UPPER BOUNDS AND ALGORITHMS FOR HARD 0-1 KNAPSACK PROBLEMS

SILVANO MARTELLO AND PAOLO TOTH

University of Bologna, Italy

(Received December 1993; revisions received April 1995, November 1995; accepted December 1995)

It is well-known that many instances of the 0-1 knapsack problem can be effectively solved to optimality also for very large values of n (the number of binary variables), while other instances cannot be solved for n equal to only a few hundreds. We propose upper bounds obtained from the mathematical model of the problem by adding valid inequalities on the cardinality of an optimal solution, and relaxing it in a Lagrangian fashion. We then introduce a specialized iterative technique for determining the optimal Lagrangian multipliers in polynomial time. A branch-and-bound algorithm is finally developed. Computational experiments prove that several classes of hard instances are effectively solved even for large values of n.

Knapsack problems are frequently used to model industrial situations such as, for example, capital budgeting, cargo loading, or cutting stock. In addition, they often arise as subproblems to be solved when more complex combinatorial optimization problems are tackled. The most famous knapsack-type problem is the single-constraint binary version. Given n *items,* with p_j = *profit* of item j, and w_j = *weight* of item j ($j = 1, \ldots, n$), and a *knapsack* of capacity c (with p_j, w_j, and c positive integers), the *0-1 Knapsack Problem* (KP01) is to select a subset of the items whose total weight does not exceed c and whose total profit is a maximum.

The problem, which is known to be *NP*-hard, has been intensively studied in the last decades and several exact and approximate algorithms for its solution can be found in the literature (see Martello and Toth 1990 for a comprehensive survey including, on diskette, Fortran implementations of such algorithms). A peculiarity of KP01 is that many instances can be quickly solved, also for very large values of n, while other instances cannot be solved for n equal to only a few hundreds. Indeed, let us consider the following well-known exact algorithms for KP01:

HS: Horowitz and Sahni (1974);

NA: Nauss (1976);

DT: Toth (1980);

FP: Fayard and Plateau (1982);

MT2: Martello and Toth (1990).

HS, NA, FP, and MT2 are branch-and-bound algorithms, while DT is a dynamic programming approach. NA, FP, and MT2 include a reduction phase (a preprocessing routine which fixes the optimal value of a number of variables); HS and DT can be similarly accelerated by initially executing the reduction procedure MTR (Martello and Toth 1988, 1990). The computational experiments commonly performed in the literature consider the following three main classes of randomly generated instances (where a and δ are prefixed constants):

Uncorrelated:	w_j uniformly random in $[1, a]$,
	p_j uniformly random in $[1, a]$,
	$c = \frac{1}{2}(w_1 + \ldots + w_n)$;
Weakly Correlated:	w_j uniformly random in $[1, a]$,
	p_j uniformly random in $[w_j - \delta, w_j + \delta]$,
	$c = \frac{1}{2}(w_1 + \ldots + w_n)$;
Strongly Correlated:	w_j uniformly random in $[1, a]$,
	$p_j = w_j + \delta$,
	$c = \frac{1}{2}(w_1 + \ldots + w_n)$.

The entries in Table I (taken from Martello and Toth 1990) give the average running times over 20 instances, obtained on a CDC-Cyber 730 for the Fortran implementations of the algorithms above. The entries "time" indicate that the algorithm did not solve the 60 instances of a class within 500 seconds. Uncorrelated and weakly correlated instances are easily solved (Martello and Toth 1990 report experiments with algorithm MT2 solving problems of these types with up to more than 100,000 variables). The strongly correlated instances are, on the other hand, very difficult: they can be solved, for small values of n and c, by the dynamic programming algorithm which, however, cannot solve larger instances because of the excessive space and time requirements. These are not the only instances for which the algorithms from the literature cannot find the optimal solution: in Section 4 we describe other hard classes of random problems.

Recently, Pandit and Ravi Kumar (1993) presented a specialized enumerative algorithm for solving instances of KP01 in which $p_j = \alpha\, w_j + \delta$ ($j = 1, \ldots, n$), where α and δ are prefixed constants. The algorithm consists of a lexicographic search in which, for different possible values of an integer k, the best solution which inserts exactly k items

Subject classifications: Programming, integer, algorithms, branch-and-bound: 0-1 knapsack problem. Programming, integer, algorithms, relaxation/subgradient: valid inequalities, Lagrangian relaxations.
Area of review: OPTIMIZATION.

Table I
Average Times over 20 Problems—CDC Cyber 730 Seconds—w_j Uniformly Random in [1,100]; $c = \frac{1}{2}(w_1 + \ldots + w_n)$

Class	n	MTR + HS	NA	FP	MT2	MTR +DT
Uncorrelated	50	0.016	0.015	0.013	0.013	0.020
p_j u. r. in [1,100]	100	0.028	0.029	0.021	0.026	0.043
	200	0.065	0.073	0.053	0.057	0.090
Weakly Correlated	50	0.025	0.035	0.021	0.020	0.071
p_j u. r. in $[w_j - 10, w_j + 10]$	100	0.042	0.086	0.039	0.031	0.158
	200	0.070	0.151	0.057	0.055	0.223
Strongly Correlated	50	time	time	17.895	4.019	0.370
$p_j = w_j + 10$	100	—	—	time	time	1.409
	200	—	—	—	—	3.936

in the knapsack is determined. As will be seen later, this algorithm efficiently solves strongly correlated instances. However, it cannot be applied to problems in which the profits do not satisfy the above condition. Hence determining an effective algorithm for general hard KP01 instances is very important for a practical solution of the problem.

In this paper we propose a new approach, which can effectively solve large, strongly correlated instances as well as other hard classes. In Section 1 we discuss the continuous relaxation of the mathematical model of the problem, and in Section 2 we present upper bounds obtained by adding to the model valid inequalities on the cardinality of an optimal solution and relaxing it in a Lagrangian fashion. We introduce a specialized iterative technique for determining the Lagrangian multipliers. These results are used in Section 3 to obtain a branch-and-bound algorithm for the exact solution of the problem. The algorithm is experimentally evaluated in Section 4 and successfully compared with the most effective algorithms in the literature.

1. CONTINUOUS RELAXATION

The integer linear programming model of the 0-1 knapsack problem (KP01) is:

$$z = \max \sum_{j=1}^{n} p_j x_j, \tag{1}$$

$$\sum_{j=1}^{n} w_j x_j \leq c, \tag{2}$$

$$x_j \in \{0, 1\} \quad (j = 1, \ldots, n), \tag{3}$$

where

$$x_j = \begin{cases} 1 & \text{if item } j \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases}$$

We assume, without loss of generality, that

$$\sum_{j=1}^{n} w_j > c, \tag{4}$$

$$\max_j \{w_j\} \leq c, \tag{5}$$

$$p_j/w_j \geq p_{j+1}/w_{j+1} \quad (j = 1, \ldots, n-1). \tag{6}$$

The best-known upper bound for KP01 is produced by its continuous relaxation) obtained by replacing (3) with $0 \leq x_j \leq 1$ $(j = 1, \ldots, n)$. Dantzig (1957) proved that, once the items are sorted according to (6), the optimal solution $(\bar{x})$ is obtained by determining the item s (*critical item*) that satisfies

$$\sum_{j=1}^{s-1} w_j \leq c < \sum_{j=1}^{s} w_j, \tag{7}$$

and defining $\bar{x}_j = 1$ for $j = 1, \ldots, s-1$, $\bar{x}_s = (c - \sum_{j=1}^{s-1} w_j)/w_s$, $\bar{x}_j = 0$ for $j = s+1, \ldots, n$. Since z must be integer, the corresponding upper bound is

$$CU = \sum_{j=1}^{s-1} p_j + \lfloor p_s \bar{x}_s \rfloor. \tag{8}$$

This computation can also produce an approximate solution through the so-called *Greedy Algorithm:* select items $1, \ldots, s-1$, then consider items $s+1, \ldots, n$ selecting the current item iff its weight does not exceed the current residual capacity.

