A dynamic programming method with dominance technique for the knapsack sharing problem

V. Boyer[*,1], **D. El Baz**[*,2], **M. Elkihel**[*,3]

[*]CNRS; LAAS; 7 avenue du Colonel Roche, F-31077 Toulouse, France
[*]Université de Toulouse; UPS; INSA; INP; ISAE; LAAS; F-31077 Toulouse, France
[1]vboyer@laas.fr,[2]elbaz@laas.fr,[3]elkihel@laas.fr

ABSTRACT

In this paper, we propose an original method to solve exactly the knapsack sharing problem (KSP) by using a dynamic programming with dominance technique. The original problem (KSP) is decomposed in a set of knapsack problems. Our method is tested on uncorrelated and correlated instances from the literature. Computational experiences show that our method is able to find an optimal solution of large instances within reasonable computing time.
Keywords: Max-min programming, Knapsack sharing problem, Dynamic programming, Combinatorial optimization.

1. Introduction

The knapsack sharing problem (KSP) is a max-min mathematical programming problem with a knapsack constraint (see [4], [6]). The KSP is NP-hard as it can be formulated as an extension to the ordinary knapsack problem.

The KSP is composed of n items divided into m different classes. Each class $\mathcal{N}_i$ has a cardinality n_i with $\sum_{i\in\mathcal{M}} n_i = n$ and $\mathcal{M} = \{1, 2, ..., m\}$. An item $j \in \mathcal{N}_i$ is associated with:

- a decision variable $x_{ij} \in \{0, 1\}$
- a profit p_{ij}
- a weight w_{ij}

We wish to determine a subset of items to be included in the knapsack, according to its capacity C, so that the minimum profit associated with the different class is maximised. The KSP can be formulated as follows:

$$(KSP)\begin{cases} \max \min\limits_{i\in\mathcal{M}} \left\{ \sum\limits_{j\in\mathcal{N}_i} p_{ij}.x_{ij} \right\} = z(KSP) \\ subject\ to \sum\limits_{i\in\mathcal{M}} \sum\limits_{j\in\mathcal{N}_i} w_{ij}.x_{ij} \le C, \\ \qquad x_{ij} \in \{0,1\}\ for\ i \in \mathcal{M}\ and\ j \in \mathcal{N}_i. \end{cases} \tag{1.1}$$

For $i \in \mathcal{M}$ and $j \in \mathcal{N}_i$, w_{ij} , p_{ij} , and C are considered as positive integers and we assume that $\sum_{i\in\mathcal{M}} \sum_{j\in\mathcal{N}_i} w_{ij} > C$.

A common way to solve the KSP consists in its decomposition in knapsack problems (see for examples [10] and [15]). Indeed, for a class $i \in \mathcal{M}$, we define the following problem:

$$(KP_i(C_i))\begin{cases} \max \sum\limits_{j\in\mathcal{N}_i} p_{ij}.x_{ij} = z(KP_i(C_i)) \\ s.t. \sum\limits_{j\in\mathcal{N}_i} w_{ij}.x_{ij} \le C_i, \\ \qquad x_{ij} \in \{0,1\}\ \ j \in \mathcal{N}_i. \end{cases} \tag{1.2}$$

The objective is then to find a set $(C_1^*, C_2^*, ..., C_m^*)$ such that

$$\sum_{i\in\mathcal{M}} C_i^* \le C\ and$$
$$\min_{i\in\mathcal{M}}\{z(KP_i(C_i^*))\} = z(KSP),$$

where, for a problem $\mathcal{P}$, $z(\mathcal{P})$ represents its optimal value. Furthermore an upper bound and a lower bound of $z(\mathcal{P})$ will be denoted, respectively, by $\overline{z}(\mathcal{P})$ and $\underline{z}(\mathcal{P})$.

