Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem

Yannick Vimont, Sylvain Boussier, Michel Vasquez

Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem

Yannick Vimont · Sylvain Boussier · Michel Vasquez
LGI2P, Parc Scientifique Georges Besse, 30035 Nimes Cedex 1, France

Abstract In a previous work we proposed a variable fixing heuristics for the 0-1 Multidimensional knapsack problem (01MDK). This approach uses fractional optima calculated in hyperplanes which contain the binary optimum. This algorithm obtained best lower bounds on the OR-LIBRARY benchmarks. Although it is very attractive in terms of results, this method does not prove the optimality of the solutions found and may fix variables to a non-optimal value. In this paper, we propose an implicit enumeration based on a reduced costs analysis which tends to fix non-basic variables to their exact values. The combination of two specific constraint propaga-tions based on reduced costs and an efficient enumeration framework enable us to fix variables on the one hand and to prune significantly the search tree on the other hand. Experimentally, our work provides two main contributions: (1) we obtain sev-eral new optimal solutions on hard instances of the OR-LIBRARY and (2) we reduce the bounds of the number of items at the optimum on several harder instances.

Keywords Multidimensional knapsack problem · Implicit enumeration · Variable fixing · Reduced costs · Constraint propagation

1 Introduction

The 0-1 multidimensional knapsack problem (01MDK) is a well-known optimization problem which belongs to the family of the NP-hard problems. It can be stated as

e-mail:

Sylvain.Boussier@ema.fr
Yannick.Vimont@ema.fr
Michel.Vasquez@ema.fr

follow:

$$01\text{MDK} \begin{cases} \text{Maximize } c \cdot x \text{ subject to,} \\ A \cdot x \leq b \text{ and } x \in \{0,1\}^n, \end{cases}$$

where $c \in \mathbb{N}^n$, $A \in \mathbb{N}^{m \times n}$ and $b \in \mathbb{N}^m$. The binary components x_j of x are decision variables: $x_j = 1$ if the item j is selected, 0 otherwise. c_j is the profit associated to the item j and A_{ij} is the cost (in terms of resource i) of the selecting item j. b_i is the amount of the resource i.

The designation comes from the original formulation which describes the problem of a hiker who has to select a subset of items from a list to take in his knapsack, so that the selected items are the most useful possible and the total weight does not exceed a given limit. It can be useful in several practical problems such as budgeting problems (investments problems), manufacturing problems (paper cutting patterns, cargo or plane loading) or telecommunication problems. The reader is referred to Fréville (2004) for more details about the different existing methods for solving this problem.

In a previous work, Vasquez and Vimont (2005) proposed a variable fixing heuristics for the 01MDK. This algorithm obtained best results on Chu and Beasley instances available on the OR-LIBRARY.[1] Although it is very attractive in terms of results, this method does not prove the optimality of the found solutions and may fix variables to a non optimal value.

Oliva et al. (2001) propose an interesting branch & bound method for solving the 01MDK. They use a reduced costs constraint based on the reduced costs of the non basic variables and the slack variables at the optimum of the relaxed $\overline{01}$MDK. Being inspired by this approach, we propose in this paper an implicit enumeration for the 01MDK which tends to fix variables to their optimal value.

At first, we will explain the main principle of our algorithm in Sect. 2 and we will detail the reduced costs based constraint we use in Sect. 3. Then we will describe the implicit enumeration framework in Sect. 4 and the reduced costs propagation embedded in Sect. 5. The computational results made on benchmarks from the OR-LIBRARY are detailed in Sect. 6.

2 General principle

In this paper, we propose an implicit enumeration which uses a reduced costs constraint to fix non-basic variables and prune nodes of the search tree. The reduced costs constraint expresses the objective function of the LP-relaxation with reduced cost coefficients and bounds this expression with any available lower bound and the optimal objective value.

Several algorithms which use the reduced costs for solving knapsack problems have been published before, the reader is referred to Saunders and Schinzinger (1970), Fayard and Plateau (1982) and Oliva et al. (2001) for articles on this subject. The originality of the proposed algorithm lies on the fact that the unpromising

[1] Actually, the instances have been generated by using the procedure proposed by Fréville and Plateau (1994).

parts of the search tree are tackled at first. Indeed, rather than keeping the branching variable to its non-basic value, we explore first the tree that corresponds to fixing it to its opposite value. Let us note that the aim of this approach is to prune the search tree as soon as possible. Hence this algorithm should be more appropriated for solving difficult instances of 01MDK than easy ones.

The method exposed here is divided in three main phases and uses several previously published results: (1) the calculation of a starting lower bound, which is carried out by the local search algorithm published in Vasquez and Vimont (2005), (2) the decomposition of the problem in several sub-problems based on the observation of Vasquez and Hao (2001) and (3) the enumeration of each sub-problem embedding the specific reduced cost propagation.

