

SOLUTION OF MULTIDIMENSIONAL KNAPSACK PROBLEM VIA COOPERATION OF DYNAMIC PROGRAMMING AND BRANCH AND BOUND

V. BOYER, Didier EL BAZ, Moussa ELKIHÉL

LAAS-CNRS, Université de Toulouse, 7, Avenue du Colonel Roche - 31077 Toulouse Cedex 4
vboyer@laas.fr, elbaz@laas.fr, elkihel@laas.fr

ABSTRACT: This article presents an exact cooperative method for the solution of the Multidimensional Knapsack Problem (*MKP*) which combines dynamic programming and branch and bound. Our method makes cooperate a dynamic programming heuristics based on surrogate relaxation and a branch and bound procedure. Our algorithm was tested for several randomly generated test sets and problems in the literature. Solution values of the first step are compared with optimal values and results provided by other well known existing heuristics. Then our exact cooperative method is compared with a classical branch and bound algorithm.

KEY WORDS: Multidimensional Knapsack Problems, Cooperative Method, Exact Method, Dynamic Programming, Branch and Bound, Surrogate Relaxation.

1. INTRODUCTION

The NP-hard multidimensional knapsack problem (*MKP*) arises in several practical contexts such as the capital budgeting, cargo loading, cutting stock problems and processors allocation in huge distributed systems.

The multidimensional knapsack problem can be written as follows:

$$(MKP) \left\{ \begin{array}{l} \max \sum_{j \in N} p_j x_j, \\ \text{s.t. } \sum_{j \in N} w_{i,j} x_j \leq c_i, \forall i \in M, \\ x_j \in \{0, 1\}, \forall j \in N. \end{array} \right. \quad (1)$$

n items have to be placed in the knapsack, according to its capacities (c_1, \dots, c_m) , $m \in \mathbb{N}$. To an item $j \in N = \{1, 2, \dots, n\}$, the following variables and vectors are associated:

- the decision variable $x_j \in \{0, 1\}$ ($x_j = 1$ if the item j is placed in the knapsack, and $x_j = 0$ otherwise),
- the profit $p_j \geq 0$ and
- the weights $w_{i,j} \geq 0$, $i \in M = \{1, \dots, m\}$.

In the sequel, we shall use the following notation: given a problem (P), its optimal value will be denoted by $v(P)$; $\bar{v}(P)$ and $\underline{v}(P)$ will represent, respectively, the value of an upper and a lower bound for $v(P)$.

To avoid any trivial solutions, we assume that:

- $\forall j \in N$ and $\forall i \in M$, $w_{i,j} \leq c_i$.
- $\forall i \in M$, $\sum_{j=1}^n w_{i,j} > c_i$.

A special case of (*MKP*) is the classical knapsack problem (with $m=1$); the Knapsack Problem (*KP*) has been given a lot of attention in the literature though it is not, in fact, as difficult as (*MKP*), more precisely, it can be solved in a pseudo-polynomial time (see [15] and [21]). Due to the intrinsic difficulty that is NP-hardness of (*MKP*), we have tried to transform the original (*MKP*) into a (*KP*) (see also [12] and [13]); for this purpose, we have used a relaxation technique, that is to say, surrogate relaxation.

In the sequel, we propose an efficient algorithm based on dynamic programming in order to find out a good lower bound of (*MKP*) by solving surrogate relaxation (see [5]).

The main steps of our algorithm can be presented as follows:

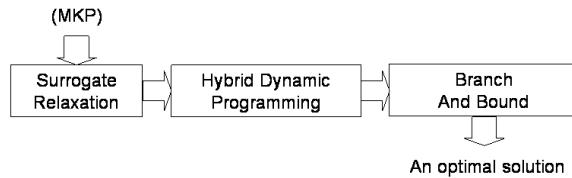


Figure 1. Computational scheme.

Section 2 deals with the construction of the surrogate constraint. In Section 3, we present the hybrid dynamic programming algorithm (HDP). In section 4 is presented the exact cooperative method. Finally, in section 5, we display and analyze some computational results obtained for different problems from the literature and randomly generated problems.