In this article, we propose a dynamic programming algorithm with dominance technique to solve the knapsack problems $(KP_i(C_i)))_{i\in\mathcal{M}}$ while trying to obtain good approximation of the values of $(C_i^*)_{i\in\mathcal{M}}$. Indeed, the algorithm will start by solving the problems $(KP_i(C_i))_{i\in\mathcal{M}}$ with, for $i \in \mathcal{M}$, $C_i \ge C_i^*$ and $\sum_{i\in\mathcal{M}} C_i^* \ge C$, and at each step of the resolution, we will try to decrease the values of $(C_i)_{i\in\mathcal{M}}$, toward $(C_i^*)_{i\in\mathcal{M}}$.

Without loss of generality, we consider in the sequel the items in a class $i \in \mathcal{M}$ sorted according to decreasing ratios $\frac{p_{ij}}{w_{ij}}$, $j \in \mathcal{N}_i$:

$$\frac{p_{i1}}{w_{i1}} \ge \frac{p_{i2}}{w_{i2}} \ge ... \ge \frac{p_{ij}}{w_{ij}} \ge ... \ge \frac{p_{in_i}}{w_{in_i}},\ i \in \mathcal{M}.$$

In the second section, we shall present the dynamic programming algorithm used to solve the problems $(KP_i)_{i\in\mathcal{M}}$. Section 3 will deal with its application to find out an optimal solution of (KSP). The methods used to evaluate the upper and the lower bounds needed during the resolution will be exposed and we will deal in particular with the computation of the capacities $(C_i^*)_{i\in\mathcal{M}}$. Finally, in section 4, the performance of the approach is evaluated through a set of problems from the literature. Some conclusions and perspectives are presented in section 5.

978-1-4244-4136-5/09/$25.00©2009IEEE

Additional notations:

Let $r \in \mathbb{R}$:

- $\lfloor r \rfloor = p$ such that $p \in \mathbb{N}$ and $p \leq r < p+1$,
- $\lceil r \rceil = p$ such that $p \in \mathbb{N}$ and $p-1 < r \leq p$,
- $|r|$ is the absolute value of r.

2. Dynamic programming

In order to solve the problems $(KP_i(C_i))_{i\in\mathcal{M}}$, we use a dynamic programming algorithm with dominance (see [1], [2], [3], [7]). The method used to compute the $(C_i)_{i\in\mathcal{M}}$ will be exposed in the next section.

In the sequel i will denote the i^{th} class of (KSP) ($i \in \mathcal{M}$).

A list $\mathcal{L}_{ik}$ is associated with each step $k \in \mathcal{N}_i$:

$$\mathcal{L}_{ik} = \left\{ (w,p) \mid w = \sum_{j=1}^{k} w_{ij}.x_{ij} \leq C_i \text{ and } p = \sum_{j=1}^{k} p_{ij}.x_{ij},\ \text{with } x_{ij} \in \{0,1\},\ j \in \{1,2,...,k\} \right\} \tag{2.1}$$

The states (w,p) in a list are sorted according to the decreasing value of p.

It follows from the dynamic programming principle that suppression of dominated states permits one to reduce drastically the size of lists $\mathcal{L}_{ik}$.

Definition: (Dominated states)
Let (w,p) be a pair of weight and profit, i.e. a state of the problem. If $\exists\ (w',p')$ such that $w' \leq w$ and $p' \geq p$, then state (w,p) is dominated by (w',p').

2.1. Computing the lists

The algorithm is initialized with the list $\mathcal{L}_{i0} = \{(0,0)\}$, and at a step $k \in \mathcal{N}_i$ the new list $\mathcal{L}_{ik}$ is obtained as follows:

$$\mathcal{L}_{ik} \quad = \quad \mathcal{L}'_{ik} - \mathcal{D}_{ik},$$

with,

$$\begin{array}{rcl} \mathcal{L}'_{ik} & = & \mathcal{L}_{i(k-1)} \oplus \{(w_{ik}, p_{ik})\} \\ & = & \mathcal{L}_{i(k-1)} \cup \{(w+w_{ik}, p+p_{ik}) \mid (w,p) \in \mathcal{L}_{i(k-1)} \\ & & \quad \text{and } w + w_{ik} \leq C_i\} \\ \mathcal{D}_{ik} & : & \text{the list of dominated states in } \mathcal{L}'_{ik} \end{array}$$

This procedure is called *NextList* and is detailled in Alg. 1.