In the following we will consider as subproblems relaxations of KP01 in which the assumption that all profits are positive integers does not hold. For such cases, if $p_s < 0$, the continuous solution must be obtained by defining $t = \max\{j : p_j > 0\}$ and setting $\bar{x}_j = 1$ for $j = 1, \ldots, t$, $\bar{x}_j = 0$ for $j = t+1, \ldots, n$, hence obtaining the upper bound $CU = \lfloor \sum_{j=1}^{t} p_j \rfloor$.

The computation of CU has time complexity $O(n)$, plus $O(n \log n)$ if the items have to be sorted according to (6) (but in this case an overall $O(n)$ time complexity can be obtained through partitioning techniques, see Balas and Zemel 1980). It is worth noting that CU coincides with the best bound which can be obtained from the Lagrangian relaxation

$$z(\lambda) = \max \sum_{j=1}^{n} p_j x_j + \lambda \left(c - \sum_{j=1}^{n} w_j x_j \right), \tag{9}$$

$$x_j \in \{0, 1\} \quad (j = 1, \ldots, n), \tag{10}$$

since this problem has the integrality property. (The optimal Lagrangian multiplier is easily determined as $\lambda = p_s/w_s$.)

Several improvements of CU have been proposed in the literature (see Martello and Toth 1990, Section 2.2). In the following we introduce new upper bounds obtained from different continuous and Lagrangian relaxations.

2. ADDITION OF VALID INEQUALITIES

In this section we show how a single valid inequality, based on the maximum or minimum cardinality of an optimal solution, is added to the mathematical model of KP01. The new constraint is redundant for the integer problem, but strengthens its continuous relaxation. We determine the new upper bounds by efficiently solving the resulting continuous problems through Lagrangian techniques, thus avoiding the use of an LP solver. Our method can be seen as a specialization of the modern *cutting plane* methods (see, e.g., Crowder et al. 1983), in which a series of valid inequalities are added to the model and the solution of the continuous relaxation of the resulting problem is obtained through an LP solver.

2.1. A Maximum Cardinality Upper Bound

Let $C(x) = \sum_{j=1}^{n} x_j$ denote the *cardinality* of a solution (x), and suppose that a value K is known such that $C(x) \leq K$ in any optimal solution (x) to KP01. A valid value for K can be determined by finding the minimum value such that the sum of the $K + 1$ items of smallest weight exceeds the capacity, i.e.,

$$K = \min\left\{ h : \sum_{j=1}^{h} w_{d(j)} > c \right\} - 1, \tag{11}$$

where $w_{d(j)} \leq w_{d(j+1)}$ $(j = 1, \ldots, n-1)$. We can then obtain an equivalent model, KP01$^{\leq}$, given by (1)–(3) and

$$\sum_{j=1}^{n} x_j \leq K. \tag{12}$$

Whenever the continuous solution to KP01 violates (12) (i.e., $s > K$, with $\bar{x}_s > 0$), a better upper bound is given by the solution value of the continuous relaxation of KP01$^{\leq}$. An efficient way to determine such a solution without using an LP solver is to dualize constraint (12) through any nonnegative Lagrangian multiplier λ:

$$(\mathrm{L1}(\lambda)) \quad U1(\lambda) = \max \sum_{j=1}^{n} p_j x_j + \lambda\left(K - \sum_{j=1}^{n} x_j \right)$$

$$= \lambda K + \max \sum_{j=1}^{n} \bar{p}_j x_j, \tag{13}$$

$$\sum_{j=1}^{n} w_j x_j \leq c, \tag{14}$$

$$0 \leq x_j \leq 1 (j = 1, \ldots, n), \tag{15}$$

where $\bar{p}_j = p_j - \lambda$ $(j = 1, \ldots, n)$ (and note that $\bar{p}_j$ can take nonpositive values). Indeed, it is known that the solution to the dual Lagrangian problem, in our case

$$U1(\lambda^*) = \min_{\lambda \geq 0}\{U1(\lambda)\}, \tag{16}$$

provides the solution value of the continuous relaxation of the given problem, and that λ^* is the value of the optimal dual variable associated with (12). This computation can be carried out very efficiently, since: (a) for any tentative value of λ the relaxed problem can easily be solved (as shown in Section 1); (b) as we shall see, a specialized iterative technique can be used for determining λ^*; and (c) the special structure of the problem allows a limited number of λ multipliers to be considered.

Let $C(x, \lambda) = \sum_{j=1}^{n} x_j$, where (x) is the solution to L1(λ), and observe that $C(x, \lambda)$ is in general a fractional value. Assume that the items have been sorted (see (6)) by breaking ties according to decreasing w_j values, so that $C(x, \lambda)$ is not multivalued at breakpoints. We show that $C(x, \lambda)$ is a monotonous function of λ.

Lemma 1. *Given two items a and b, and two values λ', λ'' such that $\lambda' < \lambda''$, if $(p_a - \lambda')/w_a \geq (p_b - \lambda')/w_b$ and $(p_a - \lambda'')/w_a \leq (p_b - \lambda'')/w_b$ then $w_b \geq w_a$.*

Proof. Immediate, since by subtracting the second inequality from the first, we get $(\lambda'' - \lambda')/w_a \geq (\lambda'' - \lambda')/w_b$. □

Corollary 1. *$C(x, \lambda)$ is monotonically nonincreasing as λ increases.*

Proof. Lemma 1 shows that, as λ increases, in the sequence sorted according to decreasing $(p_j - \lambda)/w_j$ ratios (see(6)), the items with larger weight move to the first positions. Hence $C(x, \lambda)$ can never increase. □

From the complementary slackness conditions we know that the optimal multiplier, λ^*, is the one for which $C(x, \lambda^*) = K$. Hence, from Corollary 1, λ^* can be determined through a binary search in which, at each iteration, the current value of λ is increased (resp. decreased) if $C(x, \lambda) > K$ (resp. $< K$). Given a lower bound λ_1 and an upper bound λ_2 on λ^*, at each iteration the procedure defines the tentative value as $\lambda = (\lambda_1 + \lambda_2)/2$: if $C(x, \lambda) = K$ we have found $\lambda^* = \lambda$; otherwise, the value of λ is assigned to λ_1 (if $C(x, \lambda) > K$) or to λ_2 (if $C(x, \lambda) < K$). Initially $\lambda_2 =$ Kth largest p_j value and $\lambda = \lambda_1 = 0$ (so, from (6), no sorting is needed). The procedure terminates when $\lambda_1 = \lambda_2$. For each tentative value of λ, the solution of L1(λ) requires computation of the current critical item, $s(\lambda)$. Due to the special structure of the problem, the number of these computations can be reduced as follows.

