}_{\bar{s}-1} + \max\{\lfloor \bar{c} p_{\bar{s}+1}/w_{\bar{s}+1} \rfloor, \lfloor p_{\bar{s}} - (w_{\bar{s}} - \bar{c}) p_{\bar{s}-1}/w_{\bar{s}-1} \rfloor\}$;
10.         $\tilde{z} := \max\{\tilde{z}, \tilde{p} - p_j + \bar{p}_{\bar{s}-1}\}$
          end
        else
          begin
11.         $u_j^0 := \tilde{p} - p_j + \bar{p}_f + \lfloor (\tilde{c} + w_j - \bar{w}_f) p_f/w_f \rfloor$;
12.         $\tilde{z} := \max\{\tilde{z}, \tilde{p} - p_j + \bar{p}_f\}$
          end
      end
13. for each $j \in \tilde{L} \cup \{s, \ldots, f\}$ do
      begin
14.     if $\tilde{c} - w_j \geqslant \bar{w}_1$ then
          begin
15.         find, through binary search, $\bar{s} \in \tilde{F}$ such that $\bar{w}_{\bar{s}-1} \leqslant \tilde{c} - w_j < \bar{w}_{\bar{s}}$;
16.         $\bar{c} := \tilde{c} - w_j - \bar{w}_{\bar{s}-1}$;
17.         $u_j^1 := \tilde{p} + p_j + \bar{p}_{\bar{s}-1} + \max\{\lfloor \bar{c} p_{\bar{s}+1}/w_{\bar{s}+1} \rfloor, \lfloor p_{\bar{s}} - (w_{\bar{s}} - \bar{c}) p_{\bar{s}-1}/w_{\bar{s}-1} \rfloor\}$;
18.         $\tilde{z} := \max\{\tilde{z}, \tilde{p} + p_j + \bar{p}_{\bar{s}-1}\}$
          end
        else
          begin
19.         $u_j^1 := \lfloor \tilde{p} + p_j + (\tilde{c} - w_j) p_1/w_1 \rfloor$;
20.         if $\tilde{c} - w_j \geqslant 0$ then $\tilde{z} := \max\{\tilde{z}, \tilde{p} + p_j\}$
          end
      end;
```

```
21. $N_0 := \{j \in \tilde{L} \cup \{s, \ldots, f\}: u_j^1 \leqslant \tilde{z}\}$;
22. $N_1 := \{j \in \tilde{G} \cup \{1, \ldots, s\}: u_j^0 \leqslant \tilde{z}\}$.
```

The heaviest computations are involved by loops 5–12 and 13–20: for $O(n)$ times a statement of time complexity $O(\log |\tilde{F}|)$ (lines 7 and 15) is executed. The overall time complexity is thus $O(n \log |\tilde{F}|)$, i.e., $O(n)$ for fixed $|\tilde{F}|$.

4. Algorithm

4.1. *Core Problem*

The computational experiments from the literature show that many instances of KP can be solved by branch-and-bound algorithms for very high values of n. For such problems, preliminary sorting of all items according to decreasing p_j/w_j ratios requires a large fraction of the overall computing time. On the other hand, if we define, with respect to sorted items and to an optimal solution (x_j^*), $j_1 = \min \{j: x^*_j = 0\}$, $j_2 = \max \{j: x_j^* = 1\}$, it turns out that $\hat{n} = j_2 - j_1 + 1$ is, in general, a very small fraction of n. Hence, if we knew "a priori" j_1 and j_2, we could easily solve the problem by setting $x_j^* = 1$ for all j such that $p_j/w_j > p_{j_1}/w_{j_1}$ and $x_j^* = 0$ for all j such that $p_j/w_j < p_{j_2}/w_{j_2}$, sorting the remaining $\hat{n}$ items and separating them through branch-and-bound. Clearly, j_1 and j_2 (which define the so-called *core problem*) cannot be "a priori" identified, but a good approximation can generally be obtained by prefixing a value for $\hat{n}$ and choosing j_1 and j_2 such that $j_1 < s < j_2$, where s is the critical item. This kind of approach was first proposed by Balas and Zemel (1980).