2.2. Eliminating states via upper bounds

In order to shrink a list $\mathcal{L}_{ik}$, $k \in \mathcal{N}_i$, an upper bound associated with a state $(w,p) \in \mathcal{L}_{ik}$, $\overline{z}(w,p)$, is calculated. In this purpose, we solve exactly the following linear continuous knapsack problem (see [8]):

Alg. 1: NextList

Procedure:	NextList
Input:	A knapsack problem (KP_i) A list $\mathcal{L}_{i(k-1)}$
Output:	The next list $\mathcal{L}_{ik}$
$\mathcal{L}'_{ik} := \mathcal{L}_{i(k-1)} \oplus \{(w_{ik}, p_{ik})\}$ $\mathcal{L}_{ik} := \mathcal{L}'_{ik} - \mathcal{D}_{ik}$. where $\mathcal{D}_{ik}$ represents the list of dominated states in $\mathcal{L}'_{ik}$.	

$$(LP_i^{(w,p)}(C_i)) \left\{ \begin{array}{l} \max p + \sum_{j=k+1}^{n_i} p_{ij}.x_{ij} \\ s.t. \sum_{j=k+1}^{n_i} w_{ij}.x_{ij} \leq C_i - w, \\ \quad x_{ij} \in [0,1]\ \ j \in \{k+1,...,n_i\}. \end{array} \right. \tag{2.2}$$

And we have $\overline{z}(w,p) = \left\lfloor z(LP_i^{(w,p)}(C_i)) \right\rfloor$. Furthermore, if $\overline{z}(w,p) \leq \underline{z}(KSP)$, then the states (w,p) can be eliminated.

2.3. Fixing variables

Two methods are used to fix variables of the problem (KP_i).

First, an upper bound, $\overline{z}(KSP)$ of (KSP), is evaluated once in the beginning of the procedure. Indeed, we solve exactly the corresponding continuous KSP described bellow (see [12]):

$$(CKSP) \left\{ \begin{array}{l} \max \min_{i\in\mathcal{M}} \left\{ \sum_{j\in\mathcal{N}_i} p_{ij}.x_{ij} \right\} \\ s.t. \sum_{i\in\mathcal{M}} \sum_{j\in\mathcal{N}_i} w_{ij}.x_{ij} \leq C, \\ \quad x_{ij} \in [0,1] \text{ for } i \in \mathcal{M} \text{ and } j \in \mathcal{N}_i. \end{array} \right. \tag{2.3}$$

Then $\overline{z}(KSP) = \lfloor z(CKSP) \rfloor$.

Reducing variables 1
Let $i \in \mathcal{M}$, $k \in N_i$ and $(w,p) \in \mathcal{L}_{ik}$.

If $p > \overline{z}(KSP)$, then all free variables x_{ij}, $j \in \{k+1,...,n_i\}$, can be fixed at 0 for the state (w,p).

Indeed, as $z(KSP) \leq \overline{z}(KSP)$, when $p > \overline{z}(KSP)$ we can stop the exploration of this state because it will not give a better optimal value for (KSP).

The second method to fix variables uses information provided by the solution of $(LP_i^{(w,p)}(C_i))$ associated to a state $(w,p) \in \mathcal{L}_{ik}$, $k \in N_i$. We use the following rule to fix the free variables of a state (w,p):

Alg. 2: GreedyKSP

Procedure:	GreedyKSP
Input:	A problem (KSP)
Output:	$\underline{z}$

For i from 1 to m do
  $p_i := 0$, $w_i := 0$ and $k_i := 1$
end do

STOP:=0
while STOP=0 do
  $d := argmin\{p_1, p_2, ..., p_m\}$
  if $k_d \leq n_d$ then
    if $w_d + w_{dk_d} \leq C$, then
      x_{dk_d} is fixed to 1
      $p_d := p_d + p_{dk_d}$
      $w_d := w_d + w_{dk_d}$
    end if
    $k_d := k_d + 1$
  else
    STOP:=1
  end if
end while

$\underline{z} = \min\{p_1, p_2, ..., p_m\}$.