2. THE SURROGATE RELAXATION

The surrogate relaxation of (MKP) can be defined as follows:

$$(S(u)) \begin{cases} \max \sum_{j \in N} p_j \cdot x_j, \\ \text{s.t. } \sum_{i \in M} u_i \cdot \sum_{j \in N} w_{i,j} \cdot x_j \leq \sum_{i \in M} u_i \cdot c_i, \\ x_j \in \{0, 1\}, \forall j \in N, \end{cases} \quad (2)$$

where $u^T = (u_1, \dots, u_m) \geq 0$.

Since $(S(u))$ is a relaxation of (MKP) , we have $v(S(u)) \geq v(MKP)$, and the optimal multiplier vector, u^* , satisfies:

$$v(S(u^*)) = \min_{u \geq 0} \{v(S(u))\}. \quad (3)$$

Since solving (3) is a NP-hard problem, several heuristics have been proposed in order to find out good surrogate multipliers (see in particular [7], [12] and [13]). In practice, it is not important to obtain the optimal multiplier vector, since in the general case we have no guarantee that $v(S(u^*)) = v(MKP)$. A reasonable estimation can be computed by dropping the integrality restrictions in x . In other words, let

$$(LS(u)) \begin{cases} \max \sum_{j \in N} p_j \cdot x_j, \\ \text{s.t. } \sum_{i \in M} u_i \cdot \sum_{j \in N} w_{i,j} \cdot x_j \leq \sum_{i \in M} u_i \cdot c_i, \\ x_j \in [0, 1], \forall j \in N. \end{cases} \quad (4)$$

be the continuous relaxation of $(S(u))$.

The optimal continuous surrogate multipliers are derived from u^0 , such that:

$$v(LS(u^0)) = \min_{u \geq 0} v(LS(u)). \quad (5)$$

In order to compute u^0 , we consider the linear programming problem (LP) corresponding to (MKP) :

$$(LP) \begin{cases} \max \sum_{j \in N} p_j \cdot x_j, \\ \text{s.t. } \sum_{j \in N} w_{i,j} \cdot x_j \leq c_i, \forall i \in M, \\ x_j \in [0, 1], \forall j \in N. \end{cases} \quad (6)$$

We denote by $\lambda^0 = (\lambda_1^0, \lambda_2^0, \dots, \lambda_m^0) \geq 0$ the dual optimal variables associated with the constraints

$$\sum_{j \in N} w_{i,j} \cdot x_j \leq c_i, \forall i \in M. \quad (7)$$

Then, the optimal continuous surrogate multipliers can be obtained as follows using the equation (5) (see [11] p. 132).

Theorem: The optimal continuous surrogate multiplier vector satisfies $u^0 = \lambda^0$.

Then we have the following order relation (see [12], [11] p. 130 and [19]):

$$v(LP) = v(LS(u^0)) \geq v(S(u^*)) \geq v(MKP). \quad (8)$$

The reader is referred to [3], [4] and [5] for computational studies related to bounds obtained with surrogate relaxation.

3. HYBRID DYNAMIC PROGRAMMING (HDP)

For simplicity of presentation, we will denote in the sequel $\sum_{i \in M} u_i^0 \cdot w_{i,j}$ by w_j and $\sum_{i \in M} u_i^0 \cdot c_i$ by c . Then we have:

$$(S(u^0)) \begin{cases} \max \sum_{j \in N} p_j \cdot x_j, \\ \text{s.t. } \sum_{j \in N} w_j \cdot x_j \leq c, \\ x_j \in \{0, 1\}, \forall j \in N. \end{cases} \quad (9)$$

We apply the dynamic programming algorithm to

$(S(u^0))$ and store also all feasible solutions of (MKP) . At each step, $k \in N$, we update a list which is defined as follows:

$$\mathcal{L}_k = \left\{ (w, p) \mid w = \sum_{j=1}^k w_j \cdot x_j \leq c, p = \sum_{j=1}^k p_j \cdot x_j \right\} \quad (10)$$

It follows from the dynamic programming principle that the use of the concept of dominated states permits one to reduce drastically the size of lists \mathcal{L}_k since dominated states can be eliminated from the list with no last for the solution of $(S(u^0))$:

Dominated state: Let (w, p) be a couple 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 (w, p) is dominated by (w', p') .

