Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound

Vincent Boyer*, Didier El Baz and Moussa Elkihel

CNRS, LAAS,
7 avenue du Colonel Roche,
F-31077 Toulouse, France
and
Université de Toulouse,
UPS, INSA, INP, ISAE, LAAS,
F-31077 Toulouse, France
E-mail: vboyer@laas.fr
E-mail: elbaz@laas.fr
E-mail: elkihel@laas.fr
*Corresponding author

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.
[Received 8 October 2008; Revised 28 May 28 2009; Revised 10 December 2009; Accepted 11 December 2009]

Keywords: multidimensionnal knapsack problem; MKP; cooperative method; dynamic programming; branch and bound.

Reference to this paper should be made as follows: Boyer, V., El Baz, D. and Elkihel, M. (2010) 'Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound', *European J. Industrial Engineering*, Vol. 4, No. 4, pp.434–449.

Biographical notes: Vincent Boyer received his PhD in Automatic Systems, from the Institut National des Sciences Appliquées, in 2007 and his Engineer degree in Automatic Systems and Industrial Data Processing, from the Ecole Nationale Supérieure d'Electronique, d'Electrocinétique, d'Informatique, d'Hydraulique et de Télécommunication, in 2004. He is currently doing a postdoctoral in parallel computing at the Laboratory of Analysis and Architecture of Systems, France, in the team Distributed Computing and Asynchronism. His research interests include combinatorial optimisation and parallel computing.

Copyright © 2010 Inderscience Enterprises Ltd.

Didier El Baz received his Engineer degree in Electrical Engineering and Computer Science in 1981 by INSA Toulouse, France; he received his Docteur Ingénieur in Control Theory from INSA in 1984 and his Habilitation à Diriger des Recherches in Computer Science from the Institut National Polytechnique de Toulouse in 1998. He was a Visiting Scientist in the Laboratory for Information and Decision Systems, MIT, Cambridge, USA from 1984–1985. He is presently the Head of the Distributed Computing and Asynchronism team in LAAS-CNRS, Toulouse, France. His domain of research includes applied mathematics and parallel computing.

Moussa Elkihel received his PhD in Applied Mathematics from Université des Sciences et Techniques de Lille, France in 1984. He is a Maître de Conférence à l'IUT de Mécanique de Toulouse. His domain of research includes knapsack problems and parallel computing. He was the Co-founder with Dr. El Baz of the working group Knapsack and Optimisation of the French Operation Research Group (GDR RO).

1 Introduction

The NP-hard multidimensional knapsack problem (MKP) arises in several practical contexts such as capital budgeting, cargo loading, cutting stock problems and processors allocation in huge distributed systems (see Eilon and Christofides, 1971; Gavish and Pirkul, 1982; Lorie and Savage, 1955; Martello and Toth, 1990). The exact algorithms for the MKP are essentially based on an enumeration of the solution space. Several methods have been presented in the literature; they are generally derived from the following algorithms (see Fréville, 2004; Gavish and Pirkul, 1985; Kellerer et al., 2004; Martello et al., 2000; Nemhauser and Wolsey, 1988; Sherali and Driscoll, 2000):

- Branch and bound consists in a complete enumeration; bounds are used for fathoming nodes that cannot lead to an optimal solution. Kolesar (1967) presented the first algorithm.
- Dynamic programming can be seen as a breadth-first enumeration method with the addition of some dominance techniques. It was first introduced by Bellman (1957).

A special case of MKP is the classical knapsack problem (KP). The 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 Kellerer et al., 2004; Plateau and Elkihel, 1985). Due to the intrinsic difficulty that is NP-hardness of MKP, we have tried to transform the original MKP into a KP; for this purpose, we have used a relaxation technique, that is to say, surrogate relaxation (see Gavish and Pirkul, 1985; Glover, 1968). In the sequel, we comment on an efficient heuristics based on dynamic programming in order to find out a good lower bound of MKP by solving surrogate relaxation (see Boyer et al., 2008). In this article, we show also how this heuristics can be combined with a BB algorithm in order to construct an exact cooperative method (CM).

CMs try to combine different methods of resolution for building a more efficient one. This approach has been used with success to solve the KP. Martello and Toth (1982)

were the first to present a CM for the subset sum problems. In 1984, Plateau and Elkihel (1985) generalised this approach to the KP with general constraints. This last CM has been improved by Elkihel et al. (2002) and the computational results show a reduction by two in the processing time as compared with the dynamic programming and the branch and bound.