The procedure CORE presented in this section receives in input an instance of KP and three parameters: ϑ (desired core problem size), α (tolerance) and η (bound on the number of iterations). It returns a partition $(\tilde{F}, \tilde{L}, \tilde{G})$ of N, where $\tilde{L}$ and $\tilde{G}$ are, respectively, approximations of $\{j_2 + 1, \ldots, n\}$ and $\{1, \ldots, j_1 - 1\}$, while $\tilde{F}$ defines an approximate core problem having residual capacity $\tilde{c} = c - \sum_{j \in \tilde{G}} w_j$, such that:

(i) $(1 - \alpha)\vartheta \leqslant |\tilde{F}| \leqslant (1 + \alpha)\vartheta$;

(ii) $\sum_{j \in \tilde{F}} w_j > \tilde{c} > 0$;

(iii) $\max \{p_k/w_k: k \in \tilde{L}\} \leqslant p_j/w_j \leqslant \min \{p_k/w_k: k \in \tilde{G}\}$ for all $j \in \tilde{F}$.

$\tilde{L}$ and $\tilde{G}$ are initialized to empty, and $\tilde{F}$ to N. At any iteration (lines 7–26), we try to move elements from $\tilde{F}$ to $\tilde{G}$ or $\tilde{L}$ until $|\tilde{F}|$ is inside the prefixed range. This is obtained by generating a tentative value λ ($\lambda = p_j/w_j$ for some $j \in \tilde{F}$) and partitioning $\tilde{F}$ into three sets of items j such that p_j/w_j is less than λ (set FL), equal to λ (set FF) or greater than λ (set FG). Three possibilities are then considered, according to the value of the current residual capacity $\tilde{c}$:

(a) $\sum_{j \in FG} w_j \geqslant \tilde{c}$ (lines 17–20), i.e., $\lambda < p_s/w_s$: if $|FG|$ is large enough, we move the elements of FL and FF from $\tilde{F}$ to $\tilde{L}$ and increase λ (lines 18–20); otherwise we decrease λ (line 17) so, at the next iteration, $|FG|$ will be larger;

(b) $\sum_{j \in FG \cup FF} w_j < \tilde{c}$ (lines 21–25), i.e., $\lambda > p_s/w_s$: if $|FL|$ is large enough, we move the elements of FG and FF from $\tilde{F}$ to $\tilde{G}$ and decrease λ (lines 22–25); otherwise we increase λ (line 21);

(c) $\sum_{j \in FG} w_j < \tilde{c} \leqslant \sum_{j \in FG \cup FF} w_j$ (lines 11–15), i.e., $\lambda = p_s/w_s$: if $|FF|$ is large enough, the desired core problem is defined (lines 12–14); otherwise λ is increased or decreased according to the value of $|FG \cup FF|$.

In the following description of procedure CORE, $M3\ (S)$ denotes the median of the profit/weight ratios of the first three elements of S. If the desired $\tilde{F}$ is not obtained within η iterations, execution is halted and the current $\tilde{F}$ is returned: in this case condition (i) above is not satisfied, i.e., $|\tilde{F}|$ is not inside the prefixed range.