Note that dominated states are nevertheless saved in a secondary list denoted by \mathcal{L}_{sec} since they can give rise to an optimal solution for (MKP) . The states are sorted in \mathcal{L}_{sec} according to their associated upper bound.

Let (w, p) be a state generated at stage k , we define the sub-problem associated with (w, p) by:

$$(S(u^0))_{(w,p)} \left\{ \begin{array}{l} \max \sum_{j=k+1}^n p_j \cdot x_j + p, \\ \text{s.t. } \sum_{j=k+1}^n w_j \cdot x_j \leq c - w, \\ x_j \in \{0, 1\}, j \in \{k+1, \dots, n\}. \end{array} \right. \quad (11)$$

Given a state (w, p) , an upper bound, $\bar{v}_{(w,p)}$, is obtained by solving the linear relaxation of $(S(u^0))_{(w,p)}$, i.e. $(LS(u^0))_{(w,p)}$, with the Martello And Toth algorithm (see [17]) and a lower bound, $\underline{v}_{(w,p)}$ for $(S(u^0))_{(w,p)}$, is obtained with a greedy algorithm on $(S(u^0))_{(w,p)}$.

In a list, all the states are ordered according to their decreasing upper bound. As mentioned above, our algorithm consists in applying dynamic programming (DP) to $S(u^0)$. At each stage of dynamic programming, we check the following points at the creation of a new state (w, p) :

- Is the state feasible for (MKP) (this will permit one to eliminate the unfeasible solutions)? Then, we try to improve the lower bound of (MKP) , $\underline{v}(MKP)$, with the value of p .

- Is the state dominated? In this case the state is saved in the secondary list \mathcal{L}_{sec} .

- Is the upper bound of the state (w, p) smaller than the current lower bound of $S(u^0)$? Then the state is saved too in the secondary list \mathcal{L}_{sec} .

For each state (w, p) which has not been eliminated or saved in the secondary list after these tests, we try to improve the lower bound of $(S(u^0))$, i.e. $\underline{v}(S(u^0))$, by computing a lower bound of the state with a greedy algorithm.

The dynamic programming algorithm is described below:

Dynamic Programming Algorithm (DP):

Initialisation:

$$\mathcal{L}_0 = \{(0, 0)\}, \mathcal{L}_{sec} = \emptyset$$

$\underline{v}(S(u^0)) = \underline{v}(MKP)$ (where $\underline{v}(MKP)$ is a lower bound of (MKP) given by a greedy algorithm)

Computing the lists:

For $j:=1$ to n

$$\begin{aligned} \mathcal{L}'_{j-1} &:= \{(w + w_j, p + p_j) \mid (w, p) \in \mathcal{L}_{j-1}\}; \\ &\text{Remove all states } (w, p) \in \mathcal{L}'_{j-1} \text{ which are unfeasible for } (MKP); \\ \mathcal{L}_j &:= \text{MergeLists}(\mathcal{L}_{j-1}, \mathcal{L}'_{j-1}); \end{aligned}$$

For each state $(w, p) \in \mathcal{L}_j$
 Compute $\bar{v}_{(w,p)}$ and $\underline{v}_{(w,p)}$;
 End For;

Updating the bounds:

$$\begin{aligned} p_{max} &:= \max \{p \mid (w, p) \in \mathcal{L}_j\} \text{ and} \\ v_{max} &:= \max \{\underline{v}_{(w,p)} \mid (w, p) \in \mathcal{L}_j\}; \\ \underline{v}(MKP) &:= \max \{\underline{v}(MKP), p_{max}\}; \\ \underline{v}(S(u^0)) &:= \max \{\underline{v}(S(u^0)), v_{max}\}; \end{aligned}$$

Updating \mathcal{L}_{sec} :

$$\begin{aligned} \mathcal{D}_j &:= \{(w, p) \in \mathcal{L}_j \mid (w, p) \text{ is dominated or } \bar{v}_{(w,p)} \leq \underline{v}(S(u^0))\}; \\ \mathcal{L}_{sec} &:= \mathcal{L}_{sec} \cup \mathcal{D}_j \text{ and } \mathcal{L}_j := \mathcal{L}_j - \mathcal{D}_j; \end{aligned}$$

End for.