The paper is structured as follows. Section 2 introduces the problem formulation. Section 3 deals with the construction of the surrogate constraint. In Section 4, we present the hybrid dynamic programming (HDP). Section 5 deals with the exact CM. Finally, in Section 6, we display and analyse some computational results obtained for different problems in the literature and randomly generated problems.

2 Notations

The MKP consists in selecting a subset of items in such a way that the total profit of the selected items is maximised while a set of m knapsack constraints are satisfied. It can be written as follows:

$$MKP\begin{cases}\max \sum_{j\in N} p_j \cdot x_j, \\ \text{s.t.} \sum_{j\in N} w_{i,j} \cdot x_j \le c_i, \forall i \in M, \\ x_j \in \{0,1\},\ \forall j \in N.\end{cases} \tag{1}$$

To an item $j \in N = \{1,2,\ldots,n\}$, the following variables and vectors are associated with:

- 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 $p_j \ge 0$
- the weights $w_{i,j} \ge 0,\ i \in M = \{1,\ldots,m\}$.

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

To avoid any trivial solution, we assume that:

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

3 The surrogate relaxation

The surrogate relaxation of MKP can be defined as follows:

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

where $u^T = (u_1,\ldots,u_m) \ge 0$.

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

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

Since solving (3) is a NP-hard problem, several heuristics have been proposed in order to find out good surrogate multipliers (see, in particular, Fréville and Plateau, 1993; Gavish and Pirkul, 1985; Glover, 1968). 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:

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

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

The optimal continuous surrogate multipliers are derived from u^0, which satisfies:

$$v(LS(u^0)) = \min_{u\ge 0} v(LS(u)). \tag{5}$$

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

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

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

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

Then, the optimal continuous surrogate multipliers can be given as follows using the equation (5) [see Garfinkel and Nemhauser, (1972), p.132].

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

Then, we have the following order relation [see Gavish and Pirkul, 1985; Garfinkel and Nemhauser, (1972), p.130; Osorio et al., 2002]:

$$v(LP) = v\left(LS\left(u^0\right)\right) \geq v\left(S\left(u^*\right)\right) \geq v(MKP). \tag{8}$$

The reader is referred to Boyer (2004) and Boyer et al. (2006, 2008) for the computational studies related to bounds obtained with surrogate relaxation.

4 Hybrid dynamic programming

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:

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

We apply the dynamic programming algorithm to $S\left(u^0\right)$ and store also all feasible solutions of MKP. At each stage, $k \in N$, we update a list which is defined as follows:

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

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 loss for the solution of $S\left(u^0\right)$:

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

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

Let (w, p) be a state generated at stage k, we define the subproblem associated with (w, p) as follows (see Boyer et al., 2008):

$$S(u^0)_{(w,p)} \begin{cases} \max \sum_{j=k+1}^{n} p_j \cdot x_j + p, \\ \text{s.t.} \sum_{j=k+1}^{n} w_j \cdot x_j \le c - w, \\ x_j \in \{0, 1\},\ j \in \{k+1, \ldots, n\}. \end{cases} \tag{11}$$

Given a state (w, p), an upper bound, $\overline{v}_{(w,p)}$, 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 (1990) algorithm 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 bounds. As mentioned above, our algorithm consists in applying dynamic programming 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 not optimal for $S(u^0)$, but it can be saved in the secondary list $\mathcal{L}_{\text{sec}}$.
- Is the upper bound of the state (w, p) smaller than the current lower bound of $S(u^0)$? Then, the state is not optimal for $S(u^0)$, but it can be saved too in the secondary list $\mathcal{L}_{\text{sec}}$.