Suppose that, for current λ, $C(x, \lambda) > K$ holds: we can determine a better (higher) λ', producing the same solution as λ (i.e., requiring no new computation of the critical item), as the maximum value satisfying the following relations (where we assume that the items are numbered according to decreasing $(p_j - \lambda)/w_j$ ratios):

$$\frac{p_j - \lambda'}{w_j} \geq \frac{p_{s(\lambda)} - \lambda'}{w_{s(\lambda)}} \quad (j = 1, \ldots, s(\lambda) - 1), \tag{17}$$

$$\frac{p_j - \lambda'}{w_j} \leq \frac{p_{s(\lambda)} - \lambda'}{w_{s(\lambda)}} \quad (j = s(\lambda) + 1, \ldots, n), \tag{18}$$

i.e., since from Lemma 1 Equation (17) (resp. (18)) is automatically satisfied if $w_j \geqslant w_{s(\lambda)}$ (resp. $w_j \leqslant w_{s(\lambda)}$),

$$\lambda' = \min\left(\left\{\frac{p_j w_{s(\lambda)} - p_{s(\lambda)} w_j}{w_{s(\lambda)} - w_j} : j < s(\lambda) \quad \text{and } w_j < w_{s(\lambda)}\right\},\right.$$
$$\left.\left\{\frac{p_{s(\lambda)} w_j - p_j w_{s(\lambda)}}{w_j - w_{s(\lambda)}} : j > s(\lambda) \quad \text{and } w_j > w_{w_{s(\lambda)}}\right\}\right). \tag{19}$$

(Observe that, with our choice of K, $p_{s(\lambda)} - \lambda \geqslant 0$ always holds if $C(x, \lambda) > K$.)

When for the current λ, $C(x, \lambda) < K$ holds, similar arguments lead to a better (lower) λ', producing the same solution as λ (i.e., requiring no new computation of the critical item). In this case from (17) and (18) we get:

$$\lambda' =$$
$$\max\left(\left\{\frac{p_{s(\lambda)} w_j - p_j w_{s(\lambda)}}{w_j - w_{s(\lambda)}} : j < s(\lambda) \quad \text{and } w_j > w_{w_{s(\lambda)}}\right\},\right.$$
$$\left.\left\{\frac{p_j w_{s(\lambda)} - p_{s(\lambda)} w_j}{w_{s(\lambda)} - w_j} : j > s(\lambda) \quad \text{and } w_j < w_{s(\lambda)}\right\}\right). \tag{20}$$

(If, however, $p_{s(\lambda)} - \lambda < 0$, hence the special continuous solution described in Section 1 is used, the improved multiplier must be $\lambda' = \max\{p_j : p_j - \lambda \leqslant 0\}$.)

By comparing $U1(\lambda')$ and $U1(\lambda)$, we see that in both cases the values of $\sum_{j=1}^n p_j x_j$ and $\sum_{j=1}^n x_j$ in (13) do not vary, so $U1(\lambda') \leqslant U1(\lambda)$.

Once λ^* has been found, we have a new upper bound for KP01:

$$U1 = \lfloor U1(\lambda^*) \rfloor, \tag{21}$$

and obviously $U1 \leqslant CU$.

The computation can be further accelerated by determining for each value of λ an approximate solution to KP01 through the greedy algorithm. Let z^a be the value of the best approximate solution computed so far: if $z^a = \lfloor U1(\lambda') \rfloor$ then the search can be terminated by taking $\lambda^* = \lambda'$.

Equations (19) and (20) show that the number of possible different values of λ' is $O(n^2)$. Since each λ' is produced by a computation of $s(\lambda)$, which requires $O(n)$ time, the overall time complexity for the computation of $U1$ is $O(n^3)$. From all our computational experiments it turned out, however, that the average time complexity is much smaller, in practice almost linear, since very few iterations usually produce the optimum. It is worth noting that the computation can also be implemented so as to have time complexity $O(n^2)$. Indeed, we can generate the different $O(n^2)$ values of λ' and perform a binary search taking the median of the current values at each iteration: the resulting number of iterations is $O(\log n^2)$, leading to overall time complexities $O(n \log n)$ for the $s(\lambda')$ computations and $O(n^2)$ for the median computations. This technique, however, requires time proportional to n^2 also in the best case, so on average, it is worse than the previous one.

We finally observe that similar results and the same upper bound value could be obtained by dualizing, in KP01$^{\leqslant}$, constraint (2) instead of (12).

Example 1. *Let* $n = 6$,
$(p_j) = (15, 16, 19, 17, 19, 23)$,
$(w_j) = (10, 12, 15, 14, 17, 21)$, $c = 48$.

We have $K = 3$.

First iteration. $\lambda = 0$, $\lambda_2 = 19$; $s(0) = 4 > K$;
$\lfloor U1(0) \rfloor$ $(= CU) = \lfloor 15 + 16 + 19 + 17 \tfrac{11}{14} \rfloor = 63$; $z^a = 15 + 16 + 19$.

$$\lambda' = \min\left(\left\{\frac{15 \times 14 - 17 \times 10}{14 - 10}, \frac{16 \times 14 - 17 \times 12}{14 - 12}\right\},\right.$$
$$\left.\left\{\frac{17 \times 17 - 19 \times 14}{17 - 14}, \frac{17 \times 21 - 23 \times 14}{21 - 14}\right\}\right) = 5 = \lambda_1,$$

$\lfloor U1(5) \rfloor = \lfloor 5 \times 3 + 10 + 11 + 14 + 12 \tfrac{11}{14} \rfloor = 59.$

Second iteration. $\lambda = 12$: $(\bar{p}_j) = (11, 7, 7, 5, 4, 3)$,
$(w_j) = (21, 15, 17, 14, 12, 10)$;
$s(12) = 3$, $C(x, 12) < K$; $\lfloor U1(12) \rfloor = \lfloor 12 \times 3 + 11 + 7 + 7 \tfrac{12}{17} \rfloor = 58$; $z^a = 23 + 19 + 16 = 58$. Since $\lfloor U1(12) \rfloor = z^a$, we terminate with $\lambda^* = 12$. □

2.2. A Minimum Cardinality Upper Bound

Whenever the continuous solution to KP01 does not violate (12) (hence $CU = U1$), a different bound can be obtained by using a value k such that $C(x) \geqslant k$ in any optimal solution (x) to KP01. Given any approximate solution of value z^a, let

$$k = \max\left\{h : \sum_{j=1}^{h} p_{g(j)} \leqslant z^a\right\} + 1, \tag{22}$$

where $p_{g(j)} \geqslant p_{g(j+1)}$ $(j = 1, \ldots, n - 1)$. Since we are obviously interested only in feasible solutions whose value is strictly greater than z^a, we obtain an equivalent model, **KP01$^{\geqslant}$**, given by (1)–(3) and

$$\sum_{j=1}^{n} x_j \geqslant k. \tag{23}$$

In this case, too, whenever the continuous solution to KP01 violates (23) (i.e., $s \leqslant k$), we obtain a better upper bound by solving the continuous relaxation of **KP01$^{\geqslant}$** through its Lagrangian relaxation (for $\mu \geqslant 0$):

$$(\mathrm{L2}(\mu))\ U2(\mu) = \max \sum_{j=1}^{n} p_j x_j + \mu\left(\sum_{j=1}^{n} x_j - k\right)$$
$$= -\mu k + \max \sum_{j=1}^{n} \tilde{p}_j xj, \tag{24}$$

$$\sum_{j=1}^{n} w_j x_j \leqslant c, \tag{25}$$

$$0 \leqslant x_j \leqslant 1 \qquad (j = 1, \ldots, n), \tag{26}$$

where $\tilde{p}_j = p_j + \mu (j = 1, \ldots, n)$ (note that $\tilde{p}_j$ always takes a positive value).