3 The reduced costs constraint

Let us consider the LP-relaxation of the 01MDK stated as follows:

$$
\begin{array}{lll}
(\overline{01}\text{MDK}) \quad \text{Max} & z_t = UB + \sum_{j \in N^-} \bar{c}_j x_j - \sum_{j \in N^+} \bar{c}_j (1 - x_j) + \sum_{j \in M} \bar{u}_i s_i \\
\text{subject to} & \bar{A}x + \bar{S}s = \bar{b}, \\
& x \in [0,1]^n, \quad s \geq 0.
\end{array}
$$

Where s is the vector of the m slack variables, $(\bar{x}, \bar{s})$ an optimal solution of the linear program $\overline{01}$MDK and UB its value, $(\bar{c}, \bar{u})$ the vector of the reduced costs corresponding to the variables (x, s) for the basic solution $(\bar{x}, \bar{s})$. $\bar{A}$, $\bar{S}$, $\bar{b}$ are values corresponding to the base associated to $(\bar{x}, \bar{s})$, so that $\bar{S}$ is the identity matrix. N^- represents the indexes of the non-basic variables, which are equal to their lower bound and N^+ the indexes of the non-basic variables which are equal to their upper bound. Let us suppose that we know a lower bound $LB \in \mathbb{N}$ of the original integer linear program 01MDK, then each current solution better than LB has to satisfy the following relation:

$$
\begin{aligned}
UB + \sum_{j \in N^-} \bar{c}_j x_j - \sum_{j \in N^+} \bar{c}_j (1 - x_j) + \sum_{i \in M} \bar{u}_i s_i \geq LB, \\
- \sum_{j \in N^-} \bar{c}_j x_j + \sum_{j \in N^+} \bar{c}_j (1 - x_j) - \sum_{j \in M} \bar{u}_i s_i \leq UB - LB.
\end{aligned}
$$

In our case, we do not take into account the slack variables. Indeed, at the optimum, the slack variables are positive and their associated reduced costs negative, thus the sum $\sum_{j \in M} \bar{u}_i s_i$ is always negative. Hence, we can withdraw the slack variables from the inequality mentioned above without losing sense. Moreover, we also know that at the optimum of the linear program, the non-basic variables with a positive reduced cost are equal to their upper bound ($\bar{c}_j > 0 \Rightarrow x_j \in N^+$) and the non-basic variables with a negative reduced cost are equal to their lower bound ($\bar{c}_j < 0 \Rightarrow x_j \in N^-$). Hence, if we consider the *absolute values* of the reduced costs,

we obtain the so called "reduced costs constraint":

$$\sum_{j \in N^-} |\bar{c}_j| x_j + \sum_{j \in N^+} |\bar{c}_j|(1 - x_j) \leq UB - LB. \tag{1}$$

4 Implicit enumeration framework

In this section, we only describe the implicit enumeration framework. The reduced costs propagation will be detailed in Sect. 5.

The first phase, which consists in calculating a starting lower bound, is detailed in Sect. 6. Once we obtain the lower bound, the second phase consists in partitioning the problem according to the observation of Vasquez and Hao (2001): they highlighted the fact that the number of items k can be easily bounded with two values $k_{\min}$ and $k_{\max}$. Calculating the $(k_{\max} - k_{\min} + 1)$ different values of k using this method, we partition the problem in 01MDK(k) problems for $k \in \{k_{\min}, \ldots, k_{\max}\}$ so that:

$$01\text{MDK}(k) \begin{cases} \text{Maximize } c \cdot x \text{ subject to,} \\ A \cdot x \leq b \text{ and } x \in \{0, 1\}^n, \\ 1 \cdot x = k \in \mathbb{N}. \end{cases}$$

The objective of the proposed algorithm is to solve each 01MDK(k) independently and to return the best value found.

For each sub-problem, the enumeration is tackled as classical branch & bound algorithms, with the main difference that the branching is done on the non-basic (i.e. integer) variables and not on the most fractional one as we usually do.

Let us consider S_t a partial solution (or node of the search tree) so that t is the depth parameter. A partial solution S_t is represented by a vector $(x_1, \ldots, x_n)$ where $x_i \in \{0, 1, *\}^n$ and $x_i = *$ if x_i is a free variable. S_t is composed of a subset $F(S_t)$ of variables fixed to zero or one and a subset $L(S_t)$ of free variables. $F(S_t)$ is partitioned in two subsets $F_0(S_t)$ and $F_1(S_t)$, containing respectively variables fixed to zero and variables fixed to one. Let $\overline{01}\text{MDK}_{S_t}$ be the linear relaxation of the sub-problem at node S_t, where decision variables are free variables and let $\bar{z}_{S_t}$ be its optimal value: the evaluation of S_t is defined as follows:

$$UB(S_t) = \sum_{j \in F_1(S_t)} c_j + \bar{z}_{S_t}. \tag{2}$$

At each node S_t of the search tree, the upper bound $UB(S_t)$ is computed. If S_t is unfeasible or if $UB(S_t)$ is lower or equal to the original lower bound LB, the enumeration backtracks. In the other case, the branching is done on the non-basic and free variable x_i at the optimum of the $\overline{01}\text{MDK}_{S_t}$ with the bigger absolute reduced cost value by fixing it to the opposite of its optimal value (the reason why we branch on the variable with the bigger absolute reduced cost value will be explained in Sect. 5.2). Once we explore the corresponding partial solution, we are able now to fix x_i to its optimal value $\bar{x}_i^{S_t}$ *and* to fix the next non-basic and free variable with the bigger absolute reduced cost value x_j to the opposite of its optimal value $(1 - \bar{x}_j^{S_t})$. Indeed, it

is useless to explore the partial solution S_{t+1} in which x_i is fixed to its optimal value since, in that case, $\overline{01}\text{MDK}_{S_{t+1}}$ provides the same optimal solution as $\overline{01}\text{MDK}_{S_t}$. The partial solution where all the non-basic variables are fixed to their optimal values and all the basic variables are free, is explored only once at each node.