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 DP algorithm is described below:

DP algorithm:

Initialisation:

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

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

Computing the lists:

For $j := 1$ *to* n

$L'_{j-1} := \{(w + w_j, p + p_j) \mid (w, p) \in L_{j-1}\}$;

Remove all states $(w, p) \in \mathcal{L}'_{j-1}$ *which are unfeasible for* MKP;

$\mathcal{L}_j := MergeLists(\mathcal{L}_{j-1}, \mathcal{L}'_{j-1})$;

For each state $(w, p) \in \mathcal{L}_j$

Compute $\overline{v}_{(w,p)}$ *and* $\underline{v}_{(w,p)}$;

End For;

Updating the bounds:

$p_{\max} := \max\{p \mid (w, p) \in \mathcal{L}_j\}$ *and*

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

$\underline{v}(MKP) := \max\{\underline{v}(MKP), p_{\max}\}$;

$\underline{v}(S(u^0)) := \max\{\underline{v}(S(u^0)), v_{\max}\}$;

Updating $\mathcal{L}_{\text{sec}}$*:*

$\mathcal{D}_j := \{(w, p) \in \mathcal{L}_j \mid (w, p)$ *is dominant or* $\overline{v}_{(w,p)} \le \underline{v}(S(u^0))\}$;

$\mathcal{L}_{\text{sec}} := \mathcal{L}_{\text{sec}} \cup \mathcal{D}_j$ *and* $\mathcal{L}_j := \mathcal{L}_j - \mathcal{D}_j$;

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 follows:

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 DP phase. When a variable is fixed, we have to update the active list and to eliminate all the states which are 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 update the active list; eliminate all the states that are unfeasible 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 kth stage of the DP algorithm, $J = \{k+1, \ldots, n\}$, $w = \sum_{j=1}^{k} w_j \cdot x_j$ and $p = \sum_{j=1}^{k} p_j \cdot x_j$, then we define the new subproblem (see Boyer et al., 2008):

$$MKP_{(w,\,p)} \begin{cases} \max \sum_{j \in J} p_j \cdot x_j + p, \\ \text{s.t.} \sum_{j \in J} w_{i,j} \cdot x_j \le \overline{c}_i, \forall i \in M, \\ x_j \in \{0, 1\}, \ \forall j \in J, \end{cases} \tag{12}$$

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

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.

When all the states have been treated, the process stops. The algorithm is presented in details below:

Procedure ILB:

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

For each state $(w, p) \in \mathcal{L}_{\text{sec}}$

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

End For;

$v_{\max} := \max\left\{\underline{v}_{(w,\,p)} \middle| (w, p) \in \mathcal{L}_{\text{sec}}\right\}$;

$\underline{v}(MKP) := \max\left\{\underline{v}(MKP), v_{\max}\right\}$.

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

5 Cooperative method

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

5.1 Principle

States are sorted according to their associated upper bounds. Let (w, p) be the first state of $\mathcal{L}_{\text{sec}}$ (the first state corresponds to the largest upper bound). An upper bound, $\overline{v}_{(w,p)}$, 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 be the index of the free variables (the variables that have not been already fixed by the branch and bound) and $\tilde{x}_J = \{\tilde{x}_j \mid j \in J\}$ be an optimal solution of the linear relaxation of $MKP_{(w,p)}$. Then, the branching variable x_k, $k \in J$, is such that $k = \arg\min_{j \in J} \{|\tilde{x}_j - 0.5|\}$.

Whenever we evaluate an upper bound, we use the following reducing variables rule (see Nemhauser and Wolsey, 1988):

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

- if $\tilde{x}_j = 0$ and $\tilde{v} - |\tilde{p}_j| \le \underline{v}$, then there exists an optimal solution of MKP with $x_j = 0$
- if $\tilde{x}_j = 1$ and $\tilde{v} - |\tilde{p}_j| \le \underline{v}$, 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.

5.2 Details of the algorithm

The BB method described above is used in order to explore the states saved in the secondary list $\mathcal{L}_{\text{sec}}$ 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 lists of states.*

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

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

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

Compute $\overline{v}_{(w,\,p)}$ *an upper bound of* $MKP_{(w,\,p)}$;

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

Fix variables according to reducing variables rule 2;

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 CM.

Procedure CM:

Step 1:

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

Step 2:

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

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

6 Computational experiences

Our algorithm was written in C and compiled with GNU's GCC. We present computational results obtained with an Intel Pentium Dual Core E2200 (2.2 GHz). We compare first our heuristics HDP to the following heuristics of the literature:

- AGNES of Fréville and Plateau (1994)
- ADP-based heuristics approach of Bertsimas and Demir (2002)
- simple multistage algorithm (SMA) of Hanafi et al. (1996).