At the end of the algorithm, we obtain a lower bound of (MKP) , i.e. $\underline{v}(MKP)$. In order to improve this lower bound and the efficiency of DP algorithm, we add to the algorithm a reducing variable process, which is given as follow:

Reducing variables rule 1: Let \underline{v} be a lower bound of (MKP) and v_j^0, v_j^1 , respectively, be the upper bounds of (MKP) with $x_j = 0, x_j = 1$, respectively. If $\underline{v} > v_j^k$ with $k = 0$ or 1 , then we can definitively fix $x_j = 1 - k$.

The upper bounds, v_j^0 and v_j^1 , $j \in N$, are obtained via the Martello and Toth algorithm on $(S(u^0))$. We use this reducing variables rule whenever we improve $\underline{v}(MKP)$ during the Dynamic Programming Phase. When a variable is fixed, we have to update all the states of the active list and to eliminate all the states which do not match the fixed variables or which become unfeasible.

We present now a procedure that allows us to improve significantly the lower bound given by DP algorithm. More precisely, we try to obtain better lower bounds for the states saved in the secondary list. Before calculating these bounds, we eliminate all the states that have become unfeasible or are incompatible with the variables that have been yet reduced or that have an upper bound smaller than the current lower bound of (MKP), i.e. $\underline{v}(MKP)$.

For a state (w, p) , let J be the index of free variables. If the state has been generated at the k -th stage of

DP Algorithm, $J = \{k + 1, \dots, n\}$, $w = \sum_{j=1}^k w_j \cdot x_j$ and
 $p = \sum_{j=1}^k p_j \cdot x_j$, then we define the new subproblem:

$$(MKP)_{(w,p)} \left\{ \begin{array}{l} \max \sum_{j \in J} p_j \cdot x_j + p, \\ \text{s.t. } \sum_{j \in J} w_{i,j} \cdot x_j \leq \bar{c}_i, \forall i \in M, \\ x_j \in \{0, 1\}, \forall j \in J, \end{array} \right. \quad (12)$$

$$\text{where } \bar{c}_i = c_i - \sum_{j=1}^k w_{i,j} \cdot x_j, \forall i \in M.$$

Two methods are used in order to evaluate the lower bound of a state using the subproblem defined above according to the reduced variables:

- a greedy algorithm;
- an enumerative method when the number $n' = n - k$ of variables of the subproblem is sufficiently small (given by the parameter α with $n' \leq \alpha$).

When all the states have been treated the process stops. The detail of the algorithm is given in what follows:

Procedure ILB:

Assign to $\underline{v}(MKP)$ the value of the lower bound returned by DP algorithm;

For each state $(w, p) \in \mathcal{L}_{sec}$
 Compute $\underline{v}_{(w,p)}$ a lower bound of $(MKP)_{(w,p)}$;
 End For;

$v_{max} := \max \{\underline{v}_{(w,p)} \mid (w, p) \in \mathcal{L}_{sec}\}$;
 $\underline{v}(MKP) := \max \{\underline{v}(MKP), v_{max}\}$.

The combination of the ILB procedure with the DP algorithm gives the so-called HDP heuristics.

4. COOPERATIVE METHOD (CM)

The goal of the cooperative method is to find out an exact solution of (MKP). As mentioned above, the secondary list \mathcal{L}_{sec} may contain an optimal solution of (MKP). We propose an algorithm based on a branch and bound method in order to explore the list \mathcal{L}_{sec} .

4.1. Principle

States are sorted according to their associated upper bound. Let (w, p) be the first state of \mathcal{L}_{sec} (the first state corresponds to the largest upper bound). An upper bound, $\bar{v}_{(w,p)}$, is obtained by solving the linear relaxation of $(MKP)_{(w,p)}$, using a simplex algorithm. A lower bound, $\underline{v}_{(w,p)}$, is obtained with a greedy algorithm on $(MKP)_{(w,p)}$.
 We propose the following branching strategy:

Branching rule: Let (w, p) be a state of the problem (MKP), J the index of the free variables (the variables that have not been already fixed by the branch and bound) and $\widetilde{X}_J = \{\widetilde{x}_j \mid j \in J\}$ an optimal solution of the linear relaxation of $(MKP)_{(w,p)}$. Then, the branching variable x_k , $k \in J$, is such that $k = \arg \min_{j \in J} \{|\widetilde{x}_j - 0.5|\}$.