For example, let us suppose that $\bar{x}^{S_t} = (0, 1, 0, 0, 0.5, 0.4)$ is the optimal solution of $\overline{01}\text{MDK}_{S_t}$ and that the variables are sorted by decreasing order of absolute reduced cost value. Then the nodes generated will be the followings:

$$(1, *, *, *, *, *)\ (0, 0, *, *, *, *)\ (0, 1, 1, *, *, *)\ (0, 1, 0, 1, *, *)\ (0, 1, 0, 0, *, *).$$

In the case where all non-basic variables are fixed, and all basic variables are free: a *Depth First Search* algorithm enumerates the remaining basic variables and if a feasible solution is met, the best known solution is updated. If the sum of the variables fixed to one is equal to k (the number of items corresponding to the explored hyperplane $1 \cdot x = k$) and the solution is feasible, the best known solution is updated else the enumeration backtracks.

5 Reduced costs propagation

In the following, we expose two reduced cost propagations embedded to the implicit enumeration framework. Both of them are based on the same reduced costs analysis but diverge in the way they are carried out. The first one takes the local reduced costs constraint into account at each node of the search tree, and the second one is a more global propagation which deals with the reduced costs constraint of all parents of each node.

5.1 Local propagation

Let $\text{gap}(S_t) = UB(S_t) - LB$ be the gap value at node S_t. The reduced costs constraint (1) allows us to fix several non-basic variables either to zero or to one. Indeed, if we insulate a variable x_j which is non-basic at the optimum of the $\overline{01}\text{MDK}_{S_t}$ and consider all others non-basic variables at their bounds, i.e. zero for the variables with a negative reduced cost and one for the variables with a positive reduced cost, the constraint (1) gives the two following relations:

$$(\bar{x}_j^{S_t} = 1) \quad \Rightarrow \quad x_j \geq \left\lceil 1 - \frac{\text{gap}(S_t)}{|\bar{c}_j^t|} \right\rceil, \tag{3}$$

$$(\bar{x}_j^{S_t} = 0) \quad \Rightarrow \quad x_j \leq \left\lceil \frac{\text{gap}(S_t)}{|\bar{c}_j^t|} \right\rceil \tag{4}$$

where $\bar{x}_j^{S_t}$ is the value of the non-basic and free variable x_j at the optimality of the $\overline{01}\text{MDK}_{S_t}$ and $\bar{c}_j^{S_t}$ its corresponding reduced cost. Using relations (3) and (4), we state the following *local reduced costs propagation*:

- *For each variable x_j which is non-basic at the optimum of the $\overline{01}MDK_{S_t}$, if $|\bar{c}_j^{S_t}| >$ gap(S_t): Fix x_j to zero (resp. one) if and only if $\bar{x}_j^{S_t}$ is equal to zero (resp. one).*

Thus, if a non-basic variable x_j has a reduced cost $\bar{c}_j^{S_t} >$ gap(S_t) at node S_t, x_j is fixed to the value $\bar{x}_j^{S_t}$.

Moreover, changing the value of a variable x_j implies that the left-hand side of constraint (1) at node S_t is augmented by the value $\bar{c}_j^{S_t}$. Then the value gap(S_t) can be interpreted as the amount of reduced cost available to swap the value of the non-basic variables at node S_t. The current partial solution S_t is consistent with constraint (1) if and only if $\sum_{j \in \Omega_{S_t}} \bar{c}_j^{S_t} \leq$ gap(S_t), so that $\Omega_{S_t} = \{j \ / \ x_j$ is fixed to $1 - \bar{x}_j^{S_t}\}$.

Supposing we branch on x_j at node S_t, then all the other variables $x_i \in L(S_t)$ so that $\bar{c}_j^{S_t} + \bar{c}_i^{S_t} >$ gap(S_t) cannot be fixed to the opposite of their optimal value. So, at each node of the search tree, we are able to proceed to this *second reduced costs propagation*:

- *Supposing we branch on $x_j \in L(S_t)$. For each non-basic and free variable $x_i \in L(S_t)$ $i \neq j$, if $\bar{c}_j^{S_t} + \bar{c}_i^{S_t} >$ gap(S_t): fix x_i to zero (resp. one) if $\bar{x}_i^{S_t}$ is equal to zero (resp. one) and let it free otherwise.*

For example, let us suppose that $\bar{x}_{S_t} = (0, 1, 0, 0, 1, 0.5, 0.4)$ at the optimality of the $\overline{01}$MDK$_{S_t}$ and that $\bar{c}_1^{S_t}, \bar{c}_2^{S_t} >$ gap(S_t), then we fix x_1 to 0 and x_2 to 1 according to the *local reduced costs propagation* and we generate the following children:

$$(\mathbf{0}, \mathbf{1}, 1, *, *, *, *, *) \quad (\mathbf{0}, \mathbf{1}, 0, 1, *, *, *) \quad (\mathbf{0}, \mathbf{1}, 0, 0, 0, *, *) \quad (\mathbf{0}, \mathbf{1}, 0, 0, 1, *, *).$$

Let us suppose now that $c_3 + c_5 >$ gap(S_t), then according to the second reduced costs propagation exposed above, x_5 will be directly fixed to its optimal value 1 after the branching on x_3 and the first children generated will be $(\mathbf{0}, \mathbf{1}, 1, *, 1, *, *)$ instead of $(\mathbf{0}, \mathbf{1}, 1, *, *, *, *, *)$. This second propagation enables us to reduce the number of simplex phases in the enumeration.