Procedure CORE $(\vartheta, \alpha, \eta, \tilde{F}, \tilde{L}, \tilde{G})$:

```
1. $\tilde{L} := \tilde{G} := \phi$;
2. $\tilde{F} := N$;
```

```
 3. $\tilde{c} := c$;
 4. $k := 0$;
 5. $\lambda := M3(\tilde{F})$;
 6. while $|\tilde{F}| > (1+\alpha)\vartheta$ and $k < \eta$ do
      begin
 7.     partition $\tilde{F}$ into ($FL = \{j \in \tilde{F}: p_j/w_j < \lambda\}$, $FG = \{j \in \tilde{F}: p_j/w_j > \lambda\}$, $FF$
        $= \{j \in \tilde{F}: p_j/w_j = \lambda\}$);
 8.     $a := \sum_{j \in FG} w_j$;
 9.     $b := a + \sum_{j \in FF} w_j$;
10.     if $a < \tilde{c} \leqslant b$ then
11.       if $|FF| \geqslant (1-\alpha)\vartheta$ then
            begin
12.           assuming $FF = \{f_1, \dots, f_{|FF|}\}$, find the minimum $s$ such that $\sum_{i=1}^{s} w_{f_i}$
              $> \tilde{c} - a$ and set $\tilde{F} := \{f_r, \dots, f_t\}$ with $r, t$ such that $t - r + 1$ is as close
              as possible to $\vartheta$ and $s$ to $(t+r)/2$;
13.           $\tilde{L} := \tilde{L} \cup FL \cup \{f_{t+1}, \dots, f_{|FF|}\}$;
14.           $\tilde{G} := \tilde{G} \cup FG \cup \{f_1, \dots, f_{r-1}\}$
            end
15.       else if $|FG \cup FF| < \vartheta$ then $\lambda := M3(FL)$ else $\lambda := M3(FG)$
16.     else if $a \geqslant \tilde{c}$ then
17.           if $|FG| < (1-\alpha)\vartheta$ then $\lambda := M3(FL)$ else
                begin
18.               $\tilde{L} := \tilde{L} \cup FL \cup FF$;
19.               $\tilde{F} := FG$;
20.               $\lambda := M3(\tilde{F})$
                end
21.         else if $|FL| < (1-\alpha)\vartheta$ then $\lambda := M3(FG)$ else
                begin
22.               $\tilde{G} := \tilde{G} \cup FG \cup FF$;
23.               $\tilde{F} := FL$;
24.               $\tilde{c} := \tilde{c} - b$;
25.               $\lambda := M3(\tilde{F})$
                end;
26.     $k := k + 1$
      end.
```

The heaviest computations are at lines 7, 8, 9 and 12, all requiring $O(n)$ time. Hence, if η is a constant, the procedure runs in linear time.

4.2. *Algorithm Description*

In addition to procedures RCS (§3), RPS (§3) and CORE (§4.1), the algorithm we propose for KP makes use of procedures SORT and BB below:

Procedure SORT (S):

sort the items in $S \subseteq N$ according to decreasing p_j/w_j ratios.

Procedure BB (S, N_1, *flag*, $\hat{x}$, $\bar{u}$):

assuming that the items in $S \subseteq N$ are sorted according to decreasing p_j/w_j ratios, determine, through the branch-and-bound algorithm of Martello and Toth (1977), the optimal solution ($\hat{x}_j$) ($j \in S$) of the problem defined by the items in S with residual capacity $c - \sum_{j \in N_1} w_j$;

in addition, if *flag* $= 1$, determine, for the original problem, upper bound $\bar{u}$ as described in §2.

The algorithm starts by determining an approximate core problem $\tilde{F}$ (lines 1–2). If the size of this problem is too large, the optimal solution is obtained through complete sorting, reduction and branch-and-bound (lines 14–18). Otherwise, sorting and branch-and-bound are applied only to the items in $\tilde{F}$ (lines 4–5), obtaining an approximate solution which, if its value equals that of upper bound $\bar{u}$, is also optimal (lines 6–7). If this is not the case, the original problem is reduced through RPS (line 8). If the resulting set of free items is a subset of $\tilde{F}$, then $\tilde{F}$ contains the true core problem, so the approximate solution above is again optimal (line 9); otherwise, the optimal solution for the free items is determined through branch-and-bound (lines 10–13).