Whenever we evaluate an upper bound, we use the following reducing variables rule (see [18]):

Reducing variables rule 2: Let \underline{v} be a lower bound of (MKP). Let \widetilde{v} and $\widetilde{x} = \{\widetilde{x}_j \mid j \in N\}$ be respectively the optimal value and an optimal solution of the linear relaxation of (MKP). Then we denote by $\widetilde{p} = \{\widetilde{p}_j \mid j \in N\}$, the reduce profits. For $j \in N$:

- if $\widetilde{x}_j = 0$ and $\widetilde{v} - |\widetilde{p}_j| \leq \underline{v}$ then there exists an optimal solution of (MKP) with $x_j = 0$,

- if $\tilde{x}_j = 1$ and $\tilde{v} - |\tilde{p}_j| \leq \underline{v}$ then there exists an optimal solution of (MKP) with $x_j = 1$.

This last rule permits one to reduce significantly the processing time by reducing the number of states to explore.

4.2. Details of the algorithm

The branch and bound method described above is used in order to explore the states saved in the secondary list \mathcal{L}_{sec} since this list may contain an optimal solution of (MKP) .

Procedure BB:

Let \underline{v} be the value of a lower bound of (MKP) , and \mathcal{L} a list of states.

While $\mathcal{L} \neq \emptyset$

Let (w, p) be the first state in \mathcal{L} ;

$\mathcal{L} := \mathcal{L} - \{(w, p)\}$;

Compute $\bar{v}_{(w,p)}$ an upper bound of $(MKP)_{(w,p)}$;

If $\bar{v}_{(w,p)} > \underline{v}$

Fix variables according to reducing variables rule 2 and update the state (w, p) ;

Compute $\underline{v}_{(w,p)}$ a lower bound of $(MKP)_{(w,p)}$;

If $\underline{v}_{(w,p)} > \underline{v}$, $\underline{v} := \underline{v}_{(w,p)}$ Endif;

Choose the branching variable and branch on it;

Insert the two resulting states in \mathcal{L} if they are feasible;

Endif;

Endwhile.

The cooperation of BB with HDP permits one to look for an exact solution; it corresponds to the so-called cooperative method (CM).

Procedure CM:

Step 1:

Compute \mathcal{L}_{sec} and $\underline{v}(MKP)$ using HDP heuristics.

Step 2:

Use procedure BB with $\underline{v} = \underline{v}(MKP)$ and $\mathcal{L} = \mathcal{L}_{sec}$.

The last value of \underline{v} returned by BB will be the optimal value of (MKP) .

5. COMPUTATIONAL EXPERIENCES

Our algorithm was written in C and compiled with GNU's GCC. Computational experiences were carried out using a Sun Blade 100 (500 MHz). We compare first our heuristics HDP to the following heuristics of the literature:

- AGNES of Fréville and Plateau [8];
- ADP-based heuristics approach of Bertsimas and Demir [2];
- Simple Multistage Algorithm (SMA) of Hanafi, Fréville and El Abdellaoui [14].

Our tests were made on the following problems:

- Various problems from the literature of Chu and Beasley (see [1]) composed of 9 problems with 30 instances with different sizes (100x5, 250x5, 500x5, 100x10, 250x10, 500x10, 100x30, 250x30 and 500x30), numbered respectively from 1 to 9;
- Randomly generated problems with:
 - uncorrelated data: the value of the profits and the weights are distributed independently and uniformly over [1, 1000],
 - correlated data: the value of the weights are distributed uniformly over [1, 1000] and the profits are taken as follows:

$$\sum_{k=1}^m w_{k,j} \\ \forall j \in N, p_j = \frac{\sum_{k=1}^m w_{k,j}}{m} + 100.$$

The capacity c of the knapsack is generated as follows: $\forall i \in M, c_i = 0.5 \cdot \sum_{j \in N} w_{i,j}$.

5.1. HDP heuristics

The computational results for the HDP heuristics are presented in:

- tables 1 and 2, for the problems of Chu & Beasley,
- tables 3 and 4, for randomly generated problems.

