European Journal of Operational Research 199 (2009) 658–664

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Heuristics for the 0–1 multidimensional knapsack problem

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

LAAS-CNRS, Université de Toulouse, 7 Avenue du Colonel Roche, 31077 Toulouse Cedex 4, France

ARTICLE INFO

Article history:
Received 9 September 2006
Accepted 8 June 2007
Available online 13 April 2008

Keywords:
Multidimensional knapsack problem
Dynamic-programming
Branch-and-cut
Surrogate relaxation
Heuristics

ABSTRACT

Two heuristics for the 0–1 multidimensional knapsack problem (MKP) are presented. The first one uses surrogate relaxation, and the relaxed problem is solved via a modified dynamic-programming algorithm. The heuristics provides a feasible solution for (MKP). The second one combines a limited-branch-and-cut-procedure with the previous approach, and tries to improve the bound obtained by exploring some nodes that have been rejected by the modified dynamic-programming algorithm. Computational experiences show that our approaches give better results than the existing heuristics, and thus permit one to obtain a smaller gap between the solution provided and an optimal solution.

1. Introduction

The NP-hard multidimensional knapsack problem (MKP) (see [10,13,20]) arises in several practical problems such as capital budgeting, cargo loading, cutting stock problem and processors allocation in huge distributed systems. It can be defined as

$$(\text{MKP})\begin{cases}\max \sum_{j\in N} p_j \cdot x_j, \\ \text{subject to} \sum_{j\in N} w_{i,j}\cdot x_j \leqslant c_i, \quad \forall i \in M, \\ x_j \in \{0,1\}, \quad \forall j \in N,\end{cases} \tag{1}$$

where

- $N = \{1, 2, \ldots, n\}$ and $M = \{1, 2, \ldots, m\}$,
- n is the number of items,
- m is the number of constraints,
- $p_j \geqslant 0$ is the profit of the jth item,
- $w_{i,j} \geqslant 0$, for $i \in M$, are the weights of the jth item,
- and $c_i \geqslant 0$, for $i \in M$, are the capacities of the knapsack.

In the sequel, we shall use the following notation: given a problem (P), its optimal value will be denoted by $v(P)$.

To avoid any trivial solution, we assume that

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

A specific case of (MKP) is the classical knapsack problem with $m = 1$. The unique knapsack problem (UKP) has been given considerable 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 [2,3,6,11,12]). We have then tried to transform the original (MKP) into a (UKP) (see also [15,17]). In this purpose, we have used a relaxation technique, that is to say, surrogate relaxation. The surrogate relaxation of (MKP) can be defined as follows:

$$(S(u))\begin{cases}\max \sum_{j\in N} p_j \cdot x_j, \\ \text{subject to} \sum_{i\in M} u_i \cdot \sum_{j\in N} w_{i,j}\cdot x_j \leqslant \sum_{i\in M} u_i \cdot c_i, \\ x_j \in \{0,1\}, \quad \forall j \in N,\end{cases} \tag{2}$$

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

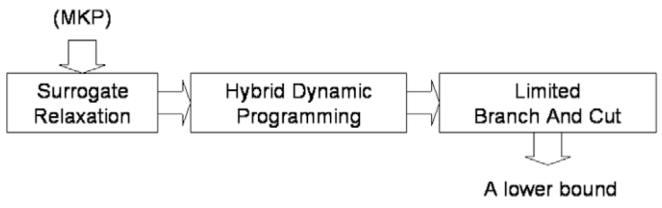
Since $(S(u))$ is a relaxation of (MKP), we have $v(S(u)) \geqslant v(\text{MKP})$, and the optimal multiplier vector, u^*, is defined as

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

Several heuristics have been proposed in order to find out good surrogate multipliers (see in particular [14,15,17]). In practice, it is not important to obtain the optimal multiplier vector, since in the general case we have no guarantee that $v(S(u^*)) = v(\text{MKP})$. Solving $(S(u))$ will give an upper bound of (MKP). In the sequel, we propose efficient algorithms based on dynamic-programming in order to find a good lower bound of (MKP) by solving $(S(u))$.

The basic algorithmic scheme can be presented as follow:

[*] Corresponding author.
E-mail addresses: vboyer@laas.fr (V. Boyer), elkihel@laas.fr (M. Elkihel), elbaz@laas.fr (D. El Baz).



We have studied first a heuristics based on dynamic-programming and surrogate relaxation of (MKP). We remarked that some of the states eliminated by the heuristics could be further explored in order to improve the bound. Thus, we designed a new heuristics which is based on a limited-branch-and-cut-procedure and that is added to first heuristics. In this last case, the following 2 phases can be considered:

– *Phase 1*: Surrogate relaxation + Hybrid-dynamic-programming.
– *Phase 2*: Limited-branch-and-cut.