Algorithm MT2

```
1.  define parameters $\vartheta$, $\alpha$ and $\eta$ (see §5);
2.  CORE ($\vartheta, \alpha, \eta, \tilde{F}, \tilde{L}, \tilde{G}$);
3.  if $|\tilde{F}| \leqslant (1-\alpha)n$ then
      begin
4.      SORT ($\tilde{F}$);
5.      BB($\tilde{F}, \tilde{G}, 1, \hat{x}, \bar{u}$);
6.      $\tilde{z} := \sum_{j\in\tilde{G}} p_j + \sum_{j\in\tilde{F}} p_j\hat{x}_j$;
7.      if $\tilde{z} = \bar{u}$ then set $x_j = 1$ for all $j \in \tilde{G} \cup \{k: \hat{x}_k = 1\}$
        else
          begin
8.          RPS ($\tilde{z}, \tilde{F}, \tilde{L}, \tilde{G}, N_0, N_1$);
9.          if $N_1 \supseteq \tilde{G}$ and $N_0 \supseteq \tilde{L}$ then set $x_j = 1$ for all $j \in \tilde{G} \cup \{k: \hat{x}_k = 1\}$
            else
              begin
10.             $F := N \setminus (N_0 \cup N_1)$;
11.             SORT($F$);
12.             BB($F, N_1, 0, \hat{x}, \bar{u}$);
13.             set $x_j = 1$ for all $j \in N_1 \cup \{k: \hat{x}_k = 1\}$
              end
          end
      end
    else
      begin
14.     SORT($N$);
15.     RCS ($N_0, N_1, \tilde{z}$);
16.     $F := N \setminus (N_0 \cup N_1)$;
17.     BB($F, N_1, 0, \hat{x}, \bar{u}$);
18.     set $x_j = 1$ for all $j \in N_1 \cup \{k: \hat{x}_k = 1\}$
      end
19. if $\sum_{j:x_j=1} p_j \geqslant \tilde{z}$ then the optimal solution is $x_j$
    else the optimal solution is that corresponding to $\tilde{z}$.
```

5. Computational Experiments

Algorithm MT2 of the previous section was coded in ANSI Fortran. The list is available on request from the authors. Table 1 gives the main characteristics of the Fortran codes available from the literature—to our knowledge—for the solution of KP. In this section we examine the computational efficiency of these codes through experiments performed on a CDC-Cyber 730. The parameters needed by MT2 (line 1) were empirically determined as

$$\vartheta = \begin{cases} n & \text{if} \quad n < 100; \\ \sqrt{n} & \text{otherwise;} \end{cases}$$

TABLE 1
Fortran Codes for KP

Code	Authors	Core memory requirements	Number of statements	List
KNAP	Nauss (1976)	$8n$	280	Available from the author
KP01	Martello-Toth (1978)	$8n$	180	Included (Martello-Toth 1978)
FPK79	Fayard-Plateau (1982)	$7n$	600	Included (Fayard-Plateau 1982)
MT2	Martello-Toth	$8n$	1000	Available from the authors

$$\alpha = 0.2;$$

$$\eta = 20.$$

Three types of randomly-generated data sets have been considered:
uncorrelated: p_j and w_j uniformly random in [1, 100];
weakly correlated: w_j uniformly random in [1, 100], p_j uniformly random in $[w_j - 10, w_j + 10]$;
strongly correlated: w_j uniformly random in [1, 100], $p_j = w_j + 10$.

We have considered small-size problems ($n = 50, 100, 200$; $c = 0.5 \sum_{j\in N} w_j$, $c = 200$) and large-size problems ($n = 500, 1000, 2000, 5000, 10000$; $c = 0.5 \sum_{j\in N} w_j$). The entries in all the tables give the average running time, expressed in seconds, computed over 20 problem instances.