Let $C(x, \mu) = \sum_{j=1}^{n} x_j$, where (x) is the solution to L2(μ), and assume that the items have been sorted (see (6)) by breaking ties according to increasing w_j values. The following are immediate extensions of Lemma 1 and Corollary 1:

Lemma 2. *Given two items a and b, and two values μ', μ'' such that $\mu' < \mu''$, if $(p_a + \mu')/w_a \geq (p_b + \mu')/w_b$ and $(p_a + u'')/w_a \leq (p_b + \mu'')/w_b$ then $w_b \leq w_a$.*

Corollary 2. *$C(x, \mu)$ is monotonically nondecreasing as μ increases.*

Hence the dual Lagrangian problem

$$U2(\mu^*) = \min_{\mu \geq 0} \{U2(\mu)\}, \tag{27}$$

can be solved efficiently through binary search. In this case we initialize μ and lower bound μ_1 to 0, and upper bound μ_2 to $\max_j\{p_j w_j\} - \min_j\{p_j w_j\}$ (which ensures that the items sorted according to decreasing $\bar{p}_j/w_j$ ratios are also sorted according to increasing weights).

The number of iterations can be reduced through considerations similar to those used in Section 2.1. If, for the current μ, $C(x, \mu) < k$ holds, we obtain a better (higher) μ' producing the same solution as μ as

$$\mu' = \min\left(\left\{\frac{p_j w_{s(m)} - p_{s(\mu)} w_j}{w_j - w_{s(\mu)}} : j < s(\mu) \text{ and } w_j > w_{s(\mu)}\right\},\right.$$
$$\left.\left\{\frac{p_{s(\mu)} w_j - p_j w_{s(\mu)}}{w_{s(\mu)} - w_j} : j > s(\mu) \text{ and } w_j < w_{s(\mu)}\right\}\right); \tag{28}$$

if, on the other hand, $C(x, \mu) > k$ holds, a better (lower) μ' is

$$\mu' = \max\left(\left\{\frac{p_{s(\mu)} w_j - p_j w_{s(\mu)}}{w_{s(\mu)} - w_j} : j < s(\mu) \text{ and } w_j < w_{s(\mu)}\right\},\right.$$
$$\left.\left\{\frac{p_j w_{s(\mu)} - p_{s(\mu)} w_j}{w_j - w_{s(\mu)}} : j > s(\mu) \text{ and } w_j > w_{s(\mu)}\right\}\right). \tag{29}$$

The new upper bound is thus:

$$U2 = \lfloor U2(\mu^*) \rfloor, \tag{30}$$

and obviously $U2 \leq CU$. The time complexity for this computation is the same as for $U1$.

2.3. Other Valid Inequalities

Consider the minimal cover induced by (11):

$$\sum_{j=1}^{K+1} x_{d(j)} \leq K, \tag{31}$$

and observe that inequality (12) can be seen as a special case of lifted inequality obtained from (31). Several different lifted inequalities could also be added to the model (see Balas 1975, Hammer et al. 1975, Padberg 1975, Wolsey 1975). This, however, would require either complex procedures for determining a number of optimal (or sufficiently good) Lagrangian multipliers or the use of an LP solver, thus increasing considerably the computational effort at each node of the branch-decision tree. A limited series of computational experiments with the branch-and-bound code of Section 3 did indeed show that the quality of the bounds is slightly improved, but the overall computing time increases.

Another possibility would be the use of a single inequality different from (12) and (23), for example a facet obtained from (31) through a sequential lifting procedure. In this case too, the increase in computational effort does not, on average, result in an upper bound improvement.

3. A BRANCH-AND-BOUND ALGORITHM

In this section we describe a depth-first branch-and-bound algorithm, MTH, which is especially tailored for hard instances of KP01, but is also effective for general instances of the problem. Each node of the branch-decision tree is generated by imposing the value zero or the value one to the current branching variable x_{j^*}. At any iteration, let Z denote the value of the best incumbent solution.

3.1. Upper Bound

The upper bound computation for the current node starts by determining, for the current instance, the maximum and minimum cardinalities K and k, and the continuous solution, i.e., the values $\bar{x}_1, \ldots, \bar{x}_s$ and CU. (Note that, given the corresponding values for the father node, all these computations can be efficiently parametrized, since only the value of a single variable has changed.) If $CU \leq Z$ or $\bar{x}_s = 0$ the node is obviously fathomed, after possible updating of the incumbent solution. Otherwise, the valid inequalities in Section 2 are checked: if (a) $k < s \leq K$ no further computation can improve the bound; otherwise, if (b) $s > K$ (resp. (c) $s \leq k$) upper bound $U1$ (resp. $U2$) is determined. When the improved bound is computed, we denote the solution corresponding to the optimal multiplier with $(\bar{x}_j)$.

For all nodes but the root, the binary search is initialized by taking for λ (resp. μ) the optimal value λ^* (resp. μ^*) found for the father node. This was experimentally found to be a good initial choice; in addition, the first iteration of the binary search requires no new sorting to determine the critical item. For the following iterations, since the position of generally only a few items changes (in the sorting according to $\bar{p}_j/w_j$ values) the new critical item is efficiently determined through *shuttle sort* techniques. In addition, as previously mentioned, the sorting is refined by breaking ties according to decreasing w_j values for the computation of $U1$, according to increasing w_j values for the computation of $U2$. Indeed, in this way we obtain a more accurate definition of $\sum_{j=1}^{s} \bar{x}_j$, hence avoiding useless computations (see (12), (23)).

Let us now consider a son node generated by the current one by imposing $x_{j^*} = \bar{x}_{j^*}$: if the same condition ((a), (b) or (c) above) holds for father and son, then we know that no new computation is needed for the son, since its upper bound value cannot change with respect to the father.

3.2. Branching Scheme

Most of the depth-first branch-and-bound algorithms presented in the literature for KP01 consider the variables according to (6) and first explore the node generated by imposing $x_{j^*} = 1$. We use the same sequence, but determine the next node to be explored according to the value of $\bar{x}_{j^*}$ obtained for the generating node: specifically, the node corresponding to $x_{j^*} = \bar{x}_{j^*}$ is explored first, since there is a high probability that this is the son node having the highest upper bound. If, however, $\bar{x}_{j^*}$ has a fractional value, the node corresponding to $x_{j^*} = 1$ is explored first (if feasible).