5.2 Global propagation

This propagation comes from the observation that each $\overline{01}$MDK$_{S_t}$ provides a reduced costs constraint and that each child of S_t has to satisfy this constraint. In their approach, Oliva et al. (2001) use the information brought by those constraints: at each node S_t, the corresponding reduced costs constraint is added to a constraint programming model and propagated on the variables domains. This process appears to be high c.p.u. time consuming and to penalize the whole computing time. In the proposed approach, the propagation only deals with reduced costs propagation in terms of available gap quantity as in the local propagation.

Indeed, let $P(S_t)$ be the set of parent nodes of S_t. Let us suppose that x_j is fixed to $1 - \bar{x}_j^{S_t}$ at node S_t and that $\bar{x}_j^{S_p} = \bar{x}_j^{S_t}$, $p \in P(S_t)$, then we can update gap(S_p) by withdrawing $|\bar{c}_j^{S_p}|$. If it appears that gap$(S_p) < 0$, S_t becomes unfeasible. Supposing x_j is fixed to the value v at node S_t, then for each parent node S_p $p \in P(S_t)$ so that $\bar{x}_j^{S_p} = 1 - v$, the value gap(S_p) is reduced by the corresponding reduced cost $|\bar{c}_j^{S_p}|$ and if gap$(S_p) < 0$, the current partial solution S_t is unfeasible.

Regarding the reduced cost propagation scheme, it appears clearly that the more the variable we branch on has a high absolute reduced cost value, the more the gap value is thin and the propagation is effective. Hence, at each node, we first branch on the free variables which have the higher reduced cost value in order to make the algorithm converge faster.

Algorithms 1 and 2 respectively detail the recursive enumeration procedure. Algorithm 1 is devoted to the variable fixing and the branching phase, Algorithm 2 to the creation of sub-problems and to the reduced costs propagation.

Algorithm 1 Enumeration
Data: $A, b, c, k, F_0(S_t), F_1(S_t), t$
if $k = 0$ **then** *return* solution
$\bar{z}_{S_t} =$ solve $\overline{01}\text{MDK}_{S_t}(k)$
$UB(S_t) = \sum_{j \in F_1(S_t)} c_j + \bar{z}_{S_t}$
$gap(S_t) = UB(S_t) - LB$
Sort the variables by descending order of $|\bar{c}_j^t|$
$i \leftarrow 1$
while $\bar{c}_i^{S_t} > gap(S_t)$ **do**
  $i \leftarrow i + 1$
repeat
  % Branching
  if $\bar{x}_i^{S_t} = 0$ **then** $F_1(S_t) \leftarrow x_i$
  if $\bar{x}_i^{S_t} = 1$ **then** $F_0(S_t) \leftarrow x_i$
  % Fixing
  for $j = 1$ *to* $i - 1$ **do**
    if $\bar{x}_j^{S_t} = 0$ **then** $F_0(S_t) \leftarrow x_j$
    if $\bar{x}_j^{S_t} = 1$ **then** $F_1(S_t) \leftarrow x_j$
  % Fixing by reduced costs cumulate
  for $j = i + 1$ *to* n **do**
    if $\bar{c}_i^{S_t} + \bar{c}_j^{S_t} > gap(S_t)$ **then**
      if $\bar{x}_j^{S_t} = 0$ **then** $F_0(S_t) \leftarrow x_j$
      if $\bar{x}_j^{S_t} = 1$ **then** $F_1(S_t) \leftarrow x_j$
  $auxiliary\ program(A, b, c, k, F_0(S_t), F_1(S_t), t)$
  $i \leftarrow i + 1$
until $i = n$

This method enables us to propagate the reduced costs without adding explicitly new constraints to the model at each node. We only consider one reduced cost constraint and the consistency of each sub-problem solution with it. The propagation is carried out by ascending the search tree from the current node to the root. At each node, we check that the reduced costs constraint is consistent with the fixed variables otherwise the current node is pruned.

Let us note that contrary to the classical Branch & Bound method, this propagation proscribes us to update the lower bound when a better feasible solution is met except at the root of the tree. Indeed, if the lower bound of a node of the branch and bound

tree is different from the lower bound of its parents, the global propagation will have no meaning since the gap value will change. However, since the root node does not have any parents, it is possible to update the lower bound at this stage.