Reducing variables 2
Let $i \in \mathcal{M}$, $k \in N_i$ and $(w,p) \in \mathcal{L}_{ik}$.

Let d be the index of the critical variable of $(LP_i^{(w,p)}(C_i))$, i.e.:

$$\sum_{j=k+1}^{d-1} w_{ij} \leq C_i - w \text{ and } \sum_{j=k+1}^{d} w_{ij} > C_i - w$$

If for $j \in \{k+1, ...d-1, d+1, ..., n_i\}$,

$$\left\lfloor z(LP_i^{(w,p)}(C_i)) - \left| p_{ij} - w_{ij}.\frac{p_{id}}{w_{id}} \right| \right\rfloor \leq \underline{z}(KSP),$$

then x_{ij} can be fixed to 1 if $j < d$ and to 0 otherwise.

3. Solving the KSP with dynamic programming

In the above section, we can see that the efficiency of the method will depend of the evaluation of the capacities $(C_i)_{i\in\mathcal{M}}$ and of the lower bounds $\underline{z}(KSP)$. In this section, we will see how these values are calculated and how dynamic programming described below is used to find an optimal solution of (KSP).

For simplicity of the notation, $\underline{z}$ will denoted $\underline{z}(KSP)$.

3.1. The main procedure *DPKSP*

Dynamic programming is used to solve all the problems $(KP_i(C_i))_{i\in\mathcal{M}}$. A first lower bound of (KSP), $\underline{z}$, is compute with the greedy heuristics *GreedyKSP* (see Alg. 2). During the resolution, we try to improve the values of $\underline{z}$ and $(C_i)_{i\in\mathcal{M}}$ and ,in the end , the last lists $(\mathcal{L}_{in_i})_{i\in\mathcal{M}}$ is used to construct an optimal solution for (KSP).

Alg. 3: DPKSP

Procedure:	DPKSP
Input:	A problem (KSP)
Output:	$z(KSP)$

Initialisation:
$\underline{z} := \underline{z}(KSP) := GreedyKSP(KSP)$
$\overline{z}(KSP) := z^*(CKSP)$
For i from 1 to m do
  $\mathcal{L}_{i0} := \{(0,0)\}$
end for
$k := 1$
$(C_i)_{i\in\mathcal{M}} := UpdateC(\underline{z}, (\mathcal{L}_{i0})_{i\in\mathcal{M}})$

Dynamic programming:
STOP:=0
while STOP=0 do
  STOP:=1
  For i from 1 to m do
    If($k \leq n_i$)
      STOP:=0;
      $\mathcal{L}_{ik} := NextList(KP_i, \mathcal{L}_{i(k-1)})$
      For each states $(w,p) \in \mathcal{L}_{ik}$ do
        If($\overline{z}(w,p) \leq \underline{z}$ then
          $\mathcal{L}_{ik} := \mathcal{L}_{ik} - \{(w,p)\}$
        Else
          Try to fix the free variables
        end if
      end for
    end if
  end for
  $\underline{z} := UpdateZ\,(\underline{z}, (C_i)_{i\in\mathcal{M}}, (\mathcal{L}_{ik})_{i\in\mathcal{M}})$
  $(C_i)_{i\in\mathcal{M}} := UpdateC(\underline{z}, (\mathcal{L}_{ik})_{i\in\mathcal{M}})$
  $k := k + 1$
end while

Finding z^*:
$z(KSP) := FindOptimalValue\,(\underline{z}, (\mathcal{L}_{in_i})_{i\in\mathcal{M}})$

The main procedure, *DPKSP*, to solve the KSP is given in Alg. 3. The procedures *UpdateZ*, *UpdateC* and *FindOptimalValue* is detailed in the next sub-sections.

3.2. The procedure *UpdateC*

In this section, we present how the values of $(C_i)_{i\in\mathcal{M}}$ are initialized and update through *DPKSP*.