3.3. Dominance Criteria

We have defined dominance criteria according to the technique described in Fischetti and Toth (1988). Given the current branching item j^* and the current solution $(x_1, \ldots, x_{j^*})$, profit $p^* = \Sigma_{j=1}^{j^*} p_j x_j$ and weight $w^* = \Sigma_{j=1}^{j^*} w_j x_j$, we determine a heuristic solution of value z^h for the sub-instance defined by items $(1, \ldots, j^*)$ and capacity w^*: if $z^h > p^*$ then we know that another node of the branch-decision tree (generated by the assignments producing such a heuristic solution) dominates the current node.

The heuristic solutions have been obtained through a modified greedy algorithm and improved through local exchanges. The greedy algorithm differs from the one described in Section 1 in two points: (i) it considers the items according to a given sequence π_i, not necessarily equal to the standard sequence (6); (ii) whenever the insertion of the current item leaves a capacity less than the minimum weight of the following items, after possible updating of the current solution, the execution continues without inserting the item. The procedure is given below. We denote with m the number of items in the sub-instance, and with b the capacity; on output, the heuristic solution (x_j^h) of value z^h leaves a residual capacity b^h.

```
**procedure** GREEDY ($m$, $(\pi_i)$, $b$, $(x_j^h)$, $z^h$, $b^h$):
**input:** $m, \pi_1, \ldots, \pi_m, b$ (**comment**: sub-instance);
**output**: $x_1^h, \ldots, x_m^h, z^h, b^h$ (**comment**: heuristic solution);
**begin**
  $z^h := s := 0$;
  **for** $i := 1$ **to** $m$ **do**
     **begin**
       $j := \pi_i, x_j^h := 0$;
       **if** $w_j \leq b$ **then**
          **if** $b - w_j \geq \min\{w_{\pi_k} : k > i\}$ **then**
               $x_j^h := 1, b := b - w_j, s := s + p_j$
          **else if** $s + p_j > z^h$ **then**
                         $z^h := s + p_j, b^h := b - w_j, q := j$
     **end**;
  $x_q^h := 1$;
  **for** $i := q + 1$ **to** $m$ **do** $x_{\pi_i}^h := 0$
**end.**
```

The procedure can easily be implemented so as to require $O(m)$ time: indeed, values $\alpha_i = \min\{w_{\pi_k} : k > i\}$ $(i = 1, \ldots, m)$ can initially be computed in $O(m)$ time, once the current sequence (π_i) is given.

When the greedy solution value is not large enough to establish dominance, a further attempt is performed by trying local exchanges on the greedy solution. For each item j such that $x_j^h = 1$, we find a new solution by setting $x_j^h = 0$ and filling in a greedy way the resulting residual capacity $w_j + b^h$ with items k such that $x_k^h = 0$. The items j are scanned according to increasing weights: in this way each greedy filling can be obtained by starting from the filling of the previous iteration and adding new items. The resulting time complexity is thus $O(m)$, once the pointers σ_i producing $w_{\sigma_i} \leq w_{\sigma_{i+1}}$ are given. The procedure is given below. We denote with δ the improvement over z^h needed to fathom the node.

```
**procedure** EXCHANGE ($m$, $b^h$, $(x_j^h)$, $\delta$, $D$):
**input**: $m$(**comment**: number of items in the sub-instance);
**input**: $b^h, x_1^h, \ldots, x_m^h$ (**comment**: given heuristic solution);
**input**: $\delta$ (**comment**: target for the improvement);
**output**: $D$ (**comment**: $D$ = *true* iff the node is dominated);
**comment:** $(\sigma_i)$ is a permutation of $(1, \ldots, m)$ such that $w_{\sigma_i} \geq w_{\sigma_{i+1}}$;
**begin**
  $i := k := 1, b := b^h, s := 0, w_{m+1} := +\infty, x_{m+1} := 0$;
  $D := false$;
  **repeat**
    $j := \sigma_i$;
    **if** $x_j^h = 1$ **then**
      **begin**
        $b := b + w_j$;
        **while** $w_k \leq b$ or $x_k = 1$ **do**
          **begin**
            **if** $x_k = 0$ **then** $s := S + p_k, b := b - w_k$;
            $k := k + 1$
          **end**;
        $b := b - w_j$;
        **if** $s - p_j \geq \delta$ **then** $D := true$
      **end**;
    $i := i + 1$
  **until** $i > m$ or $D = true$
**end.**
```

We can now describe the dominance criteria applied at the decision nodes. All the heuristic solutions we determine are different from the one corresponding to the current node. Hence the dominance criterion $z^h > p^*$ can be strengthened to $z^h \geq p^*$, provided we ensure that no pair of nodes exists where each one dominates the other. This is obtained by fathoming the node, in case equality holds, only if it is produced by the condition $x_{j^*} = 1$.

If the node is obtained by imposing branching item j^*, we look for the alternative heuristic solution of value z^h by excluding item j^* and considering the previous items. Several attempts, requiring increasing computing times, are performed. We first consider the items preceding j^* and currently excluded from the solution: if j^* is directly dominated by a single item or by a subset of items obtained by GREEDY with capacity w_{j^*}, then the node is obviously dominated. Otherwise we consider all the items preceding j^*, and apply GREEDY (and, possibly, EXCHANGE) using the total available capacity w^*. Since each of these items has a profit per unit weight larger than that of j^*, the main goal is to use as many capacity units as possible. The same situation occurs for the *Subset-Sum Problem* (a KP01 in which $p_j = w_j$ for all j) for which the best heuristics (see, e.g., Martello and Toth 1990, Chapter 4) consider the items according to decreasing weight: hence this sorting is adopted for GREEDY.

If the node is obtained by excluding the branching item j^*, we look for the alternative heuristic solution of value z^h by imposing item j^* and considering the previous items with capacity $w^* - w_{j^*}$.

```
procedure DOMINANCE ($j^*$, $(x_j)$, $p^*$, $w^*$, $D$):
input: $x_1, \ldots, x_{j^*}$, $p^*$, $w^*$ and $j^*$ (comment: current solution and branching item);
output: $D$ (comment: $D = true$ iff the node is dominated);
begin
  $D := false$;
  if $x_{j^*} = 1$ (comment: the branching imposes the item) then
    if $\max\{p_j : j < j^*, x_j = 0 \text{ and } w_j \leq w_{j^*}\} \geq p_{j^*}$ then
      $D := true$
    else
      begin
        let $\pi_i$ $(i = 1, \ldots, m)$ be the $i$-th item such that
              $\pi_i < j^*$ and $x_{\pi_i} = 0$;
        GREEDY ($m$, $(\pi_i)$, $w_{j^*}$, $(x_j^h)$, $z^h$, $b^h$);
        if $z^h \geq p_{j^*}$ then $D := true$
        else (comment: consider all the items preceding $j^*$)
          begin
            let $(\tau_i)$ be a permutation of $(1, \ldots, j^* - 1)$
                  such that $w_{\tau_i} \geq w_{\tau_{i+1}}$;
            GREEDY ($j^* - 1$, $(\tau_i)$, $\omega^*$, $(x_j^h)$, $z^h$, $b^h$);
            if $z^h \geq p^*$ then $D := true$
            else EXCHANGE ($j^* - 1$, $b^h$, $(x_j^h)$, $p^* - z^h$, $D$)
          end
      end
  else (comment: the branching excludes the item) if $w_{j^*} \leq$
      $w^*$ then
    begin
      let $\tau_i = i$ $(i = 1, \ldots, j^* - 1)$;
      GREEDY ($j^* - 1$, $(\tau_i)$, $w^* - w_{j^*}$, $(x_j^h)$, $z^h$, $b^h$);
      $z^h := z^h + p_{j^*}$;
      if $z^h > p^*$ then $D := true$
      else EXCHANGE ($j^* - 1$, $b^h$, $(x_j^h)$, $p^* - z^h + 1$, $D$)
    end
end.
```