Algorithm 2 Auxiliary program
Data: $A, b, c, k, F_0(S_t), F_1(S_t), t$
% Creation of the sub-problem
$\_k \leftarrow k - |F_1(S_t)|$
for $i = 0$ *to* $m - 1$ **do**
  $u = 0$
  for $j = 1$ *to* n **do**
    if $x_j \in F_1(S_t)$ **then**
      $\_b_i \leftarrow b_i - a_{ij}$
    else
      if $x_j \notin F_0(S_t)$ **then**
        $\_c_u \leftarrow c_j$
        $\_a_{iu} \leftarrow a_{ij}$
        $u \leftarrow u + 1$
% reduced costs propagation
for $j = 1$ *to* n **do**
  if $x_j \in F_0(S_t)$ **then**
    for $level = t$ *to* 0 **do**
      if $\bar{x}_j^{level} \neq 0$ **then** $gap(level) = gap(level) - |\bar{c}_j^{level}|$
      if $gap(level) < 0$ **then** *return*
  if $x_j \in F_1(S_t)$ **then**
    for $level = t$ *to* 0 **do**
      if $\bar{x}_j^{level} \neq 1$ **then** $gap(level) = gap(level) - |\bar{c}_j^{level}|$
      if $gap(level) < 0$ **then** *return*
$F_0(S_{t+1}) = F_0(S_t)$
$F_1(S_{t+1}) = F_1(S_t)$
enumeration$(\_A, \_b, \_c, \_k, F_0(S_{t+1}), F_1(S_{t+1}), t + 1)$

6 Computational experiments

Our enumeration procedure has been tested on benchmarks produced by Chu and Beasley (1998) issued from the OR-LIBRARY.[2] These benchmarks were generated using the procedure proposed by Fréville and Plateau (1994). This collection contains problems with $n = 100$, 250 and 500 variables and $m = 5$, 10 and 30 constraints. Thirty problems were generated for each n–m combination, giving a total of 270 problems. The a_{ij} integer values were generated by using a uniform random generator $U(0, 1000)$ and the right-hand side values were generated in correlation with the a_{ij} so that $b_i = \alpha \sum_{j=1}^{n} a_{ij}$ where $\alpha = 1/4, 1/2$ and $3/4$ is the tightness ratio. The costs of the objective function (c_j) were generated in correlation with the a_{ij} so that

[2] Available at http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

$c_j = \sum_{i=1}^{m} a_{ij}/m + 500d_j$ where d_j is a random generated number between 0 and 1. The whole name of each problem is **cbm.n_r**, where m is the number of constraints, n the number of variables and r the number of the instance. Each set of 30 instances with the given numbers m and n is divided into 3 series with a different tightness ratio: problems 1 to 9 correspond to the ratio $\alpha = 1/4$, problems 10 to 19 to the ratio $\alpha = 1/2$ and problems 20 to 29 to the ratio $\alpha = 3/4$. Our algorithm has been tested on a P4 3.2 GHz with 1 GB RAM and the results are compared to those produced by the commercial solver CPLEX 9.2.

6.1 Lower bound calculation

The lower bounds were calculated separately with two previously published algorithms: (1) RL^{tabu} proposed by Vasquez and Hao (2001), which is a hybrid approach combining linear programming and an efficient tabu search and (2) $Fix + RL^{\text{tabu}}$ proposed by Vasquez and Vimont (2005), which is the same algorithm embedding a variable fixing heuristics. We computed the lower bounds with RL^{tabu} for the 100 variable instances and with $Fix + RL^{\text{tabu}}$ for the 250 and 500 variable instances. The reason why we use both algorithms is that the variable fixing heuristics embedded in $Fix + RL^{\text{tabu}}$ makes the algorithm consume more c.p.u. time and that RL^{tabu} performs quite the same results for the smaller instances. It is important to note that we limited the number of iterations for both algorithms in order to have an appropriate ratio time consumption/quality of the solutions. This explains why the lower bounds exposed in this paper are lower than the ones published in Vasquez and Vimont (2005). For the 100 variable instances, RL^{tabu} is limited to 50 000 iterations and $Fix + RL^{\text{tabu}}$ is limited to 100 000 iterations for the 250 variable instances and 200 000 iterations for the 500 variable instances.

6.2 Computational results

Our enumeration algorithm found the optimum of all the 10 constraint, 250 variable instances (Table 2) even if the c.p.u. time consumption exceed 4 hours. These instances are hard to solve and we expose here new results never obtained before. CPLEX solves 46% of them in less than 4 hours of computational time and run out of memory for the others 54%. Our enumeration algorithm solves 94% of them in the same time limit. Our algorithm proves all the 500 variable, 5 constraint instances in less than 4 hours of computational time. However the optimal solutions were already published by James and Nakagawa (2005).

Our algorithm consumes little RAM memory when solving hard instances and this enables us to solve instances to the optimum even if the required computational time is greater than 4 hours (the maximum time is around 24 hours for the cb10.250_7 instance). Indeed, for the majority of the 10 constraint, 250 variable instances, CPLEX exceeds the capacity of RAM memory after 4 hours of computational time when our enumeration solves all the instances. Another positive point of our algorithm is that its structure enables to use parallel processors to reduce the overall run time required to solve a given instance. Indeed, we can assign the resolution of each 01MDK(k) with $k \in [k_{\min}, k_{\max}]$ to each processor. The parallel approach can widely decrease the run time required to solve hard instances.