Code FKP79 includes its own sorting procedure. The sortings needed by KNAP, KP01 and MT2 have been obtained through a subroutine (included in MT2) derived from subroutine SORTZV of the CERN Library. For $n = 50, 100, 200, 500, 1000, 2000, 5000, 10000$ this subroutine requires respectively 0.008, 0.018, 0.041, 0.114,

TABLE 2
Uncorrelated Problems. p_j and w_j Uniformly Random in [1, 100]. *CDC-Cyber* 730 *Seconds. Average Times over* 20 *Problems*

c	n	KNAP	KP01	FPK79	MT2
200	50	0.015	0.015	0.013	0.014
	100	0.025	0.026	0.018	0.026
	200	0.055	0.051	0.032	0.043
$0.5 \sum_{j\in N} w_j$	50	0.015	0.016	0.013	0.015
	100	0.029	0.030	0.021	0.025
	200	0.073	0.068	0.053	0.044

TABLE 3
Weakly Correlated Problems. w_j Uniformly Random in [1, 100], *p_j in* $[w_j - 10, w_j + 10]$. *CDC-Cyber* 730 *Seconds. Average Times over* 20 *Problems*

c	n	KNAP	KP01	FPK79	MT2
200	50	0.019	0.017	0.016	0.016
	100	0.038	0.032	0.023	0.023
	200	0.060	0.055	0.030	0.036
$0.5 \sum_{j\in N} w_j$	50	0.035	0.022	0.021	0.020
	100	0.086	0.040	0.039	0.034
	200	0.151	0.069	0.057	0.040

TABLE 4
Strongly Correlated Problems. w_j Uniformly Random in [1, 100], $p_j = w_j + 10$.
CDC-Cyber 730 *Seconds. Average Times over* 20 *Problems*

c	n	KNAP	KP01	FPK79	MT2
200	50	0.117	0.028	0.047	0.029
	100	0.259	0.052	0.096	0.066
	200	3.595	0.367	0.928	0.616
$0.5\ \Sigma_{j \in N} w_j$	50	55.555	4.870	17.895	5.721
	100	time limit	100.050	time limit	120.500
	200	—	time limit	—	time limit

0.250, 0.529, 1.416, 3.010 seconds on a CDC-Cyber 730. All times given in the tables include the corresponding sorting time. A time limit of 500 seconds was assigned to each code to solve the problems generated for each data set and value of c. In the case of a time limit, if the number of problems solved is significant, we give the corresponding average time.

Tables 2, 3 and 4 compare the codes on small-size problems. Uncorrelated and weakly correlated problems are solved very quickly by all the codes. FPK79 and MT2 are practically equivalent and better than the others. For strongly correlated problems, KP01 and MT2 are clearly the best codes.

Large-size problems are considered in Tables 5 and 6. For the uncorrelated case, FPK79 and MT2 are confirmed as the best codes. For the weakly correlated case, MT2 is clearly superior to all the other codes.

We do not consider large-size strongly correlated problems because of their intrinsic difficulty (see Table 4). We consider, however, the extreme case of correlation between profits and weights, i.e., $p_j = w_j$ for all j. This is the *subset-sum problem* which arises quite frequently in practice, for example in cutting-stock applications. One could expect intractability of these problems, mainly because (i) all the upper bounds for KP give the trivial value c so (ii) the reduction procedures have no effect. Table 7 shows, on

TABLE 5
Uncorrelated Problems. p_j and w_j Uniformly Random in [1, 100]. *CDC-Cyber* 730 *Seconds.*
Average Times over 20 *Problems*

c	n	KNAP	KP01	FPK79	MT2
$0.5\ \Sigma_{j \in N} w_j$	500	0.168	0.148	0.078	0.101
	1000	0.380	0.297	0.175	0.168
	2000	1.246	0.590	0.295	0.297
	5000	92.800	1.555	0.674	0.650
	10000	time limit	3.219	1.162	1.265

TABLE 6
Weakly Correlated Problems. w_j Uniformly Random in [1, 100], p_j *in* $[w_j - 10, w_j + 10]$.