At any decision node, the sortings needed by GREEDY and EXCHANGE can be obtained in $O(j^*)$ time from the corresponding sortings relative to the previous node. Hence the overall time complexity for procedure DOMINANCE is $O(j^*)$, i.e., $O(n)$.

3.4. Partial Dynamic Programming

Preliminary computational experiments showed that for instances requiring exploration of a high number of decision nodes, most of the computing time was spent looking for a good filling of a small residual capacity using the last items in sequence (6). Specifically, values $\bar{n} < n$ and $\bar{c} < c$ exist such that, for most of the decision nodes produced by a branching item j^* such that $j^* + 1 \geq \bar{n}$, the residual capacity $c^* = c - w^*$ $(= c - \sum_{j=1}^{j^*} w_j x_j)$ is not larger than $\bar{c}$. In such situations, the descending nodes determine the optimal solution to the subproblem

$$(D(j^* + 1, c^*))\; z(j^* + 1, c^*) = \max \sum_{j=j^*+1}^{n} p_j x_j, \qquad (32)$$

$$\sum_{j=j^*+1}^{n} w_j x_j \leq c^*, \qquad (33)$$

$$x_j \in \{0, 1\} (j = j^* + 1, \ldots, n). \qquad (34)$$

The computational efficiency of algorithm MTH was then improved by solving all these subproblems once through a standard dynamic programming recursion (see, e.g., Martello and Toth 1990, Section 2.6), i.e., by computing, in $O(\bar{c}(n - j^*))$ time, $z(j^* + 1, c^*)$ for $j^* + 1 = n, n - 1, \ldots, \bar{n}$ and $c^* = 1, 2, \ldots, \bar{c}$.

During branch-and-bound, whenever $j^* + 1 \geq \bar{n}$ and $c^* \leq \bar{c}$, the dynamic programming list immediately gives the optimal value produced by the descending nodes $z^* = p^* + z(j^* + 1, c^*)$, where $p^* = \sum_{j=1}^{j^*} p_j x_j$. If $z^* > Z$ (the incumbent solution value), Z is updated; in any case, a backtracking follows.

3.5. Root Node

At the root node of the branch-decision tree of algorithm MTH several computations are performed to optimally solve the instance quickly or, if this fails, to reduce its size. Most of these computations are derived from the techniques implemented for "easy" instances of KP01 in algorithm MT2 in Martello and Toth (1990). Since for easy instances most of the computing time is taken up sorting the items according to (6), algorithm MT2 initially attempts to solve the problem by sorting only a small subset of the items. For better understanding, we give a brief outline of MT2:

STEP A. Partition set $\{1, \ldots, n\}$, without sorting the items, into three subsets $J1$, C and $J0$ such that: (a) $p_j/w_j \geq p_k/w_k \geq p_i/w_i$ $\forall\, j \in J1, k \in C, i \in J0$, (b) $\sum_{j \in J1} w_j < c < \sum_{j \in J1 \cup C} w_j$, and (c) $|C| \ll n$. An attempt is made by assuming that $x_j = 1$ for $j \in J1$ and $x_j = 0$ for $j \in J0$. Hence

STEP B. Sort the items in C according to (6) and determine the optimal solution value z of the KP01 defined by the items in C with capacity $c - \sum_{j \in J1} w_j$ (*core problem*);

this is obtained through MT1, a branch-and-bound algorithm based on continuous relaxations, which also computes an upper bound U (described in Martello and Toth 1988) on the complete instance. Let $Z = z + \sum_{j \in J1} p_j$. If $Z = U$ then the attempt is successful and the problem solved. Otherwise, continue with step C.

STEP C. Reduce the original instance (without sorting the items), i.e., determine $\overline{J1} = \{j$: if x_j is set to 0 then the upper bound value is $\leq Z\}$ and $\overline{J0} = \{j$: if x_j is set to 1 then the upper bound value is $\leq Z\}$, i.e., in any solution improving the incumbent one x_j must take the value 1 if $j \in \overline{J1}$, or the value 0 if $j \in \overline{J0}$.

STEP D. If $J1 \subseteq \overline{J1}$ and $J0 \subseteq \overline{J0}$, then we know that the core problem represented a correct reduction of the original instance; hence the solution of value Z is optimal. Otherwise, let $C = \{1, \ldots, n\} \backslash (\overline{J1} \cup \overline{J0})$, sort the items in C according to (6), and exactly solve, through algorithm MT1, the reduced problem defined by the items in C with capacity $c - \sum_{j \in \overline{J1}} w_j$.

At the root node of algorithm MTH we perform a series of computations, looking for a quick solution of the instance, but also avoiding computing time waste for the cases where such a technique appears to be hopeless. To this end we start with Steps A–C of MT2, but impose a limit on the number of iterations performed by MT1 since, for hard problems, even small-size instances can require extremely high computing times (see Table I). If the attempt is unsuccessful, we sort the items and reduce the instance. Computational experiences showed that, at this point, there are still possibilities for MT1 to solve the reduced instance, or at least improve the incumbent solution. Hence a new attempt is executed by limiting the iterations to a value depending on the expected difficulty of the instance. A good difficulty indicator is provided by the relative reduction in the number of variables we have obtained: if few variables have been fixed, the instance is generally very hard, so we give a small iteration limit to MT1 since the probability that it will be successful is low; the opposite holds if many variables have been fixed, so a large iteration limit is allowed in this case. A detailed statement of the sequence of computations follows.

STEP 1. As Step A of MT2.

STEP 2. As Step B of MT2, but MT1 is assigned a limit of 2,000 iterations for solving the core problem; let z denote the value of the solution found by MT1 (possibly not optimal), and set $Z = z + \sum_{j \in J1} p_j$. If $Z = U$ then the problem is solved. Otherwise, continue with Step 3.

STEP 3. If MT1 reached the iterations limit, go to Step 4. Otherwise (i.e., if the solution found by MT1 is optimal for the core problem), execute Step C of MT2: if $J1 \subseteq \overline{J1}$ and $J0 \subseteq \overline{J0}$ then the solution of value Z is optimal; otherwise, continue with Step 4.