Table 1 Results obtained on cb5.250 problems

Instance	lb		opt					Cplex
	z^*	t^*	z^{opt}	gap($\times 10^2$)	k^{opt}	t^{opt}	t^{total}	t^{Cpx}
cb5.250_0	59312	80	59312	0	73	0	80	**33**
cb5.250_1	61472	79	61472	0	74	3	**82**	156
cb5.250_2	62130	61	62130	0	76	1	62	**11**
cb5.250_3	59453	106	59463	0.016	71	93	**199**	861
cb5.250_4	58951	95	58951	0	74	6	**101**	261
cb5.250_5	60062	78	60077	0.024	74	28	**106**	384
cb5.250_6	60396	98	60414	0.029	76	6	**104**	171
cb5.250_7	61449	101	61472	0.037	73	36	**137**	239
cb5.250_8	61885	73	61885	0	76	3	76	**69**
cb5.250_9	58959	87	58959	0	72	0	87	**16**
cb5.250_10	109086	96	109109	0.021	132	15	**111**	124
cb5.250_11	109841	97	109841	0	135	2	99	**40**
cb5.250_12	108508	72	108508	0	130	3	**75**	129
cb5.250_13	109378	98	109383	0.004	134	12	**110**	157
cb5.250_14	110718	95	110720	0.001	130	21	**116**	671
cb5.250_15	110256	91	110256	0	132	5	**96**	227
cb5.250_16	109040	79	109040	0	133	3	**82**	195
cb5.250_17	109042	95	109042	0	132	14	**109**	314
cb5.250_18	109971	100	109971	0	130	5	**105**	165
cb5.250_19	107058	90	107058	0	131	6	96	**18**
cb5.250_20	149659	91	149665	0.004	191	5	**96**	101
cb5.250_21	155940	92	155944	0.002	190	3	95	**62**
cb5.250_22	149334	95	149334	0	191	24	**119**	141
cb5.250_23	152130	103	152130	0	193	2	105	**94**
cb5.250_24	150353	98	150353	0	191	4	102	**90**
cb5.250_25	150045	73	150045	0	191	0	73	**9**
cb5.250_26	148607	75	148607	0	192	0	75	**6**
cb5.250_27	149772	95	149782	0.006	193	8	**103**	127
cb5.250_28	155061	91	155075	0.009	190	1	92	**17**
cb5.250_29	154662	95	154668	0.003	191	4	**99**	121

Tables 1, 2 and 3 detail the results obtained on problems with 250 and 500 variables. Bold face highlights the best c.p.u. time required between our algorithm and CPLEX. The description of the columns is:

- instance: the name of the instance;
- z^*: the value of the lower bound;
- t^*: the c.p.u. time in seconds required to calculate the lower bound;
- z^{opt}: the optimal solution found by the enumeration;
- gap($\times 10^2$): the gap between the lower bound and the optimal value;
- k^{opt}: the number of items in the optimal solution;

Table 2 Results obtained on cb10.250 problems

Instance	lb		opt					Cplex
	z^*	t^*	z^{opt}	gap($\times 10^2$)	k^{opt}	t^{opt}	t^{total}	t^{Cpx}
cb10.250_0	59187	205	59187	0	68	4921	**5126**	*X*
cb10.250_1	58710	292	58781	0.001	69	43618	43910	**4369**
cb10.250_2	58094	174	58097	0.005	69	1335	**1509**	6746
cb10.250_3	60989	206	61000	0.018	70	8874	**9080**	*X*
cb10.250_4	58068	276	58092	0.041	67	14487	**14763**	*X*
cb10.250_5	58824	135	58824	0	68	964	**1099**	7394
cb10.250_6	58704	154	58704	0	67	898	**1052**	8255
cb10.250_7	58930	200	58936	0.01	69	87129	**87329**	*X*
cb10.250_8	59382	188	59387	0.008	68	4240	**4428**	*X*
cb10.250_9	59208	173	59208	0	69	6729	**6902**	*X*
cb10.250_10	110913	180	110913	0	129	4772	**4952**	*X*
cb10.250_11	108702	249	108717	0.013	127	7216	**7465**	*X*
cb10.250_12	108932	190	108932	0	128	3192	**3382**	*X*
cb10.250_13	110086	220	110086	0	131	14871	**15091**	*X*
cb10.250_14	108485	184	108485	0	128	2122	**2306**	9869
cb10.250_15	110845	184	110845	0	130	5770	**5954**	*X*
cb10.250_16	106077	263	106077	0	129	7604	**7867**	*X*
cb10.250_17	106685	216	106686	0	128	5322	**5538**	*X*
cb10.250_18	109822	251	109829	0.006	127	4566	**4817**	*X*
cb10.250_19	106723	263	106723	0	131	1121	**1384**	5716
cb10.250_20	151801	203	151809	0.005	187	770	**973**	4695
cb10.250_21	148772	146	148772	0	188	2138	**2284**	*X*
cb10.250_22	151900	175	151909	0.005	189	653	**828**	5048
cb10.250_23	151274	173	151324	0.033	189	5316	5489	**2234**
cb10.250_24	151966	179	151966	0	191	753	**932**	5727
cb10.250_25	152096	179	152109	0.008	189	639	**818**	2826
cb10.250_26	153131	154	153131	0	189	85	**239**	374
cb10.250_27	153563	210	153578	0.009	187	8259	**8469**	*X*
cb10.250_28	149130	302	149160	0.02	187	974	**1276**	1638
cb10.250_29	149704	230	149704	0	190	596	**826**	2487