The sole Phase 1 and the combination of the two phases represent the two heuristics proposed in this article. The first phase provides a lower bound of (MKP) and a list of states that will be treated in the second phase in order to improve the bound.

Solutions obtained with the above heuristics are compared with results given by heuristics from the literature such as:

– AGNES of Freville and Plateau [7].
– ADP-based heuristics approach of Bertsimas and Demir [8].
– Simple Multistage Algorithm (SMA) of Hanafi, Freville and El Abdellaoui [9].

Note that AGNES combines different greedy methods based on surrogate relaxation and the solutions provided are improved by a neighborhood search (a neighborhood is defined around a solution and is explored to find out a better solution). The ADP-based heuristics uses a diversification method, a taboo algorithm, completed too by a neighborhood search. The Simple Multistage Algorithm is an approximate dynamic-programming approach.

The first phase of our method can be seen as a diversification method which provides a family of feasible solutions for (MKP), and we try to improve the bound by exploring the neighborhood of these solutions with the second phase. This approach allows us to make a real cooperation between the two phases.

In the sequel, each steps of our algorithm is described.

The paper is organized as follows. In Section 2, we present the hybrid-dynamic-programming algorithm (HDP) and in Section 3 a cooperative method: the so-called limited-branch-and-cut method (LBC). Finally, in Section 4, we provide and analyze some computational results obtained with different instances from the literature and randomly generated instances. Our heuristics are also compared with other existing heuristics.

2. The surrogate relaxation

Solving (3) is not easy. As mentioned above, many heuristics exist and provide good approximations of u^*. A reasonable estimation can be calculated by dropping the integer restrictions in x. In other words, let

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

The best surrogate constraint is then generated by u^0, where

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

In order to calculate the best surrogate constraint we consider the linear programing (LP) corresponding to (MKP)

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

We denote by $\lambda^0 = (\lambda_1^0, \lambda_2^0, \ldots, \lambda_m^0) \geqslant 0$ the dual optimal variables corresponding to the constraints

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

We are now ready to show how to calculate the best surrogate constraint using the definition (5).

Theorem 1 (see [16, p. 132]). *The best surrogate constraint is generated by* $u^0 = \lambda^0$.

Then we have the following order relation (see [15,16, p. 130] and [18]):

$$v(LP) = v(LS(u^0)) \geqslant v(S(u^*)) \geqslant v(\text{MKP}). \tag{8}$$

Table 1 gives the bounds obtained with the surrogate relaxation for a set of instances from the literature.

3. Hybrid-dynamic-programming (HDP)

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

$$(S(u))\begin{cases}\max \sum_{j\in N} p_j\cdot x_j,\\ \text{subject to} \sum_{j\in N} w_j\cdot x_j \leqslant c,\\ x_j\in\{0,1\},\quad \forall j\in N.\end{cases} \tag{9}$$

We apply the dynamic-programming list algorithm to $(S(u^0))$ and we keep only the feasible solutions of (MKP). At each step, we update a list which is defined as follows:

$$\text{For } k\in N,\quad \mathscr{L}_k = \left\{(w,p)|w=\sum_{j=1}^{k} w_j\cdot x_j \leqslant c,\ p=\sum_{j=1}^{k} p_j\cdot x_j\right\} \tag{10}$$

The use of the concept of dominated states can permit one to reduce drastically the size of lists $\mathscr{L}_k$ since dominated states, according to Bellman's optimality principle, can be eliminated from the list:

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 $w'\leqslant w$ and $p'\geqslant p$, then (w,p) is dominated by (w',p').

Note that dominated states must be saved in a secondary list denoted by $\mathscr{L}_{\text{sec}}$ since they can nevertheless give rise to an optimal solution of (MKP).

3.1. Dynamic-programming algorithm (DP)

List of states $\mathscr{L}_{k+1}$ are generated recursively by the dynamic-programming list algorithm (see [5] for more details and some examples). At stage $k+1$ the list $\mathscr{L}_{k+1}$ is constructed as follows:

The set of new states generated at stage $k+1$ is given by

$$\begin{aligned}\mathscr{L}'_{k+1} &= \mathscr{L}_k \oplus (w_{k+1}, p_{k+1})\\ &= \{(w+w_{k+1}, p+p_{k+1})|(w,p)\in\mathscr{L}_k \quad\text{and}\quad w+w_{k+1}\leqslant c\},\end{aligned}$$