For $i \in \mathcal{M}$, and $k \in \mathcal{N}_i \cup \{0\}$, the following linear problem is associated to a state $(w,p) \in \mathcal{L}_{ik}$:

$$(minW_i((w,p),\underline{z})) \begin{cases} \min w + \sum_{j=k+1}^{n_i} w_{ij}.x_{ij} \\ s.t.\ p + \sum_{j=k+1}^{n_i} p_{ij}.x_{ij} \geq \underline{z} + 1, \\ \quad x_{ij} \in [0,1] \ \ j \in \{k+1, ..., n_i\}. \end{cases} \tag{3.1}$$

Let us define:

$$minC_i(\mathcal{L}_{ik}, \underline{z})) = \min_{(w,p)\in\mathcal{L}_{ik}} \{z(minW_i((w,p),\underline{z}))\}$$

Alg. 4: UpdateC

Procedure:	UpdateC
Input:	$\underline{z}(KSP)$ the lists $(\mathcal{L}_{ik})_{i\in\mathcal{M}}$
Output:	the capacities $(C_i)_{i\in\mathcal{M}}$

For i from 1 to m do

$$C_i := C - \sum_{i'\in\mathcal{M}-\{i\}} minC_{i'}(\mathcal{L}_{i'k}, \underline{z}(KSP)))$$

end for
(see section 3.2 for details)

.

If we want to improve the current lower bound, $\underline{z}$, then we must have, $for\ i \in \mathcal{M}$:

$$\sum_{j\in\mathcal{N}_i} w_{ij}.x_{ij} \leq C - \sum_{i'\in\mathcal{M}-\{i\}} minC_{i'}(\mathcal{L}_{i'k}, \underline{z}))$$

$$with\ x_{ij} \in \{0,1\},\ for\ j \in \mathcal{N}_i.$$

According to this, for $i \in \mathcal{M}$, C_i will be initialized with $C_i = C - \sum_{i'\in\mathcal{M}-\{i\}} minC_{i'}(\mathcal{L}_{i'0}, \underline{z}))$ and at each step k of *DPKSP*, we will try to improve the value of C_i with $C_i = C - \sum_{i'\in\mathcal{M}-\{i\}} minC_{i'}(\mathcal{L}_{i'k}, \underline{z}))$(see Alg. 4).

3.3. The procedure *UpdateZ*

Instead of updating the lower bound, $\underline{z}$, with the heuristics *GreedyKSP*, which is time consuming, we use all the lists $(\mathcal{L}_{ik})_{i\in\mathcal{M}}$ at the step k to try to improve more efficiency this bound.

Indeed, for each states in the list, a local greedy heuristics is used in order to select a particular state. The selected state of each list is combined with the others to try to improve $\underline{z}$. The details of the heuristics is given in procedure *UpdateZ* (see Alg. 5).

3.4. The procedure FindOptimalValue

In the end of the dynamic programming step, the lists $(\mathcal{L}_{in_i})_{i\in\mathcal{M}}$, with no dominated states, are acquired. In this section, we will see how these lists are combined in $\mathcal{O}\left(\max_{i\in\mathcal{M}}\{C_i\}\right)$ time to find the optimal value of (KSP).

The states (w, p) in a list have been sorted according to their decreasing value of p. Due to the dominance principle, they are also sorted according to their decreasing value of w. Thus, if we want to check if a given bound $\overline{z} \geq \underline{z}$ is feasible we have to take in each list $\mathcal{L}_{in_i}$,$i \in \mathcal{M}$, the state (w^i, p^i) which realize:

$$w^i = \min\{w \mid p \geq \overline{z},\ (w,p) \in \mathcal{L}_{in_i}\} \qquad (3.2)$$

If $\sum_{i\in\mathcal{M}} w^i \leq C$ then we found a better feasible bound for (KSP), i.e. $\overline{z}' = \min_{i\in\mathcal{M}}\{p^i\} \geq \overline{z} \geq \underline{z}$. Furthermore, all the states $(w, p) \in \mathcal{L}_{in_i}$ such that $p < p^i$ can be eliminated.