- t^{opt}: the c.p.u. time in seconds required to find the optimal solution;
- t^{total}: the total time required ($t^* + t^{\text{opt}}$);
- t^{Cpx}: the c.p.u. time in seconds required to CPLEX 9.2 to find the optimal solution or X if it exceeds 4 hours of computational time;

6.3 Bounds reduction of the number of items at the optimum

For the hard instances (cb30.250, cb10.500 and cb30.500), we attempted to reduce the bounds of the number of items at the optimum by experimenting our algorithm within a limit of 10 hours of computational time. Indeed, even if we stop it prema-

Table 3 Results obtained on cb5.500 problems

Instance	lb		opt					Cplex
	z^*	t^*	z^{opt}	gap($\times 10^2$)	k^{opt}	t^{opt}	t^{total}	t^{Cpx}
cb5.500_0	120114	480	120148	0.028	147	851	**1331**	8001
cb5.500_1	117844	597	117879	0.029	148	437	1034	**855**
cb5.500_2	121105	927	121131	0.021	144	185	**1112**	4687
cb5.500_3	120785	474	120804	0.015	149	156	**630**	6050
cb5.500_4	122291	406	122319	0.022	147	152	**558**	1578
cb5.500_5	121984	603	122024	0.032	153	329	**932**	3678
cb5.500_6	119117	680	119127	0.008	145	171	**851**	6937
cb5.500_7	120551	595	120568	0.014	150	209	**804**	2672
cb5.500_8	121566	513	121586	0.016	149	659	**1172**	*X*
cb5.500_9	120663	776	120717	0.044	151	2007	**2783**	5756
cb5.500_10	218411	612	218428	0.007	267	226	**838**	1457
cb5.500_11	221118	584	221202	0.037	265	1562	2146	**905**
cb5.500_12	217523	842	217542	0.008	264	792	**1634**	3728
cb5.500_13	223534	837	223560	0.011	263	374	**1211**	5009
cb5.500_14	218966	474	218966	0	267	11	**485**	**485**
cb5.500_15	220520	679	220530	0.004	262	135	**814**	3122
cb5.500_16	219973	525	219989	0.007	266	68	**593**	1211
cb5.500_17	218194	670	218215	0.009	265	141	**811**	1096
cb5.500_18	216976	603	216976	0	262	94	**697**	2373
cb5.500_19	219689	612	219719	0.013	267	357	**969**	2522
cb5.500_20	295828	760	295828	0	383	7	767	**47**
cb5.500_21	308078	570	308086	0.002	384	104	674	**475**
cb5.500_22	299788	742	299796	0.002	385	41	783	**187**
cb5.500_23	306476	574	306480	0.001	384	46	**620**	1182
cb5.500_24	300340	543	300342	0	385	67	610	**204**
cb5.500_25	302565	607	302571	0.001	385	27	**634**	988
cb5.500_26	301327	439	301339	0.003	385	27	466	**366**
cb5.500_27	306430	496	306454	0.007	383	62	558	**148**
cb5.500_28	302809	640	302828	0.006	384	84	724	**367**
cb5.500_29	299904	425	299910	0.002	378	135	560	**478**

turely, the enumeration can provide interesting results: (1) It can fix variables to their optimal value in each explored hyperplane and (2) it can explore all the sub-tree corresponding to $1 \cdot x = k$ and prove that no solution with k number of items better than the lower bound exists. The main interest here is to reduce the bounds of the number of items at the optimum (i.e. to cut some hyperplanes). Hence, we thought about the best way to explore hyperplanes within a given time limit. Considering the fact that the function $\bar{z}(\mu) = c \cdot \bar{x}_\mu$ (optimum of $\overline{01}$MDK(μ) where $\mu \in \mathbb{R}$) is convex,[3] the

[3] This property is directly proved by applying the theorem 10.2 of the book Linear Programming by Chvátal (1983).