Table 1
Surrogate relaxation on some instances from the literature

Name of the instance	$n \times m$	$v(\text{MKP})$	$v(LP)$	$v(LS(u^0))$	$v(S(u^0))$
Petersen 1	6×10	3800	4134.07	4134.07	3800
Petersen 2	10×10	87,061	92,977.70	92,977.70	91,779
Petersen 3	15×10	4015	4127.89	4127.89	4105
Petersen 4	20×10	6120	6155.33	6155.33	6120
Petersen 5	28×10	12,400	12,462.10	12462.10	12,440
Petersen 6	39×5	10,618	10,672.35	10,672.35	10,662
Petersen 7	50×5	16,537	16,612.82	16,612.82	16,599
Hansen and Plateau 1	28×4	3418	3472.35	3472.35	3462
Hansen and Plateau 2	35×4	3186	3261.82	3261.82	3248
Weingartner 1	28×2	141,278	14,2019.00	142,019.00	141,548
Weingartner 2	28×2	130,083	13,1637.47	131,637.47	130,883
Weingartner 3	28×2	95,677	99,647.08	99,647.08	97,906
Weingartner 4	28×2	119,337	122,505.25	122,505.25	121,087
Weingartner 5	28×2	98,796	100,433.16	100,433.16	98,796
Weingartner 6	28×2	130,623	131,335.00	131,335.00	130,733
Weingartner 7	105×2	1,095,445	1,095,721.25	1,095,721.25	1,095,591
Weingartner 8	105×2	624,319	628,773.69	628,773.69	627,976

and we have: $\mathscr{L}_{k+1} := \mathscr{L}_k \cup \mathscr{L}'_{k+1} - \mathscr{D}_{k+1}$, where $\mathscr{D}_{k+1}$, the set of dominated pairs at stage $k+1$, is defined as follow:

$$\mathscr{D}_{k+1} = \{(w,p) | (w,p) \in \mathscr{L}_k \cup \mathscr{L}'_{k+1} \text{ and } \exists (w',p') \in \mathscr{L}_k \cup \mathscr{L}'_{k+1} \text{ with } w' \leqslant w,\ p \leqslant p' \text{ and } (w',p') \neq (w,p)\}.$$

Initially, we have $\mathscr{L}_0 = \{(0,0)\}$.

Let (w,p) be a state generated at stage k, we define the subproblem associated with (w,p) by

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

An upper bound, $\bar{v}_{(w,p)}$, of the above problem, is obtained by solving the linear relaxation of $(S(u))_{(w,p)}$, i.e. $(LS(u))_{(w,p)}$, with the Martello and Toth algorithm (see [4]) and a lower bound, $\underline{v}_{(w,p)}$, is obtained with a greedy algorithm on $(S(u))_{(w,p)}$.

In a list, all the states are sorted by their decreasing upper bound. As mentioned above, our algorithm consists in applying dynamic-programming (DP) to solve $S(u^0)$. At each stage of DP, the following points are checked when a new state (w,p) is generated:

– 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), $\underline{v}(\text{MKP})$, with the value of p.
– Is the state dominated? In this case the state is saved in the secondary list $\mathscr{L}_{\text{sec}}$.
– Is the upper bound associated with the state (w,p) smaller than the current lower bound of $S(u^0)$? Then the state is saved too in the secondary list $\mathscr{L}_{\text{sec}}$.

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.

DP algorithm is described below:

```
**Dynamic-programming algorithm (DP):**
*Initialisation:*
  $\mathscr{L}_0 = \{(0,0)\}$, $\mathscr{L}_{\text{sec}} = \emptyset$
  $\underline{v}(S(u^0)) = \underline{v}(\text{MKP})$ (where $\underline{v}(\text{MKP})$ is a lower bound of (MKP) given by a greedy algorithm).
*Computing the lists:*
  For $j := 1$ to $n$
    $\mathscr{L}'_j := \mathscr{L}_{j-1} \oplus (w_j, p_j)$
    Remove all states $(w,p) \in \mathscr{L}'_j$ that are unfeasible for (MKP)
    $\mathscr{L}_j := \text{MergeLists}(\mathscr{L}_{j-1}, \mathscr{L}'_j)$
    For each state $(w,p) \in \mathscr{L}_j$ Compute $\bar{v}_{(w,p)}$ and $\underline{v}_{(w,p)}$
    *Updating the bounds:*
      $p_{max} := max\{p | (w,p) \in \mathscr{L}_j\}$ and $v_{max} := max\{\underline{v}_{(w,p)} | (w,p) \in \mathscr{L}_j\}$
      $\underline{v}(\text{MKP}) := max\{\underline{v}(\text{MKP}), p_{max}\}$
      $\underline{v}(S(u^0)) := max\{\underline{v}(S(u^0)), v_{max}\}$
    *Updating $\mathscr{L}_{\text{sec}}$:*
      $\mathscr{D}_j = \{(w,p) | (w,p) \text{ is dominated or } \bar{v}_{(w,p)} \leqslant \underline{v}(S(u^0))\}$
      $\mathscr{L}_{\text{sec}} := \mathscr{L}_{\text{sec}} \cup \mathscr{D}_j$ and $\mathscr{L}_j := \mathscr{L}_j - \mathscr{D}_j$
  End for.
```

In the end of the algorithm, we obtain a lower bound of (MKP). In order to improve the lower bound and the efficiency of DP algorithm, we add to the algorithm a reducing-variable procedure, which is defined 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 $x_j = 1 - k$.