Alg. 5: UpdateZ

Procedure:	UpdateZ
Input:	$\underline{z}(KSP)$ the capacities $(C_i)_{i\in\mathcal{M}}$ the lists $(\mathcal{L}_{ik})_{i\in\mathcal{M}}$
Output:	an update value of $\underline{z}(KSP)$

For $i \in \mathcal{M}$ do $\mathcal{L}'_{ik} = \emptyset$

Greedy like step:
For i form 1 to m do
  For $(w, p) \in \mathcal{L}_{ik}$ do
    $W := w$ and $P := p$
    For j from $k+1$ to n_i do
      If $P \geq \underline{z}(KSP) + 1$ then
        exit the loop for
      end if
      If $W + w_{ij} \leq C_i$ then
        $W := W + w_{ij}$ and $P := P + p_{ij}$
      end if
    end for
    If $P \geq \underline{z}(KSP) + 1$ then
      $\mathcal{L}'_{ik} := \mathcal{L}'_{ik} \cup \{(W, P)\}$
    end if
  end for
end for

Selected states:
For i from 1 to m do
  Choose $(W_i, P_i) \in \mathcal{L}'_{ik}$ such that
  $W_i := \min_{(W,P)\in\mathcal{L}'_{ik}}\{W\}$
end for

Updating $\underline{z}(KSP)$:
If $\sum_{i\in\mathcal{M}} W_i \leq C$ then
  $\underline{z}(KSP) := \min_{i\in\mathcal{M}}\{P_i\}$
end if

Alg. 6: FindOptimalValue

Procedure:	FindOptimalValue
Input:	$\underline{z}(KSP)$ the lists $(\mathcal{L}_{in_i})_{i\in\mathcal{M}}$
Output:	the optimal value $z(KSP)$

Initialization:
For i from 1 to n do
  Let (w^i, p^i) be the first states in $\mathcal{L}_{in_i}$
end do
$\overline{z} := \min_{i\in\mathcal{M}}\{p^i\}$
$z(KSP) := \underline{z}(KSP)$

Checking feasibility:
While $\overline{z} > \underline{z}(KSP)$ do
  For i from 1 to n do
    Find $(w^i, p^i) \in \mathcal{L}_{in_i}$ such that
    $w^i := \min\{w \mid p \geq \overline{z},\ (w,p) \in \mathcal{L}_{in_i}\}$
  end for
  If $\sum_{i\in\mathcal{M}} w^i \leq C$ then
    $z(KSP) := \overline{z}$
    Exit the procedure
  Else
    For i from 1 to m do
      $\mathcal{L}_{in_i} := \mathcal{L}_{in_i} - \{(w,p) \in \mathcal{L}_{in_i} \mid p > p^i\}$
    end for
    $\overline{z} := \min_{i\in\mathcal{M}} \max_{(w,p)\in\mathcal{L}_{in_i}} \{p \mid p < \overline{z}\}$
  end if
end while

Otherwise, if $\sum_{i\in\mathcal{M}} w^i > C$, all the states $(w,p) \in \mathcal{L}_{in_i}$ such that $p > p^i$ (and so $w > w^i$) can be eliminated as they will not provide a better solution. Indeed, in this case, we have the following inequalities for the the optimal value:

$$z(KSP) < \overline{z} \leq p^i,\ i \in \mathcal{M}$$

. Therefore, we have to decrease the value of the bound $\overline{z}$ and checked if this new bound is feasible.

In the procedure *FindOptimalValue*, the state (w^i, p^i), $i \in \mathcal{M}$, is initialized with the first state in $\mathcal{L}_{in_i}$, and the first bound to check will be $\overline{z} = \min_{i\in\mathcal{M}}\{p^i\}$. Then, the states (w^i, p^i), $i \in \mathcal{M}$, are updated according to equation (3.2) and all the states in $\mathcal{L}_{in_i}$ with a higher profit p are eliminated. If $\overline{z}$ is feasible, we find the optimal bound and the procedure is stopped, otherwise, $\overline{z}$ is updated to:

$$\overline{z} = \min_{i\in\mathcal{M}} \max_{(w,p)\in\mathcal{L}_{in_i}} \{p \mid p < \overline{z}\},$$

and the procedure restart until a better feasible bound is found or $\overline{z} \leq \underline{z}$.