STEP 4. Sort all the items according to (6) and reduce the sorted instance, i.e., determine $\overline{J1} = \{j$: if x_j is set to 0 then the upper bound value is $\leq Z\}$ and $\overline{J0} = \{j$: if x_j is set to 1 then the upper bound value is $\leq Z\}$.

STEP 5. Solve the reduced problem, defined by the items in $C = \{1, \ldots, n\} \backslash (\overline{J1} \cup \overline{J0})$ with capacity $c - \sum_{j \cup \overline{J1}} w_j$, through algorithm MT1 with a limit of Θ iterations, with $\Theta = 2{,}000$ if $|C| \geq n/5$, $\Theta = 100{,}000$ otherwise. Let z' denote the value of the solution found by MT1, and set $Z' = \max(Z, z' + \sum_{j \in \overline{J1}} p_j)$. If $Z' = U$ or MT1 did not reach the iterations limit, then the solution of value Z' is optimal. Otherwise, continue with Step 6.

STEP 6. If $Z' > Z$ use this new value to enlarge sets $\overline{J1}$ and $\overline{J0}$. In any case compute, for the reduced problem defined by the items in $\{1, \ldots, n\} \backslash (\overline{J1} \cup \overline{J0})$ with capacity $c - \sum_{j \in \overline{J1}} w_j$, the values K (Equation (11)) and k (Equation (22), using Z' for z^a). If $k < s \leq K$ start the branch-and-bound process. Otherwise, if $s > K$ (resp. $s \leq k$) compute upper bound $U1$ (resp. $U2$): if the computed bound equals Z' then the solution of value Z' is optimal, otherwise start the branch-and-bound process.

4. COMPUTATIONAL EXPERIMENTS

The branch-and-bound algorithm of the previous section, MTH, was coded in Fortran and computationally tested on the three classes of problems described in the introduction and on the following additional classes of hard problems:

Inverse strongly correlated: p_j uniformly random in $[1, a]$,
$w_j = p_j + \delta$,
$c = \frac{1}{2} \sum_{j=1}^{n} w_j$;

Almost strongly correlated: w_j uniformly random in $[1, a]$,
p_j uniformly random in $[w_j + 0.99\ \delta, w_j + 1.01\ \delta]$,
$c = \frac{1}{2} \sum_{j=1}^{n} w_j$;

Uncorrelated with similar weights: p_j uniformly random in $[1, a]$,
w_j uniformly random in $[100\ a, 100.1\ a]$,

$$c = \frac{1}{2} \sum_{j=1}^{n} w_j.$$

(Uncorrelated problems with similar weights have been suggested by Amado and Barcia 1993.) MTH was compared with the following algorithms:

MT2, MTR+DP (see the introduction);

PR: Pandit and Ravi Kumar (Fortran program provided by the authors).

The entries in Tables II and III give the average running times over 10 instances, obtained on a Digital VAXstation 3100. Each algorithm had a limit of 1,000 CPU seconds and 10 Mbytes of core memory assigned for each instance. For the cases in which not all the instances were solved, we give in brackets the number of solved instances; the average values, however, are computed over all instances. Entries "time" or "memory" indicate that the algorithm

Table II
Average Values over 10 Problems. VAXstation 3100 seconds (number of solved problems, if less than 10); $a = 1000$, $\delta = 100$, $c = \frac{1}{2} \sum_{j=1}^{n} w_j$.

Class	n	MT2	MTR+DT	PR	MTH			
		time	time	time	time	nodes	U/z	CU/z
	50	0.01	0.02		0.01	0	1.357	1.369
	100	0.01	0.02		0.02	0	1.115	1.121
Uncorrelated:	200	0.02	0.04		0.04	0	1.029	1.030
	500	0.05	0.09		0.09	0	1.007	1.007
w_j in [1, 1000],	1000	0.09	0.24		0.18	0	1.002	1.002
p_j in [1, 1000].	2000	0.15	0.44		0.26	0	1.001	1.001
	5000	0.33	1.23		0.71	0	1.000	1.000
	10000	0.50	3.73		1.10	0	1.000	1.000
	50	0.01	0.06		0.02	0	1.171	1.175
	100	0.02	0.08		0.02	0	1.049	1.050
Weakly correlated:	200	0.04	0.15		0.06	0	1.014	1.014
	500	0.08	0.49		0.14	0	1.003	1.003
w_j in [1, 1000],	1000	0.12	1.66		0.19	0	1.001	1.001
p_j in $[w_j - 100, w_j + 100]$.	2000	0.16	2.29		0.28	0	1.000	1.000
	5000	0.23	3.57		0.27	0	1.000	1.000
	10000	0.31	5.86		0.44	0	1.000	1.000
	50	0.13	0.51	0.01	0.05	0	1.011	1.198
	100	134.48 (9)	2.03	0.03	0.08	1	1.000	1.141
Strongly correlated:	200	685.59 (4)	11.98	0.09	0.39	1	1.000	1.095
	500	642.62 (4)	45.01	0.50	1.23	1	1.000	1.022
w_j in [1, 1000],	1000	941.87 (1)	124.78 (5)	1.94	1.74	1	1.000	1.017
$p_j = w_j + 100$.	2000	time	237.13 (1)	8.11	3.22	1	1.000	1.010
	5000	time	space	51.03	2.08	1	1.000	1.005
	10000	time	space	207.48	3.58	1	1.000	1.002
	50	0.77	0.31	0.01	0.06	1	1.000	1.286
	100	73.50	0.77	0.02	0.09	1	1.000	1.152
Inverse	200	506.53 (5)	1.82	0.08	0.19	1	1.000	1.089
strongly correlated:	500	time	6.30	0.48	0.68	1	1.000	1.038
p_j in [1,1000],	1000	990.49 (1)	17.73 (2)	1.90	0.67	1	1.000	1.019
$w_j = p_j + 100$.	2000	time	space	7.93	1.05	1	1.000	1.008
	5000	time	space	50.69	1.67	1	1.000	1.003
	10000	time	space	207.06	3.16	1	1.000	1.002
	50	0.09	0.51		0.07	2	1.016	1.190
	100	117.19 (9)	2.11		0.63	58	1.003	1.129
Almost	200	627.60 (4)	12.06		1.38	48	1.000	1.083
strongly correlated:	500	573.45 (5)	44.47		1.04	9	1.000	1.017
w_j in [1,1000],	1000	time	118.78 (5)		5.95	32	1.000	1.012
p_j in $[w_j + 99, w_j + 101]$.	2000	time	space		10.37	10	1.000	1.006
	5000	time	space		46.98	11	1.000	1.002
	10000	time	space		106.11	17	1.000	1.000
	50	0.11	0.02 (8)		0.06	1	1.022	1.457
	100	0.03	0.06 (5)		0.05	0	1.024	1.161
Uncorrelated with	200	110.44 (9)	0.20 (3)		0.79	12	1.009	1.121
similar weights:	500	300.18 (7)	0.50 (2)		3.34	34	1.002	1.058
w_j in [100000, 100100],	1000	200.09 (8)	1.13 (3)		0.16	0	1.000	1.016
p_j in [1,1000].	2000	301.25 (7)	space		10.35	23	1.000	1.010
	5000	652.66 (4)	space		37.53	25	1.000	1.002
	10000	595.37 (5)	space		108.61	21	1.000	1.001

solved no instance because of time limit and/or excessive core memory requirements. Algorithm PR can only be used for instances satisfying

$$p_j = \alpha\, w_j + \delta (j = 1, \ldots, n), \qquad (35)$$

so it was run only on strongly and inverse strongly correlated instances. For algorithm MTH we also give the average number of branch-decision nodes for which upper bound $U1$ or $U2$ was computed, and the upper bounds performances expressed by the average ratios $\max(U1, U2)/z$, CU/z computed at the root node. A null number of nodes indicates that the solution was found by the root-node computations at Steps 1–5.