These upper bounds are obtained with the Martello and Toth algorithm on $(S(u^0))$. We use this reducing-variables rule whenever we improve $\underline{v}(\text{MKP})$ 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 are unfeasible. The results of DP are presented in Table 2.

3.2. Improvement of the lower bound (ILB)

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 which are uncompatible with the variables that have been yet reduced or that have an upper bound smaller than the current lower bound of (MKP), i.e. $\underline{v}(\text{MKP})$.

For a state (w,p), let J be the index set of free variables and $I = N - J$ the set of fixed variables. If the states have been generated at the kth stage of DP Algorithm, $J = \{k+1,\dots,n\}$, $w = \sum_{k}^{j=1} w_j \cdot x_j$ and $p = \sum_{j=1}^{k} p_j \cdot x_j$, where x_j, $j \in I$ denote here the values of the component of vector x which have been already compute during the k first step of the dynamic-programming list method. Then we define the new subproblem:

$$(\text{MKP})_{(w,p)}\begin{cases}\max \sum_{j\in J} p_j \cdot x_j + p,\\ \text{subject to } \sum_{j\in J} w_{i,j}\cdot x_j \leqslant \bar{c}_i, \quad \forall i \in M,\\ x_j \in \{0,1\}, \quad \forall j \in J,\end{cases} \tag{12}$$

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

Two methods are used in order to evaluate the lower bound of the above problem:

– 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: $n' \leqslant \alpha$).

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 $(w,p) \in \mathscr{L}_{\text{sec}}$
Compute $\underline{v}_{(w,p)}$ a lower bound of $(\text{MKP})_{(w,p)}$
Endfor.
$v_{\max} := \max\{\underline{v}_{(w,p)} | (w,p) \in \mathscr{L}_{\text{sec}}\}$.
$\underline{v}(\text{MKP}) := \max\{\underline{v}(\text{MKP}), v_{\max}\}$.
```

The improvement given by ILB (with low cost in term of processing time) can be clearly seen from the comparison of Tables 2 and 3. Empirically, the value $\alpha = 10$ has given the better results.

Table 2
Lower bounds given by DP

Name of the instance	$n \times m$	v(MKP)	$\underline{v}$(MKP)	Gap (%)
Petersen 1	6×10	3800	3700	2.63
Petersen 2	10×10	87,061	83,369	4.24
Petersen 3	15×10	4015	3245	19.18
Petersen 4	20×10	6120	6010	1.80
Petersen 5	28×10	12,400	11,930	3.79
Petersen 6	39×5	10,618	10,313	2.87
Petersen 7	50×5	16,537	16,449	0.53
Hansen and Plateau 1	28×4	3418	3347	2.08
Hansen and Plateau 2	35×4	3186	3098	2.76
Weingartner 1	28×2	141,278	140,477	0.57
Weingartner 2	28×2	130,083	130,723	0.12
Weingartner 3	28×2	95,677	95,627	0.05
Weingartner 4	28×2	119,337	104,799	12.18
Weingartner 5	28×2	98,796	98,796	0.00
Weingartner 6	28×2	130,623	130,233	0.30
Weingartner 7	105×2	1,095,445	1,094,757	0.06
Weingartner 8	105×2	624,319	619,101	0.84

Table 3
Lower bounds with ILB, $\alpha = 10$

Name of the instance	$n \times m$	v(MKP)	$\underline{v}$(MKP)	Gap (%)
Petersen 1	6×10	3800	3800	0.00
Petersen 2	10×10	87,061	87,061	0.00
Petersen 3	15×10	4015	4015	0.00
Petersen 4	20×10	6120	6120	0.00
Petersen 5	28×10	12,400	12,400	0.00
Petersen 6	39×5	10,618	10,618	0.00
Petersen 7	50×5	16,537	16,508	0.18
Hansen and Plateau 1	28×4	3418	3418	0.00
Hansen and Plateau 2	35×4	3186	3148	1.19
Weingartner 1	28×2	141,278	141,278	0.00
Weingartner 2	28×2	130,083	130,083	0.00
Weingartner 3	28×2	95,677	95,677	0.00
Weingartner 4	28×2	119,337	119,337	0.00
Weingartner 5	28×2	98,796	98,796	0.00
Weingartner 6	28×2	130,623	130,623	0.00
Weingartner 7	105×2	1,095,445	1,095,445	0.00
Weingartner 8	105×2	624,319	624,319	0.00

4. Limited-branch-and-cut (LBC)

In this section we present the last part of our algorithm, it permits one to improve the lower bound provided by the (ILB) procedure. As mentioned above, the states in the secondary list $\mathscr{L}_{\text{sec}}$ can give rise to better results for (MKP). We propose an algorithm based on a branch-and-cut method in order to explore a neighborhood of the states in $\mathscr{L}_{\text{sec}}$.

4.1. Classic branch-and-cut

Let (w,p) be the first state of $\mathscr{L}_{\text{sec}}$ (the states are sorted according to their decreasing upper bounds).

An upper bound of $(\text{MKP})_{(w,p)}$, $\bar{v}_{(w,p)}$, is obtained by solving its linear relaxation, using a simplex algorithm, and a lower bound, $\underline{v}_{(w,p)}$, is obtained with a greedy algorithm on $(\text{MKP})_{(w,p)}$.

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 and $\widetilde{X_J} = \{\tilde{x}_j | j \in J\}$ an optimal solution of the linear relaxation of $(\text{MKP})_{(w,p)}$. 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-variable method:

Reducing-variables rule 2 (see [19]): Let $\underline{v}$ be a lower bound of (MKP). Let $\tilde{v}$ be the optimal bound and $\widetilde{X} = \{\tilde{x}_j | j \in N\}$ an optimal solution of the linear relaxation of (MKP). Then we denote by $\widetilde{P} = \{\tilde{p}_j | j \in N\}$, the reduced profits. For $j \in N$, if $\tilde{x}_j = 0$, $\tilde{x}_j = 1$, respectively, and $\tilde{v} - |\tilde{p}_j| \leqslant \underline{v}$, then there exists an optimal solution for (MKP) with $x_j = 0$, $x_j = 1$, respectively.

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

4.2. Limited-branch-and-cut (LBC)

We propose a method, based on the branch-and-cut technique described above, to explore quickly the states saved in the secondary list. Indeed, at each step of the algorithm, we enforce the value of the variables in order to limit the processing time. We use the following heuristics to fix variables:

Reducing-variables rule 3: Let $\tilde{v}$ be the optimal bound and $\widetilde{X} = \{\tilde{x}_j | j \in N\}$ an optimal solution of the linear relaxation of (MKP). For $j \in N$, if $\tilde{x}_j = 0$, $\tilde{x}_j = 1$, respectively, then x_j is fixed to 0, 1, respectively.

In order to limit the exploration, we consider only 50% of the secondary list, that is to say, we decide to consider only half the states in $\mathscr{L}_{\text{sec}}$ with the best upper bounds.

Procedure LBC:

```
Assign to $\underline{v}$(MKP) the value of the lower bound returned by ILB algorithm.
While $\mathscr{L}_{\text{sec}} \neq \emptyset$
    Let $(w,p)$ be the first state in $\mathscr{L}_{\text{sec}}$
    $\mathscr{L}_{\text{sec}} := \mathscr{L}_{\text{sec}} - (w,p)$
    Compute $\bar{v}_{(w,p)}$ an upper bound of $(\text{MKP})_{(w,p)}$
    If $\bar{v}_{(w,p)} > \underline{v}(\text{MKP})$
        Fix variables according to reducing-variables rule 3 and update $p$
        Compute $\underline{v}_{(w,p)}$ a lower bound of $(\text{MKP})_{(w,p)}$
        If $\underline{v}_{(w,p)} > \underline{v}(\text{MKP})$ then $\underline{v}(\text{MKP}) := \underline{v}_{(w,p)}$ Endif
        Chose the branching variable and branch on it
        Insert the two resulting states in $\mathscr{L}_{\text{sec}}$ if they are feasible
    Endif
Endwhile.
```

Table 4
Small instances from the literature

Name of the instance	Number of instances	Average gap to optimality (%)			
		HDP	SMA	ADP	AGNES
Petersen	7	0.02	8.24	1.62	1.05
Hansen and Plateau	2	0.45	8.34	7.28	1.44
Weingartner	8	0.01	4.67	4.05	3.37
Freville and Plateau	6	0.44	12.85	6.86	1.91
Fleisher	1	0.00	12.16	0.00	3.60
Sent	2	0.00	1.65	0.35	0.20

5. Computational experiences

Our heuristics were programmed in C and compiled with GNU's GCC. Computational experiences were carried out using an Intel Pentium M Processor 725. We compare our heuristics to the following heuristics of the literature:

– AGNES of Freville and Plateau [7].
– ADP-based heuristic approach of Bertsimas and Demir [8].
– Simple Multistage Algorithm (SMA) of Hanafi, Freville and El Abdellaoui [9].

Our tests were made on the following instances:

– small instances from the literature of Petersen, Weingartner, Hansen and Plateau, Freville and Plateau, Fleisher and Sent ([1]).
– Large instances from the literature of Chu and Beasley ([1]).
– Randomly generated instances:

$$(\text{MKP})\begin{cases} \max \sum_{j\in N} p_j \cdot x_j, \\ \text{subject to} \sum_{j\in N} w_{i,j}\cdot x_j \leqslant \epsilon . \sum_{j\in N} w_{i,j}, \quad \forall i \in M, \\ x_j \in \{0,1\}, \quad \forall j \in N, \end{cases} \tag{13}$$

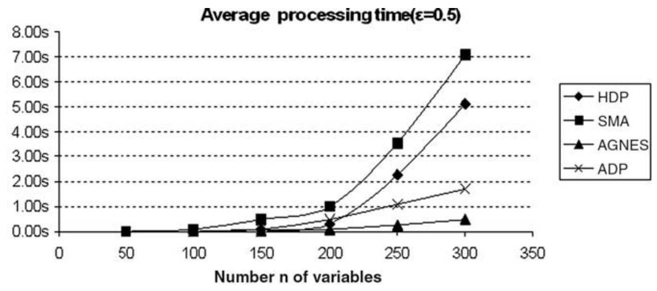
where $N = \{1,2,\ldots,n\}$, $M = \{1,2,\ldots,]\frac{n}{2}[\}$, $p_j \in [0,1000]$ for all $j \in N$, $w_{i,j} \in [0,1000]$, for all $i \in M$, $j \in N$ and $\epsilon \in]0,1[$.

Note that, for the instances randomly generated and from Chu and Beasley, the bounds provided by the heuristics are compared with the optimal solution of the corresponding linear relaxation (see (6)).

5.1. HDP heuristics

5.1.1. Instances from the literature