Some results for the DP heuristics are presented in tables 1 and 2.

From Tables 1 and 3, we note that the lower bound given by HDP is better than the one obtained with other methods. According to tables 2 and 4 we note that the bounds provided by HDP are obtained at the price of reasonable computational time.

| p. | size n xm | heuristics | | | | |
|----|--------------|------------|------|------|------|-------|
| | | DP | HDP | SMA | ADP | AGNES |
| 1 | 100x5 | 1.96 | 0.69 | 2.68 | 1.72 | 0.88 |
| 2 | 250x5 | 0.58 | 0.21 | 1.17 | 0.58 | 0.29 |
| 3 | 500x5 | 0.27 | 0.07 | 0.59 | 0.26 | 0.12 |
| 4 | 100x10 | 2.87 | 1.25 | 3.6 | 1.97 | 1.54 |
| 5 | 250x10 | 1.03 | 0.47 | 1.6 | 0.76 | 0.57 |
| 6 | 500x10 | 0.54 | 0.21 | 0.8 | 0.38 | 0.26 |
| 7 | 100x30 | 4.23 | 2.05 | 5.13 | 2.7 | 3.22 |
| 8 | 250x30 | 1.7 | 0.9 | 2.6 | 1.18 | 1.41 |
| 9 | 500x30 | 1.39 | 0.49 | 1.45 | 0.58 | 0.72 |

Table 1: HDP heuristics: problems of Chu and Beasley (gap to optimal solution (%)).

| Prob. | size n xm | heuristics | | | |
|-------|--------------|------------|------|------|-------|
| | | HDP | SMA | ADP | AGNES |
| UD | 50x25 | 1.81 | 5.13 | 3.31 | 4.46 |
| UD | 100x50 | 1.19 | 3.29 | 1.53 | 2.86 |
| UD | 150x75 | 0.72 | 2.15 | 1.05 | 1.90 |
| UD | 200x100 | 0.56 | 1.77 | 0.78 | 1.50 |
| UD | 250x125 | 0.52 | 1.64 | 0.71 | 1.53 |
| UD | 300x150 | 0.50 | 1.48 | 0.55 | 1.34 |
| UD | 400x200 | 0.45 | 1.20 | 0.48 | 0.94 |
| UD | 500x250 | 0.36 | 1.08 | 0.44 | 0.87 |
| CD | 50x5 | 1.75 | 6.43 | 3.48 | 4.12 |
| CD | 100x10 | 1.07 | 3.51 | 1.50 | 2.55 |
| CD | 150x15 | 1.15 | 2.28 | 1.44 | 2.85 |
| CD | 200x20 | 0.99 | 2.05 | 1.07 | 2.18 |
| CD | 250x25 | 0.97 | 1.64 | 0.98 | 1.92 |

UD: Instance with Uncorrelated Data

CD: Instance with Correlated Data

| p. | size n xm | heuristics | | | | |
|----|--------------|------------|-------|-------|------|-------|
| | | DP | HDP | SMA | ADP | AGNES |
| 1 | 100x5 | 0.03 | 0.07 | 0.15 | 0.12 | 0.10 |
| 2 | 250x5 | 0.27 | 0.52 | 1.94 | 0.24 | 0.10 |
| 3 | 500x5 | 1.50 | 2.07 | 15.63 | 1.03 | 0.34 |
| 4 | 100x10 | 0.05 | 0.12 | 0.17 | 0.15 | 0.10 |
| 5 | 250x10 | 0.45 | 0.94 | 2.39 | 0.34 | 0.10 |
| 6 | 500x10 | 2.36 | 3.81 | 19.49 | 1.47 | 0.44 |
| 7 | 100x30 | 2.49 | 5.36 | 0.39 | 0.24 | 0.10 |
| 8 | 250x30 | 22.26 | 36.66 | 4.79 | 1.27 | 0.34 |
| 9 | 500x30 | 81.31 | 88.07 | 40.80 | 4.10 | 1.03 |

Table 2: HDP heuristics: problems of Chu and Beasley (computational time (s)).