Our tests were made on the following problems:

- various problems from the literature of Chu and Beasley (see Beasley, 1990) composed of nine problems with 30 instances with different sizes (100 × 5, 250 × 5, 500 × 5, 100 × 10, 250 × 10, 500 × 10, 100 × 30, 250 × 30 and 500 × 30), numbered respectively from one to nine
- randomly generated problems with:
  1. uncorrelated data: the value of the profits and the weights are distributed independently and uniformly over [1, 1,000]
  2. correlated data: the value of the weights is distributed uniformly over [1, 1,000] and the profits are taken as follows:

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

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

For a fixed size $n \times m$, ten instances have been computed and tested with the different algorithms.

6.1 HDP heuristics

The computational results for the HDP heuristics are presented in:

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

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

Remark: As the optimal bounds are unknown for all the instances, the bound given by the heuristics is compared to the optimal bound of the continuous relaxation.

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 excellent bounds provided by HDP are obtained at the price of reasonable computational time.

Contrarily to other heuristics, HDP uses an enumerative method to find out a lower bound of the optimal solution. This explains why we have obtained good approximations with HDP within a relative high processing time on correlated instances. However, compared to SMA, which is build on a Taboo search, our method is more efficient in terms of approximation and processing time in the general case.

Table 1 HDP heuristics: problems of Chu and Beasley [gap to optimal value (%)]

P.	*Size* $n \times m$	*Heuristics*				
		DP	*HDP*	*SMA*	*ADP*	*AGNES*
1	100 × 5	1.96	0.69	2.68	1.72	0.88
2	250 × 5	0.58	0.21	1.17	0.58	0.29
3	500 × 5	0.27	0.07	0.59	0.26	0.12
4	100 × 10	2.87	1.25	3.60	1.97	1.54
5	250 × 10	1.03	0.47	1.60	0.76	0.57
6	500 × 10	0.54	0.21	0.80	0.38	0.26
7	100 × 30	4.23	2.05	5.13	2.7	3.22
8	250 × 30	1.70	0.90	2.60	1.18	1.41
9	500 × 30	1.39	0.49	1.45	0.58	0.72

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

P.	*Size* $n \times m$	*Heuristics*				
		DP	*HDP*	*SMA*	*ADP*	*AGNES*
1	100 × 5	0.03	0.03	0.03	0.03	0.03
2	250 × 5	0.07	0.10	0.43	0.03	0.03
3	500 × 5	0.23	0.37	3.30	0.17	0.07
4	100 × 10	0.00	0.03	0.07	0.03	0.03
5	250 × 10	0.10	0.13	0.53	0.07	0.03
6	500 × 10	0.43	0.63	4.23	0.27	0.10
7	100 × 30	0.87	1.67	0.07	0.07	0.03
8	250 × 30	8.57	9.87	1.10	0.20	0.07
9	500 × 30	21.46	26.37	8.97	0.67	0.20

Table 3 Heuristics: randomly generated problems [gap to optimal value (%)]

P.	*Size* $n \times m$	*Heuristics*			
		HDP	*SMA*	*ADP*	*AGNES*
UD	50 × 25	1.81	5.13	3.31	4.46
UD	100 × 50	1.19	3.29	1.53	2.86
UD	150 × 75	0.72	2.15	1.05	1.90
UD	200 × 100	0.56	1.77	0.78	1.50
UD	250 × 125	0.52	1.64	0.71	1.53
UD	300 × 150	0.50	1.48	0.55	1.34
UD	400 × 200	0.45	1.20	0.48	0.94
UD	500 × 250	0.36	1.08	0.44	0.87
CD	50 × 5	1.75	6.43	3.48	4.12
CD	100 × 10	1.07	3.51	1.50	2.55
CD	150 × 15	1.15	2.28	1.44	2.85
CD	200 × 20	0.99	2.05	1.07	2.18
CD	250 × 25	0.97	1.64	0.98	1.92

Notes: UD – instance with uncorrelated data and CD – instance with correlated data

Table 4 Heuristics: randomly generated problems [computational time (s)]