From Tables 4 and 5, we note that the lower bound given by HDP is better than the others. It is difficult to compare processing time with the small instances since it takes less than 1 second to solve all at once. In Table 5, we could remark that for the seven first sets we have a competiting processing time compared with other heuristics.



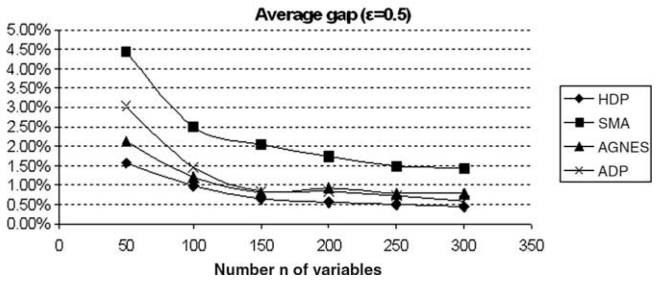
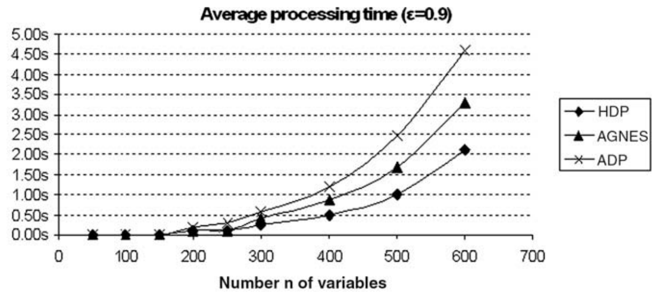


Fig. 1. Average performances with $\epsilon = 0.5$.



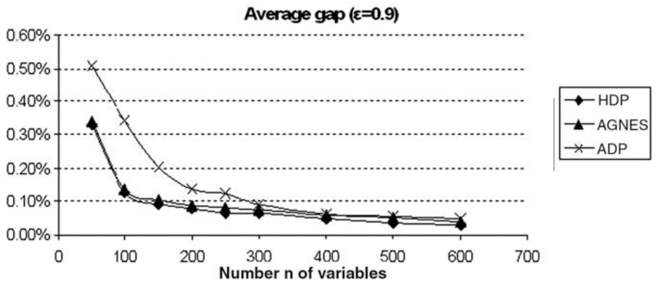


Fig. 2. Average performances with $\epsilon = 0.9$.

Table 5
Large instances from the literature

Name of the instance	Number of instances	Size $n \times m$	Average gap (%)				Average processing time (s)			
			HDP	SMA	ADP	AGNES	HDP	SMA	ADP	AGNES
Chu and Beasley 1	30	100×5	0.69	2.68	1.72	0.88	<0.001	0.03	0.03	<0.001
Chu and Beasley 2	30	250×5	0.22	1.17	0.58	0.29	0.10	0.57	0.07	<0.001
Chu and Beasley 3	30	500×5	0.08	0.59	0.26	0.12	0.53	4.57	0.30	0.10
Chu and Beasley 4	30	100×10	1.22	3.60	1.97	1.54	0.03	0.03	0.03	<0.001
Chu and Beasley 5	30	250×10	0.46	1.60	0.76	0.57	0.17	0.70	0.10	0.03
Chu and Beasley 6	30	500×10	0.21	0.80	0.38	0.26	0.83	5.70	0.43	0.13
Chu and Beasley 7	30	100×30	2.04	5.13	2.70	3.22	1.67	0.10	0.07	0.03
Chu and Beasley 8	30	250×30	0.89	2.60	1.18	1.41	9.87	1.40	0.37	0.10
Chu and Beasley 9	30	500×30	0.48	1.45	0.58	0.72	26.37	11.93	1.20	0.30

Table 6
Small instances from the literature

Name of the instance	Number of instances	Average gap to optimality (%)	
		HDP	HDP + LBC
Petersen	7	0.02	0.02
Hansen and Plateau	2	0.45	0.45
Weingartner	8	0.01	0.00
Freville and Plateau	6	0.44	0.20
Fleisher	1	0.00	0.00
Sent	2	0.00	0.00

5.1.2. Randomly generated instances

We have compared processing time in function of the number n of variables of (MKP) and ϵ. For a given number n of variables we have generated randomly 25 instances in order to compute average performance of our heuristics.

The average performance is represented in Figs. 1 and 2, respectively for $\epsilon = 0.5$ and $\epsilon = 0.9$. Computational experiences show that we have obtained better bounds than other heuristics, but with $\epsilon = 0.5$, processing time tends to be important. That is not surprising as HDP is constructed with a dynamic-programming algorithm.

5.2. HDP + LBC heuristic

Adding LBC procedure leads to greater computing time but permits one to have a better approximation of the optimal bound. In this section we will compare the bounds provided by HDP + LBC with those given by HDP alone.

5.2.1. Instances from the literature