| Prob. | size n xm | heuristics | | | |
|-------|--------------|------------|---------|-------|-------|
| | | HDP | SMA | ADP | AGNES |
| UD | 50x25 | 0.03 | 0.07 | 0.10 | 0.02 |
| UD | 100x50 | 0.22 | 0.78 | 0.53 | 0.10 |
| UD | 150x75 | 0.50 | 3.90 | 1.09 | 0.30 |
| UD | 200x100 | 3.06 | 11.39 | 3.51 | 0.65 |
| UD | 250x125 | 9.77 | 25.45 | 7.35 | 1.33 |
| UD | 300x150 | 84.46 | 57.27 | 14.85 | 2.44 |
| UD | 400x200 | 227.96 | 171.74 | 41.93 | 5.91 |
| UD | 500x250 | 519.10 | 1110.55 | 80.52 | 12.42 |
| CD | 50x5 | 0.64 | 0.04 | 0.06 | 0.03 |
| CD | 100x10 | 23.26 | 0.36 | 0.41 | 0.18 |
| CD | 150x15 | 30.05 | 1.41 | 1.41 | 0.62 |
| CD | 200x20 | 42.48 | 3.53 | 3.53 | 1.46 |
| CD | 250x25 | 80.90 | 7.15 | 7.15 | 2.81 |

UD: Instance with Uncorrelated Data

CD: Instance with Correlated Data

5.2. Exact methods

In this section, we compare computational results obtained with CM with the one obtained by using the classical branch and bound method (BB). Note that if computational time exceeds 10 minutes, then

Table 4: Heuristics: Randomly generated problems (computational time (s)).

| Prob. | nxm | Gap (%) | t_BB (s) | t_CM(s) |
|-------|--------|---------|----------|---------|
| 1 | 100x5 | 0,00 | 166,60 | 160,06 |
| 2 | 250x5 | 0,0038 | 501,61 | 493,56 |
| 3 | 500x5 | -0,0146 | 600,00 | 600,00 |
| 4 | 100x10 | 0,00 | 533,95 | 600,00 |
| 5 | 250x10 | -0,0157 | 600,00 | 600,00 |
| 6 | 500x10 | -0,0525 | 600,00 | 600,00 |
| 7 | 100x30 | -0,0602 | 600,00 | 600,00 |
| 8 | 250x30 | 0,0158 | 600,00 | 600,00 |
| 9 | 500x30 | -0,0223 | 600,00 | 600,00 |

Gap: gap between CM & BB

t_BB: BB computational time

t_CM: CM computational time

Table 5: CM exact method: problems of Chu and Beasley

the methods stop and return the best value of lower bound they have obtained. In order to compare these bounds, the gaps displayed is defined as follows:

$$Gap = \frac{v_{BB} - v_{CM}}{v_{BB}}, \quad (13)$$

where v_{BB} and v_{CM} are the value of the bound delivered by BB and CM respectively. Of course, when all computational times are under 10 minutes, $v_{BB} = v_{CM} = v(MKP)$, the optimal value, and $Gap = 0$.

We present results for the problems considered previously.

Table 5 and Table 6 show that the computational times for BB and CM are quite similar. Concerning the gap, we note that it is, in most cases, negative, that is to say, when we stop the process when it exceed 10 minutes, CM delivers a better bound than BB. According to these results, CM seems to converge more rapidly toward the optimal value than BB.

6. CONCLUSION

The main advantage of the HDP heuristics is to obtain a processing time similar to the one of dynamic programming algorithm applied to a classical (KP) while having good performance in terms of gap. HDP seems to be a good heuristics since it gives better solutions than the one obtained with other heuristics with a quite good processing time.

| Prob. | nxm | Gap (%) | t_BB (s) | t_CM (s) |
|-------|---------|---------|----------|----------|
| UD | 50x25 | 0.00 | 0,35 | 0,33 |
| UD | 100x50 | 0.00 | 0,81 | 0,99 |
| UD | 150x75 | 0.00 | 328,73 | 329,60 |
| UD | 200x100 | -0.01 | 600,00 | 600,00 |
| UD | 300x150 | -0.01 | 600,00 | 600,00 |
| UD | 400x200 | 0.00 | 600,00 | 600,00 |
| UD | 500x250 | -0.01 | 600,00 | 600,00 |
| CD | 50x5 | -0.02 | 600,00 | 600,00 |
| CD | 100x10 | -0.10 | 600,00 | 600,00 |
| CD | 150x15 | -0.05 | 600,00 | 600,00 |

UD: Instance with Uncorrelated Data

CD: Instance with Correlated Data

Gap: gap between CM & BB

t_BB: BB computational time

t_CM: CM computational time

Table 6: CM exact method: randomly generated problems.