P.	*Size* $n \times m$	*Heuristics*			
		HDP	*SMA*	*ADP*	*AGNES*
UD	50 × 25	0.00	0.00	0.00	0.00
UD	100 × 50	0.00	0.03	0.00	0.00
UD	150 × 75	0.00	0.13	0.00	0.00
UD	200 × 100	0.50	0.43	0.10	0.03
UD	250 × 125	1.20	1.10	0.20	0.07
UD	300 × 150	6.10	2.17	0.28	0.09
UD	400 × 200	24.10	6.40	0.67	0.22
UD	500 × 250	29.80	41.09	1.42	0.47
CD	50 × 5	0.07	0.00	0.00	0.00
CD	100 × 10	2.79	0.00	0.00	0.00
CD	150 × 15	10.98	0.10	0.00	0.00
CD	200 × 20	22.95	0.40	0.00	0.00
CD	250 × 25	43.68	1.00	0.10	0.03

Notes: UD – instance with uncorrelated data and CD – instance with correlated data

6.2 Exact methods

In this section, we compare computational results obtained with CM with the one obtained by using the classical BB method. Note that if computational time exceeds 60 minutes, then the methods stop and return the best value of lower bound they have obtained. In order to compare these bounds, the gaps displayed are defined as follows:

$$Gap = \frac{v_{BB} - v_{CM}}{v_{BB}}, \tag{13}$$

where v_{BB} and v_{CM} are the values of the bound delivered by BB and CM, respectively. Of course, when all computational times are under 60 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 the process is stopped after 60 minutes, CM delivers a better bound than the classical BB method. According to these results, CM seems to converge more rapidly towards the optimal value than BB.

Table 5 CM exact method: problems of Chu and Beasley

P.	$n \times m$	*Gap* ($^0/_{00}$)	*t_BB (s)*	*t_CM (s)*
1	100 × 5	0.000	19.40	14.53
2	250 × 5	0.000	2,243.40	2,125.58
3	500 × 5	–0.014	3,600.00	3,600.00
4	100 × 10	0.000	446.43	446.03
5	250 × 10	0.000	3,600.00	3,600.00
6	500 × 10	–0.015	3,600.00	3,600.00
7	100 × 30	0.000	3,024.83	2,877.40
8	250 × 30	–0.013	3,600.00	1,800.00
9	500 × 30	0.000	3,600.00	3,600.00

Notes: Gap – gap between CM and BB, t_BB – BB computational time and t_CM – CM computational time

Table 6 CM exact method: randomly generated problems

P.	$n \times m$	*Gap* ($^0/_{00}$)	*t_BB (s)*	*t_CM (s)*
UD	50 × 25	0.000	0.00	0.00
UD	100 × 50	0.000	2.80	2.00
UD	150 × 75	0.000	21.40	18.60
UD	200 × 100	–0.017	1,693.30	1,656.90
UD	300 × 150	–0.004	3,600.00	3,600.00
UD	400 × 200	0.001	3,600.00	3,600.00
UD	500 × 250	0.000	3,600.00	3,600.00
CD	50 × 5	–0.010	2,928.50	2,716.80
CD	100 ×10	–1.00	3,600.00	3,600.00
CD	150 × 15	–0.893	3,600.00	3,600.00
CD	200 × 20	–0.694	3,600.00	3,600.00
CD	250 × 25	–0.131	3,600.00	3,600.00

Notes: UD – instance with uncorrelated data, CD – instance with correlated data, Gap – gap between CM and BB, t_BB – BB computational time and t_CM – CM computational time

These results can be explained by the good lower bound provided by the HDP phase. This permits one to fathom unpromising nodes saved during the HDP phase in $\mathcal{L}_{\text{sec}}$ and to focus the work of the BB phase on the hard nodes.

Furthermore, computational experiences validate the fact that the HDP phase can cooperate with an enumerative method, like BB, in order to explore entirely all the states in $\mathcal{L}_{\text{sec}}$ and that the resulting exploration overhead is not prohibitive. This highlights the flexibility of our approach.

7 Conclusions

The main advantage of the HDP heuristics is to obtain a processing time similar to the one of DP 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.

Cooperation of BB with HDP permits one to obtain an exact method. Computing experiments on problems from the literature show that the combination of HDP and BB gives a processing times similar to the one of a classical BB. However, this CM seems to improve the convergence toward the optimal value.

HDP could be combined easily with other methods, like a Taboo search, in order to improve the lower bounds by performing exploration of the neighbourhood of the states saved in the secondary list; this solution could be an alternative to limit the processing time.