CDC-Cyber 730 *Seconds. Average Times over* 20 *Problems*

c	n	KNAP	KP01	FPK79	MT2
$0.5\ \Sigma_{j \in N} w_j$	500	1.425	0.138	0.106	0.084
	1000	time limit	0.276	0.316	0.129
	2000	—	0.594	24.777	0.292
	5000	—	1.509	time limit	0.566
	10000	—	3.383	—	1.189

TABLE 7

Subset-Sum Problems. w_j Uniformly Random in [1, 100], $p_j = w_j$. *CDC-Cyber* 730 *Seconds. Average Times over* 20 *Problems*

c	n	KNAP	KP01	FPK79	MT2
$0.5 \sum_{j \in N} w_j$	50	0.026	0.013	0.041	0.010
	100	0.047	0.023	0.124	0.023
	200	0.116	0.043	0.288	0.028
	500	0.279	0.120	1.825	0.051
	1000	0.389	0.251	6.364	0.090
	2000	0.583	0.548	23.357	0.184
	5000	1.675	1.538	time limit	0.444
	10000	3.505	3.214	—	0.939

the contrary, that, with the exception of FPK79, all the codes (and in particular MT2) are capable of solving large problems of this kind efficiently. Such behaviour is explained by the fact that, for uniformly random problems, a large number of solutions have the optimal value c, so execution of branch-and-bound algorithms terminates in general very early.

The results of the present section indicate that MT2 is the best code available from the literature for KP. In addition, to our knowledge, no other algorithm from the literature (including those whose code is not available) has obtained better results. Instead, for the subset-sum problem, specialized algorithms—whose corresponding codes, however, are not available—can give much smaller computing times (see e.g. Balas and Zemel 1980, Martello and Toth 1987).[1]

[1] This work was supported by Ministero della Pubblica Istruzione, Italy.

References

BALAS, E. AND E. ZEMEL, "An Algorithm for Large Zero-One Knapsack Problems," *Oper. Res.*, 28 (1980), 1130–1154.

DANTZIG, G. B., "Discrete Variable Extremum Problems," *Oper. Res.*, 5 (1957), 266–277.

DEMBO, R. S. AND P. L. HAMMER, "A Reduction Algorithm for Knapsack Problems." *Methods Oper. Res.*, 36 (1980), 49–60.

DUDZINSKI, K. AND S. WALUKIEWICZ, "Upper Bounds for the 0-1 Knapsack Problem," Report MPD-10-49/84, Systems Research Institute, Warsaw, 1984.

FAYARD, D. AND G. PLATEAU, "An Algorithm for the Solution of the 0-1 Knapsack Problem," *Computing*, 28 (1982), 269–287.

GAREY, M. R. AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

HOROWITZ, E. AND S. SAHNI, "Computing Partitions with Applications to the Knapsack Problem," *J. Assoc. Comput. Mach.*, 21 (1974), 277–292.

INGARGIOLA, G. P. AND J. F. KORSH, "A Reduction Algorithm for Zero-One Single Knapsack Problems," *Management Sci.*, 20 (1973), 460–463.

MARTELLO, S. AND P. TOTH, "An Upper Bound for the Zero-One Knapsack Problem and a Branch and Bound Algorithm" *European J. Oper. Res.*, 1 (1977), 169–175.

——— AND ———, "Algorithm for the Solution of the 0-1 Single Knapsack Problem," *Computing*, 21 (1978), 81–86.

——— AND ———, "A Mixture of Dynamic Programming and Branch-and-Bound for the Subset-Sum Problem" *Management Sci.*, 30 (1984), 765–771.

——— AND ———, "Algorithms for Knapsack Problems" in S. Martello, G. Laporte, M. Minoux and C. Ribeiro (Eds.), *Surveys in Combinatorial Optimization*, Ann. Discrete Math. 31, North-Holland, Amsterdam, 1987.

MÜLLER-MERBACK, H., "An Improved Upper Bound for the Zero-One Knapsack Problem: A Note on the Paper by Martello and Toth," *European J. Oper. Res.*, 2 (1978), 212–213.

NAUSS, R. M., "An Efficient Algorithm for the 0-1 Knapsack Problem," *Management Sci.*, 23 (1976), 27–31.

TOTH, P., "A New Reduction Algorithm for 0-1 Knapsack Problems," Presented at the ORSA/TIMS Joint National Meeting, Miami, 1976.