Cooperation of BB (Branch and Bound) with HDP permits one to obtain an exact method. Computing experimentations on problems from the literature shows that the combination of HDP and BB gives a processing times similar to the one of a classical branch and bound. However, this cooperative method seems to improve the convergence toward the optimal value.

HDP could be combined easily with other methods, like a Taboo search for example, in order to improve its performances to explore the neighborhood of the states saved in the secondary list. That solution could be an alternative to limit the processing time.

References

- [1] J E Beasley. Or-library: <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapsinfo.html>, 1990.
- [2] D. Bertsimas and R. Demir. An approximate dynamic-programming approach to multi-dimensional knapsack problem. *Management Science*, 4:550–565, 2002.
- [3] V. Boyer. Méthodes et/ou mixte pour la programmation linéaire en variables 0-1. DEA report, 2004. LAAS-CNRS Toulouse (France).
- [4] V. Boyer and al. An efficient heuristics for the multidimensional knapsack problem. *ROADEF'06, Presses Universitaires de Valenciennes*, pages 95–106, 2006.

- [5] V. Boyer and al. Heuristics for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 2008. doi:10.1016/j.ejor.2007.06.068.
- [6] M. Elkihel. Programmation dynamique et rotations de contraintes pour les problèmes d'optimisation entière. Thèse de Doctorat, 1984. Université des Sciences et Technologies de Lille (France).
- [7] A. Freville and G. Plateau. An exact search for the solution of the surrogate dual of the 0-1 bidimensional knapsack problem. *European Journal of Operational Research*, 68:413–421, 1993.
- [8] A. Freville and G. Plateau. An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Applied Mathematics*, 49:189–212, 1994.
- [9] A. Fréville. The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research*, 155:1–21, 2004.
- [10] M. R. Garey and D. S. Jonhson. Computer and intractability. a guide to the theory of np-completeness. ISBN 0-7167-1044-7, 1979.
- [11] S. Garfinkel and L. Nemhauser. *Integer Programming*. Wiley Interscience, 1972.
- [12] B. Gavish and H. Pirkul. Efficient algorithms for solving multiconstraint 0-1 knapsack problems to optimality. *Mathematical Programming*, 31:78–205, 1985.
- [13] F. Glover. Surrogate constraints. *Operations Research*, 16:741–749, 1968.
- [14] S. Hanafi and al. *Meta-Heuristics: Theory and Application*, chapter Comparaison of heuristics for the 0-1 multidimensional knapsack problem, pages 446–465. Kluwer Academic, 1996.
- [15] H. Kellerer and al. *Knapsack Problems*. Springer, 2004.
- [16] S. Martello and al. New trends in exact algorithms for the 0-1 knapsack problem. *European Journal of Operational Research*, 123:325–332, 2000.
- [17] S. Martello and P. Toth. *Knapsack Problems - Algorithms and Computer Implementations*. Wiley & Sons, 1990.
- [18] L. Nemhauser and A. Wolsey. *Integer and combinatorial optimization*. Wiley Interscience, 1988.
- [19] M. Osario and al. Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. *Annals of Operations Research*, 117:71–93, 2002.
- [20] G. Plateau. Contribution à la résolution des programmes mathématiques en nombres entiers. Thèse de Doctorat, 1979. Université des Sciences et Technologies de Lille.
- [21] G. Plateau and M. Elkihel. A hybrid method for the 0-1 knapsack problem. *Methods of Operations Research*, 49:277–293, 1985.
- [22] V. Poirriez and R. Andonov. Unbounded knapsack problem: new results. In *Algorithms and Experiments*, pages 103–111, 1998.
- [23] D. Sherali and J. Driscoll. Evolution and state-of-the-art in integer programing. *Journal of Computational and Applied Mathematics*, 124:319–340, 2000.