Table 6 shows that for the instances of Freville and Plateau, LBC improves significantly the lower bounds. For the other instances, as the bound provided by HDP is very close to the optimal one, it is hard to improve it. Processing time to solve all these instances at once is about 1 seconds. For the large instances presented in Table 7, HDP + LBC improves in all cases the bound provided by HDP. Note that, in order to limit the processing time, we stop LBC if the time spend in this procedure exceeds the one of HDP.

5.2.2. Randomly generated instances

According to the results presented in Figs. 3 and 4, we can see that the improvement of the lower bounds is better for instances with a small capacity. Then, when ϵ is close to 0.9, we cannot use LBC procedure, but sometimes it permits one to reduce the gap with the optimal solution. As the bound provided by HDP is close to the optimal one, it is not easy to improve it, then, in some cases, the LBC procedure could not give a better approximation. Although processing time seems to be important, we have on average 3 times less time than an exact method with a good approximation of the optimal solution.

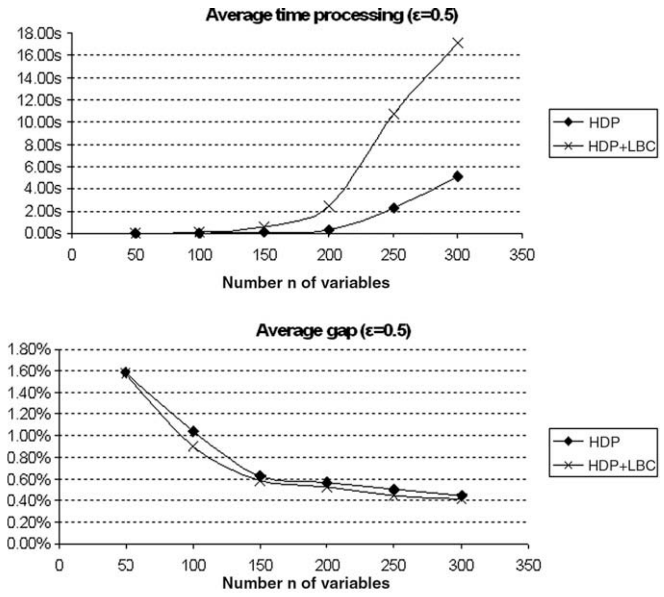


Fig. 3. Average performances with $\epsilon = 0.5$.

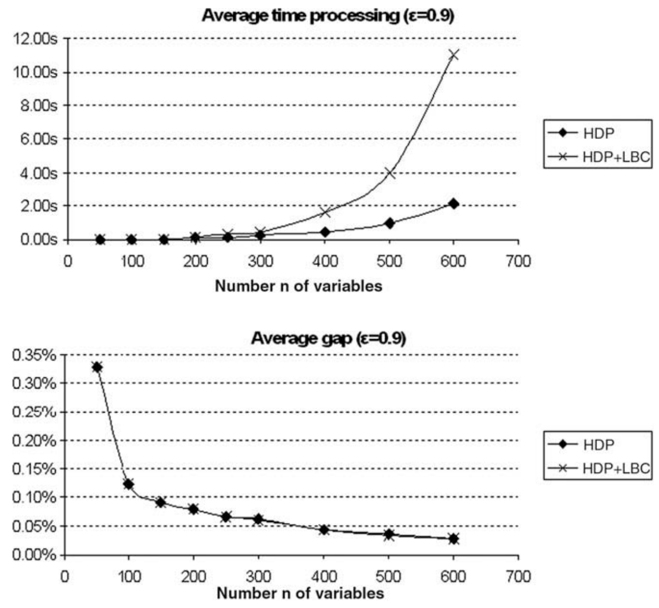


Fig. 4. Average performances with $\epsilon = 0.9$.

Table 7
Large instances from the literature

Name of the instance	Number of instances	Size $n \times m$	Average gap (%)		Average processing time (s)	
			HDP	HDP + LBC	HDP	HDP + LBC
Chu and Beasley 1	30	100×5	0.69	0.57	<0.001	0.60
Chu and Beasley 2	30	250×5	0.22	0.16	0.10	1.00
Chu and Beasley 3	30	500×5	0.08	0.07	0.53	1.13
Chu and Beasley 4	30	100×10	1.22	0.95	0.03	1.00
Chu and Beasley 5	30	250×10	0.46	0.32	0.17	0.97
Chu and Beasley 6	30	500×10	0.21	0.16	0.83	4.03
Chu and Beasley 7	30	100×30	2.04	1.81	1.67	3.97
Chu and Beasley 8	30	250×30	0.89	0.77	9.87	21.20
Chu and Beasley 9	30	500×30	0.48	0.42	26.37	93.37

6. Conclusion

The main idea of HDP is to obtain a processing time similar to the one of dynamic-programming algorithm applied to a classical (UKP) while having good performance in terms of gap. Removing the improvement procedure could sometimes diminish the processing time by two but deteriorates significantly the lower bound. Nevertheless HDP seems to be a good heuristics as it gives a better solution than the existing ones with a quite good processing time.

Using a procedure like LBC improves the lower bound obtained by HDP with a smaller computing time than an exact method. Computing results with instances from the literature show that we increase the occurrence of obtaining the optimal solution from 76% up to 84%. Then using HDP + LBC permits one to have a good approximation of the optimal bound.