Table III
Average Values over 10 Problems. VAXstation 3100 seconds (number of solved problems, if less than 10); $a = 10000$, $\delta = 1000$, $c = \frac{1}{2} \sum_{j=1}^{n} w_j$.

		MT2	MTR+DT	PR	MTH			
Class	n	time	time	time	time	nodes	U/z	CU/z
	50	0.01	0.03		0.01	0	1.379	1.392
	100	0.01	0.03		0.02	0	1.117	1.123
Uncorrelated:	200	0.02	0.04		0.04	0	1.027	1.029
	500	0.05	0.11		0.09	0	1.008	1.008
w_j in [1,10000],	1000	0.09	0.23		0.19	0	1.002	1.002
p_j in [1,10000].	2000	0.19	0.48		0.29	0	1.001	1.001
	5000	0.41	0.88 (8)		0.77	0	1.000	1.000
	10000	0.87	5.56 (3)		1.64	0	1.000	1.000
	50	0.02	0.06		0.03	0	1.198	1.203
	100	0.02	0.09		0.02	0	1.047	1.049
Weakly correlated:	200	0.05	0.18		0.82	18	1.016	1.017
	500	0.14	1.29 (6)		0.19	0	1.004	1.004
w_j in [1,10000],	1000	0.29	2.30 (3)		0.30	0	1.001	1.001
p_j in $[w_j - 1000, w_j + 1000]$.	2000	0.50	space		0.46	0	1.000	1.000
	5000	1.13	space		1.14	0	1.000	1.000
	10000	2.08	space		2.16	0	1.000	1.000
	50	0.15	4.55	0.04	0.79	11	1.019	1.219
	100	149.93 (9)	5.58 (5)	0.05	0.14	2	1.000	1.152
Strongly correlated:	200	776.48 (3)	21.24 (2)	0.12	0.67	3	1.000	1.104
	500	828.38 (3)	space	0.54	1.13	1	1.000	1.025
w_j in [1, 10000],	1000	time	space	1.99	4.01	1	1.000	1.016
$p_j = w_j + 1000$.	2000	time	space	7.98	14.84	1	1.000	1.007
	5000	time	space	50.67	38.92	1	1.000	1.002
	10000	time	space	205.45	54.16	1	1.000	1.001
	50	0.89	1.97	0.02	1.11	12	1.012	1.297
	100	102.36	2.29 (3)	0.03	0.12	2	1.000	1.154
Inverse	200	525.15 (5)	13.12 (1)	0.08	0.34	1	1.000	1.096
strongly correlated:	500	900.61 (1)	29.34 (1)	0.49	0.75	1	1.000	1.041
p_j in [1,10000],	1000	900.92 (1)	36.32 (1)	1.92	1.47	1	1.000	1.017
$w_j = p_j + 1000$.	2000	821.89 (2)	space	7.80	14.15	2	1.000	1.007
	5000	time	space	50.00	53.40	1	1.000	1.003
	10000	time	space	207.08	96.85	1	1.000	1.001
	50	0.12	1.83		1.39	34	1.027	1.212
	100	127.47 (9)	7.23 (5)		6.99	203	1.004	1.141
Almost	200	700.13 (4)	29.83 (1)		23.54	1248	1.001	1.094
strongly correlated:	500	735.48 (3)	space		28.61	1345	1.000	1.020
w_j in [1,10000],	1000	time	space		29.04	959	1.000	1.012
p_j in $[w_j + 990, w_j + 1010]$.	2000	time	space		30.90	1947	1.000	1.005
	5000	time	space		131.71	914	1.000	1.002
	10000	time	space		119.85	982	1.000	1.000

The instances in Table II were generated with $a = 1000$ and $\delta = 100$, and those in Table III with $a = 10000$ and $\delta = 1000$, i.e., with the input data in larger ranges. As a result, the last class of instances could not be generated for Table III as the integers involved could not be handled by the computer.

As far as strongly or inverse strongly correlated instances are concerned, PR and MTH are the only algorithms capable of effectively solving all instances. The most efficient algorithm is PR for small values of n, and MTH for large values of n.

For the remaining classes of hard problems (almost strongly correlated, uncorrelated with similar weights) MTH is clearly the only effective algorithm. It also has a very good behaviour for easy problems (uncorrelated, weakly correlated), although MT2 is slightly better for uncorrelated instances.

The running times of PR are practically independent of the range, while those of the remaining algorithms are, with few exceptions, higher in Table III.

ACKNOWLEDGMENTS

We thank S.N.N. Pandit and M. Ravi Kumar for making available the source code of their algorithm. This research was supported by Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST), Italy and by Consiglio Nazionale delle Ricerche (CNR), Italy.

REFERENCES

AMADO, L. AND P. BARCIA. 1993. Matroidal Relaxations for 0-1 Knapsack Problems. *O. R. Lett.* **14,** 147–152.

BALAS, E. 1975. Facets of the Knapsack Polytope. *Math. Programming,* **8,** 146–164.

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

CROWDER, H., E. L. JOHNSON, AND M. W. PADBERG. 1983. Solving Large-Scale Zero-One Linear Programming Problems. *Opns. Res.* **31,** 803–834.

DANTZIG, G. B. 1957. Discrete Variable Extremum Problems. *Opns. Res.* **5,** 266–277.

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

FISCHETTI, M. AND P. TOTH. 1988. A New Dominance Procedure for Combinatorial Optimization Problems. *O. R. Lett.* **7,** 181–187.

HAMMER, P. L., E. L. JOHNSON, AND U. N. PELED. 1975. Facets of Regular 0-1 Polytopes. *Math. Programming,* **8,** 179–206.

HOROWITZ, E. AND S. SAHNI. 1974. Computing Partitions with Applications to the Knapsack Problem. *J. ACM,* **21,** 277–292.

MARTELLO, S. AND P. TOTH. 1988. A New Algorithm for the 0-1 Knapsack Problem. *Mgmt. Science.* **34,** 633–644.

MARTELLO, S. AND P. TOTH. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester.

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

PADBERG, M. W. 1975. A Note on Zero-One Programming. *Opns. Res.* **23,** 833–837.

PANDIT, S.N.N. AND M. RAVI KUMAR. 1993. A Lexicographic Search for Strongly Correlated 0-1 Knapsack Problems. *Opsearch,* **30,** 97–116.

TOTH, P. 1980. Dynamic Programming Algorithms for the Zero-One Knapsack Problem. *Computing,* **25,** 29–45.

WOLSEY, L. A. 1975. Faces of Linear Inequalities in 0-1 Variables. *Math. Programming,* **8,** 165–178.