Thus, all the lists will be looked through only once. The algorithm of *FindOptimalValue* is given in Alg. 6.

4. Computational experiences

The procedure *DPKSP* has been writen in C and computational experiences have been carried out using an Intel Core 2 Duo T7500 (2.2 GHz).

Table 1: Uncorrelated instances

Group	n	m	Instances $1 \leq x \leq 4$
Am.x	1000	2 to 50	A02.x, A05.x, A10.x, A20.x, A30.x, A40.x, A50.x
Bm.x	2500	2 to 50	B02.x, B05.x, B10.x, B20.x, B30.x, B40.x, B50.x
Cm.x	5000	2 to 50	C02.x, C05.x, C10.x, C20.x, C30.x, C40.x, C50.x
Dm.x	7500	2 to 50	D02.x, D05.x, D10.x, D20.x, D30.x, D40.x, D50.x
Em.x	10000	2 to 50	E02.x, E05.x, E10.x, E20.x, E30.x, E40.x, E50.x
Fm.x	20000	2 to 50	F02.x, F05.x, F10.x, C20.x, F30.x, F40.x, F50.x

Table 2: Correlated instances

Group	n	m	Instances $1 \leq x \leq 4$
AmC.x	1000	2 to 10	A02C.x, A05C.x, A10C.x
BmC.x	2500	2 to 10	B02C.x, B05C.x, B10C.x
CmC.x	5000	2 to 10	C02C.x, C05C.x, C10C.x
DmC.x	7500	2 to 10	D02C.x, D05C.x, D10C.x
EmC.x	10000	2 to 10	E02C.x, E05C.x, E10C.x
FmC.x	20000	2 to 10	F02C.x, F05C.x, F10C.x

We used the set of instances of Hifi (ftp:// cermsem.univ-paris1.fr / pub / CERMSEM / hifi / KSP) which gives 168 uncorrelated instances and 72 strongly correlated instances (see [10] for further details). All the optimal values are known for these instances. They are detailed in tables 1 and 2 where each group of problems contains four instances.

The average processing time on the four instances of each group of problems is given in tables 3 and 4. These tables show that *DPKSP* is able to solve large instances (up to 20000 variables) within reasonable computing time. With the correlated instances an optimal solution is obtain in less than 13 minutes and with the uncorrelated instances the time processing is less than 1.5 minutes.

We can remark, in particular with the first instances Am.x, Bm.x, Cm.x, and Dm.x, that our method is more efficient when the number m of classes is low (less than 2 or 5) or high (over 30). This could be explained as in the first case we have only 2 or 5 knapsack problems to solve, and in the second case the capacities $(C_i)_{i\in\mathcal{M}}$ and $(n_i)_{i\in\mathcal{M}}$ decrease when m increases and the resulting knapsack problems $(KP_i(C_i))_{i\in\mathcal{M}}$ are easier to solve.

If we compare our results with those of Hifi and al. (see [10] and [11]), it seems that *PDKSP* gives better results on the set of problem A, B and C and when the number of class is low (below 5).

5. Conclusion

In this paper, we have proposed a method to solve the KSP with dynamic programming and lists. The use of this method allows us to apply dominance technique and reducing variables rules in order to improve efficiency.

Table 3: Uncorrelated instances: time processing

Inst.	t. (s.)	Inst.	t. (s.)	Inst.	t. (s.)
A02.x	0.02	C02.x	0.21	E02.x	3.02
A05.x	0.15	C05.x	1.96	E05.x	5.55
A10.x	0.14	C10.x	4.02	E10.x	15.24
A20.x	0.06	C20.x	4.55	E20.x	18.38
A30.x	0.01	C30.x	2.53	E30.x	13.27
A40.x	0.00	C40.x	0.85	E40.x	13.24
A50.x	0.01	C50.x	0.66	E50.x	15.26
B02.x	0.16	D02.x	0.66	F02.x	3.00
B05.x	0.69	D05.x	4.39	F05.x	35.16
B10.x	1.20	D10.x	6.94	F10.x	46.31
B20.x	0.69	D20.x	10.83	F20.x	68.89
B30.x	0.01	D30.x	8.22	F30.x	67.21
B40.x	0.01	D40.x	7.02	F40.x	78.46
B50.x	0.01	D50.x	4.02	F50.x	78.86