Table 4 New bounds for the number of items at the optimum

cb30.250				**cb10.500**				**cb30.500**			
Pb	LB	$k_{\min}$	$k_{\max}$	Pb	LB	$k_{\min}$	$k_{\max}$	Pb	LB	$k_{\min}$	$k_{\max}$
0	56824	**62**	**64**	0	117811	**134**	**136**	0	116056	128	133
1	58520	**65**	**66**	1	119232	**134**	**137**	1	114810	**126**	131
2	56553	**62**	**65**	2	119215	**135**	**137**	2	116712	**126**	**131**
3	56930	**63**	**65**	3	118813	**136**	**138**	3	115329	**126**	131
4	56629	**63**	**65**	4	116509	**133**	**138**	4	116525	**126**	**132**
5	57189	62	65	5	119504	**136**	**139**	5	115741	**129**	133
6	56328	**62**	**65**	6	119827	**138**	**139**	6	114181	**127**	**131**
7	56457	**63**	**65**	7	118329	**134**	**137**	7	114348	**126**	**131**
8	57442	62	66	8	117815	**136**	**137**	8	115419	126	132
9	56447	**63**	65	9	119231	**136**	**139**	9	117116	**126**	132
10	107770	**124**	**127**	10	217377	**256**	**259**	10	218104	**250**	254
11	108392	**124**	**126**	11	219077	**258**	**260**	11	214648	**250**	254
12	106399	**123**	**126**	12	217806	**255**	**258**	12	215978	**248**	**252**
13	106876	**124**	**126**	13	216868	**258**	**260**	13	217910	**250**	255
14	107414	**124**	**127**	14	213850	**255**	**260**	14	215689	**250**	254
15	107271	**125**	**127**	15	215086	**256**	**258**	15	215890	**251**	257
16	106365	**125**	**128**	16	217940	**259**	**261**	16	215907	**251**	255
17	104013	**124**	**127**	17	219984	**256**	**259**	17	216542	251	256
18	106835	**123**	126	18	214375	**256**	**258**	18	217340	251	256
19	105780	**125**	**127**	19	220899	**254**	**255**	19	214739	**251**	255
20	**150163**	**187**	**189**	20	304387*	**379**	**379**	20	301675	**374**	**377**
21	149958	**187**	**188**	21	302379	**380**	**380**	21	300055	**373**	379
22	153007	**187**	**188**	22	302416	**379**	**381**	22	305087	374	379
23	153234	**187**	**188**	23	300757	**379**	**380**	23	302032	**374**	**378**
24	150287	187	188	24	304374	**380**	**381**	24	304462	**375**	378
25	148574	**186**	**188**	25	301836*	**375**	**375**	25	297012	372	377
26	147477	**187**	**188**	26	304952	**377**	**379**	26	303364	372	377
27	152912	**186**	**188**	27	296478	**379**	**379**	27	307007	**375**	379
28	149570	**186**	**188**	28	301359	**379**	**380**	28	303199	374	379
29	149587	**186**	**188**	29	307089*	**378**	**378**	29	300572	**374**	378

best order to explore hyperplanes in a limited time, is to start from the possible extreme integer values of $[k_{\min}, k_{\max}]$ to the center integer values: $k_{\min}$, $k_{\max}$, $k_{\min}+1$, $k_{\max}-1$, etc. The objective is to prove that no solution $z > z^*$ with k items exists, so we first tackle the "easier" hyperplanes with the smaller value $\bar{z}(k)$.

We expose on Table 4 the new bounds for the number of items at the optimum so that $k_{\min} \leq k^{\text{opt}} \leq k_{\max}$, boldface highlights the new bounds provided comparatively to those previously known with the method exposed in Sect. 4. The description of the data per column is:

- *Pb*: the number of the instance.

- *LB*: the lower bound used. LB is in boldface if the lower bound has been improved and the star (*) indicates that LB is the optimal value.
- $k_{\min}$: the lower bound of the number of items at the optimum.
- $k_{\max}$: the upper bound of the number of items at the optimum.

We reduced the bounds of 88% of the instances within a limit of 10 hours of computational time. A hyperplane is cut if our algorithm explores the whole sub-tree of the corresponding value k and does not find any better solution than the best known solution, in the other case, if it finds a better solution than the best known solution, or if it reaches the time limit, it stops the exploration of the current hyperplane.

7 Conclusion

In this paper we have demonstrated how an implicit enumeration combining constraint and linear programming can be efficient to solve large-scale 01 multidimensional knapsack problems. Our enumeration method includes two specific reduced costs propagations which enable on the one hand to fix variables and on the other hand, to prune nodes of the search tree. Experimentally, our algorithm leads us to good results in terms of computational times. The new optimal solutions never published before and the new bounding of the number of items at the optimum for hard instances constitutes the two main contributions of our approach. Moreover the provided solutions can help researchers in the field of heuristics to evaluate their solutions. For the harder instances which are not solvable in a reasonable c.p.u time, we plan to experiment a distributed version of our algorithm.

References

Chu PC, Beasley JE (1998) A genetic algorithm for the multidimensional knapsack problem. J Heuristics 4(1):63–86

Chvátal V (1983) Linear programming. Freeman, New York

Fayard D, Plateau G (1982) An algorithm for the solution of the 0–1 knapsack problem. Computing 28 (3):269–287

Fréville A (2004) The multidimensional 0–1 knapsack problem: an overview. Eur J Oper Res 155:1–21

Fréville A, Plateau G (1994) An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. Discret Appl Math 49:189–212

James RJW, Nakagawa Y (2005) Enumeration methods for repeatedly solving multidimensional knapsack sub-problems. Technical report E88-D, 10:83-103, The Institute of Electronics, Information and Communication Engineers

Oliva C, Michelon P, Artigues C (2001) Constraint and linear programming: using reduced costs for solving the zero/one multiple knapsack problem. In International conference on constraint programming, CP 01, proceedings of the workshop on cooperative solvers in constraint programming (CoSolv 01), pp 87–98

Saunders RM, Schinzinger R (1970) A shrinking boundary algorithm for discrete system models. IEEE Trans Syst Sci Cybern 6:133–140

Vasquez M, Hao JK (2001) An hybrid approach for the 0–1 multidimensional knapsack problem. In Proceedings of the 17th international joint conference on artificial intelligence, IJCAI-01, Seattle, WA

Vasquez M, Vimont Y (2005) Improved results on the 0-1 multidimensional knapsack problem. Eur J Oper Res 165(1):70–81