A work on the robustness of the algorithm has to be done, by studying a good preprocessing procedure for example in order to reduce the processing time.

References

[1] OR-Library, J.E. Beasley, <http://www.people.brunel.ac.uk/mastjjb/jeb/orlib/mknapinfo.html>.
[2] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, 2004.
[3] S. Martello, D. Pisinger, P. Toth, New trends in exact algorithms for the 0–1 knapsack problem, European Journal of Operational Research 123 (2000) 325–332.
[4] S. Martello, P. Toth, Knapsack Problems – Algorithms and Computer Implementations, Wiley and Sons, 1990.
[5] 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 65 (2005) 74–84.
[6] G. Plateau, M. Elkihel, A hybrid method for the 0–1 knapsack problem, Methods of Operations Research 49 (1985) 277–293.
[7] A. Freville, G. Plateau, An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem, Discrete Applied Mathematics 49 (1994) 189–212.
[8] D. Bertsimas, R. Demir, An approximate dynamic-programming approach to multi-dimensional knapsack problem, Management Science 4 (2002) 550–565.
[9] S. Hanafi, A. Freville, A. El Abdellaoui, Comparison of heuristics for the 0–1 multidimensional knapsack problem, Meta-Heuristics: Theory and Application, Kluwer Academic, 1996. pp. 446–465.
[10] V. Boyer, Méthodes et/ou mixte pour la programmation linéaire en variables 0–1, DEA report, LAAS-CNRS Toulouse (2004).
[11] M. Elkihel, Programmation dynamique et rotations de contraintes pour les problèmes d'optimisation entière, Ph.D Thesis, Université des Sciences et Techniques de Lille, 1984.
[12] G. Plateau, Contribution à la résolution des programmes mathématiques en nombres entiers, Thèse d'Etat de l'Université des Sciences et Techniques de Lille (1979).
[13] A. Freville, The multidimensional 0–1 knapsack problem: An overview, European Journal of Operational Research 155 (2004) 1–21.
[14] A. Freville, 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 (1993) 413–421.
[15] B. Gavish, H. Pirkul, Efficient algorithms for solving multiconstraint 0–1 knapsack problems to optimality, Mathematical Programming 31 (1985) 78–205.
[16] S. Garfinkel, L. Nemhauser, Integer Programming, Wiley Interscience, 1972.
[17] F. Glover, Surrogate constraints, Operations Research 16 (1968) 741–749.
[18] M. Osario, F. Glover, P. Hammer, Cutting and surrogate analysis for improved multidimensional knapsack problem, Annal of Operations Research 117 (2002) 71–93.
[19] L. Nemhauser, A. Wolsey, Integer and Combinational Optimization, Wiley Interscience Publication, 1988.
[20] D. Sherali, J. Driscoll, Evolution and state-of-the-art in integer programing, Journal of Computational and Applied Mathematics 124 (2000) 319–340.