References

Beasley, J.E. (1990) *Multidimensional Knapsack Problem*, OR-Library, available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib-/mknapinfo.html.

Bellman, R. (1957) *Dynamic Programming*, Princeton University Press, Princeton.

Bertsimas, D. and Demir, R. (2002) 'An approximate dynamic-programming approach to multi-dimensional knapsack problem', *Management Science*, Vol. 4, pp.550–565.

Boyer, V. (2004) *Méthodes et/ou Mixte pour la Programmation Linéaire en Variables 0–1*, DEA report, LAAS-CNRS Toulouse, France.

Boyer, V., El Baz, D. and Elkihel, M. (2006) 'An efficient heuristics for the multidimensional knapsack problem', *ROADEF'06, Presses Universitaires de Valenciennes*, pp.95–106.

Boyer, V., El Baz, D. and Elkihel, M. (2008) 'Heuristics for the 0–1 multidimensional knapsack problem', *European Journal of Operational Research*, doi: 10.1016/j.ejor.2007.06.068.

Eilon, S. and Christofides, N. (1971) 'The loading problem', *Management Science*, Vol. 17, pp.259–268.

Elkihel, M., Authié, G. and Viader, F. (2002) 'An efficient hybrid dynamic algorithm to solve 0–1 knapsack problems', *International Symposium on Combinatorial Optimization (CO'2002)*, Paris, France.

Fréville, A. (2004) 'The multidimensional 0–1 knapsack problem: an overview', *European Journal of Operational Research*, Vol. 155, pp.1–21.

Fréville, A. and Plateau, G. (1993) 'An exact search for the solution of the surrogate dual of the 0–1 bidimensional knapsack problem', *European Journal of Operational Research*, Vol. 68, pp.413–421.

Fréville, A. and Plateau, G. (1994) 'An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem', *Discrete Applied Mathematics*, Vol. 49, pp.189–212.

Garfinkel, S. and Nemhauser, L. (1972) *Integer Programming*, Wiley Interscience.

Gavish, B. and Pirkul, H. (1982) 'Allocation of data bases and processors in a distributed computing system', *Management of Distributed Computing Data Processing*, Vol. 31, pp.215–231.

Gavish, B. and Pirkul, H. (1985) 'Efficient algorithms for solving multiconstraint 0–1 knapsack problems to optimality', *Mathematical Programming*, Vol. 31, pp.78–205.

Glover, F. (1968) 'Surrogate constraints', *Operations Research*, Vol. 16, pp.741–749.

Hanafi, S., Fréville, A. and El Abdellaoui, A. (1996) 'Meta-heuristics: theory and application, chapter comparaison of heuristics for the 0–1 multidimensional knapsack problem', Kluwer Academic, pp.446–465.

Kellerer, H., Pferschy, U. and Pisinger, D. (2004) *Knapsack Problems*, Springer.

Kolesar, P.J. (1967) 'A branch and bound algorithm for the knapsack problem', *Management Science*, Vol. 13, pp.723–735.

Lorie, J. and Savage, L.J. (1955) 'Three problems in capital rationing', *The Journal of Business*, Vol. 28, pp.229–239.

Martello, S. and Toth, P. (1982) 'A mixed algorithm for the subset-sum problem', EURO V, TIMS, Lausanne, France.

Martello, S. and Toth, P. (1990) *Knapsack Problems – Algorithms and Computer Implementations*, Wiley & Sons.

Martello, S., Pisinger, D. and Toth, P. (2000) 'New trends in exact algorithms for the 0–1 knapsack problem', *European Journal of Operational Research*, Vol. 123, pp.325–332.

Nemhauser, L. and Wolsey, A. (1988) *Integer and Combinational Optimization*, Wiley Interscience.

Osorio, M., Glover, F. and Hammer, P. (2002) 'Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions', *Annals of Operations Research*, Vol. 117, pp.71–93.

Plateau, G. and Elkihel, M. (1985) 'A hybrid method for the 0–1 knapsack problem', *Methods of Operations Research*, Vol. 49, pp.277–293.

Sherali, D. and Driscoll, J. (2000) 'Evolution and state-of-the-art in integer programing', *Journal of Computational and Applied Mathematics*, Vol. 124, pp.319–340.