Table 4: Correlated instances: time processing

Inst.	t. (s.)	Inst.	t. (s.)	Inst.	t. (s.)
A02C.x	1.18	C02C.x	18.48	E02C.x	59.44
A05C.x	1.32	C05C.x	26.25	E05C.x	149.29
A10C.x	1.72	C10C.x	33.28	E10C.x	164.33
B02C.x	9.61	D02C.x	53.40	F02C.x	642.84
B05C.x	10.08	D05C.x	53.61	F05C.x	724.81
B10C.x	9.63	D10C.x	98.14	F10C.x	598.05

The original problem (KSP) is decomposed in a set of knapsack problems and these problems are solved via dynamic programming. Their capacities are initialized with overestimation values which are updated and decreased through out the resolution.

Computational experiences show that *DPKSP* is able to solve large instances within reasonable computing time. However, in order to decrease the processing time, it will be interesting to test different sorting methods of variables and to improve the computation of the capacities used to solve the set of knapsack problems.

References

[1] R. Bellman, "Dynamic Programming", Princeton University Press, Princeton, NJ, 1957.

[2] V. Boyer, D. El Baz & M. Elkihel, "A heuristic for the 0-1 multidimensional knapsack problem", European Journal of Operational Research, doi :10.1016/j.ejor.2007.06.068, 2008.

[3] V. Boyer, D. El Baz & M. Elkihel,"An exact cooperative method for solving the 0-1 multi- dimensional knapsack problem", MOSIM'08, Vol. 2, pp927-934, 2008.

[4] J.R. Brown, "Bounded knapsack sharing", Mathematical Programming, Vol. 67, pp343-382, 1994.

[5] J.R. Brown, "Solving knapsack sharing with general tradeoff functions", Mathematical Programming, Vol. 51, pp55-73, 1991.

[6] J.R. Brown, "The knapsack sharing", Operations Research, Vol. 27, pp341-355, 1979.

[7] D. El Baz & M. Elkihel, "Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0-1 knapsack problem", Journal of Parallel and Distributed Computing, Vol. 65, pp74-84, 2005.

[8] G.B. Dantzig, "Discrete variable extrenum problems", Operations Research, Vol. 5, pp266-277, 1957.

[9] P.C. Gilmore & R.E. Gomory, "The theory and computation of knapsack functions", Operations Research, Vol. 13, pp879-919, 1966.

[10] M. Hifi & S. Sadfi, "The Knapsack Sharing Problem: An Exact Algorithm", Journal of Combinatorial Optimization, Vol. 6, pp35-54, 2002.

[11] M. Hifi, H. M'Halla & S. Sadfi, "An exact algorithm for the knapsack sharing problem: ", Computers and Operations Research, Vol. 32, pp1311-1324, 2005.

[12] T. Kuno, H. Konno &E. Zemel, "A linear-time algorithm for solving continuous maximin knapsack problems", Operations Research Letters, Vol. 10, pp23-26, 1991.

[13] S. Martello & P. Toth, "Knapsack Problems: Algorithms and Computer Implementation", John Wiley & Sons, New York, 1990.

[14] G.L. Nemhauser, L.A. Wolsey, "Integer and Combinatorial Optimization", Wiley, New York, 1988.

[15] T. Yamada, M. Futakawa & S. Kataoka, "Some exact algorithms for the knapsack sharing problem", European Journal of Operational Research, Vol. 106, pp177-183, 1998.

[16] T. Yamada & M. Futakawa, "Heuristic and reduction algorithms for the knapsack sharing problem", Computers and Operations Research, Vol. 24, pp961-967, 1997.