Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research 165 (2005) 70–81

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Discrete Optimization

Improved results on the 0–1 multidimensional knapsack problem

Michel Vasquez *, Yannick Vimont

URC, École des Mines d'Alès—Commissariat à l'Énergie Atomique, site EERIE, Parc Scientifique Georges Besse, F-30035 Nîmes Cedex 1, France

Received 8 January 2003; accepted 9 January 2004
Available online 12 March 2004

Abstract

Geometric Constraint and Cutting planes have been successfully used to solve the 0–1 multidimensional knapsack problem. Our algorithm combines Linear Programming with an efficient tabu search. It gives best results when compared with other algorithms on benchmarks issued from the OR-Library. Embedding this algorithm in a variables fixing heuristic still improves our previous results. Furthermore difficult sub problems with about 100 variables issued from the 500 original ones could be generated. These small sub problems are always very hard to solve.

Keywords: Variables fixing; Heuristics; Linear programming; Tabu search

1. Introduction

The 0–1 multidimensional knapsack problem (01MDK) is a NP-hard problem which arises in several practical problems such as the capital budgeting problem, cargo loading [8,16], cutting stock problem, and computing processors allocation in huge distributed systems. It can be stated as follows:

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

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

A specific case of the 01MDK problem is the classical knapsack problem ($m=1$), which has been given much attention in the literature [14] though it is not, in fact, as difficult as 01MDK: more precisely, it can be solved in a pseudo-polynomial time. Much research has been conducted around the 01MDK problem. Exact solving of the problem is usually considered in a branch and bound framework, and some schemes were designed to provide good lower and upper bounds to solve the problem optimally. Shih [16] found an upper bound by solving m single constrained knapsack problems, and was able to solve to optimality randomly generated instances with up

* Corresponding author. Fax: +33-4-66387074.
E-mail addresses: vasquez@site-eerie.ema.fr (M. Vasquez), vimont@site-eerie.ema.fr (Y. Vimont).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2004.01.024

to five knapsack constraints and ninety variables. Gavish and Pirkul [9] improved significantly these results in a branch and bound framework by using different relaxation techniques and were able to solve instances up to 5 constraints and 200 variables.

Due to the intrinsic difficulty (NP-hardness) of the 01MDK problem, which leads to intractable computation time for larger instances, several heuristics have been used to solve it, including simulated annealing [5], tabu search [11,12], and genetic algorithms [2]. Large instances ($n = 500$, $m = 30$ from OR-Library) have thus been tackled successfully (i.e. very good lower bounds were obtained). The best known results within these benchmarks were detailed in [17–19]. They were resulting from an hybrid algorithm which used global information (the fractional solution to the linear relaxation of the problem) to guide a *tabu search* phase. The significance of the cut $\sum_1^n x_j = k$ with k integer was also emphasized. Starting from this point, we try to intensify the local search around best promising zones in order to produce better lower bounds. To do so, we have combined this previous algorithm with a problem reduction technique. This requires to select and fix some variables by using "good" points in the search space. Our heuristic is efficient and is able to improve again our previous results [18,19] on the OR-Library benchmarks.

In Section 2, we remind main principles on which our heuristic is based. Section 3 gives a thorough justification of the rule we used to choose the order in which hyperplanes $\sum_1^n x_j = k$ are examined. In Section 4, we present the variables fixing heuristic technique called *limited branch and bound*. In Section 5, we present experimental results on large instances ($n = 500$ and $m = 30$). Then, we conclude by highlighting one weakness about our algorithm, and consider further work to be conducted.

2. Previous works reminding

The main idea of our previous algorithm [17–19] is to search around a fractional optimal solution with additional constraints. Starting from the obviousness that each 01MDK solution verifies the following property: $1 \cdot x = \sum_1^n x_j = k \in \mathbb{N}$ where k is an integer. Adding this constraint to the fractional relaxed 01MDK, we obtain a series of problems such as

$$01\text{MDK}(k)\begin{cases}\text{maximize } c \cdot x \text{ s.t.}\\ A \cdot x \leqslant b \text{ and } x \in [0-1]^n \text{ and}\\ 1 \cdot x = k \in \mathbb{N}.\end{cases}$$

Several authors used a key parameter called the "number of items", for solving the simple knapsack problem [14], and for bounding the number of items at optima [6], but not directly for solving the 01MDK as we intend to do it. The bounding values of k: $k_{\min}$ and $k_{\max}$, are computed by using again two other linear problems: $k_{\min}$ is the nearest integer greater or equal to the optimal value of the linear program:

$$\begin{cases}\text{minimize } \sum_i x_i \text{ s.t.}\\ A \cdot x \leqslant b \text{ and } c \cdot x \geqslant (z^* + 1) \text{ and } x \in [0-1]^n\end{cases}$$

and $k_{\max}$ is the nearest integer lesser or equal to the optimal value of the following linear program:

$$\begin{cases}\text{maximize } \sum_i x_i \text{ s.t.}\\ A \cdot x \leqslant b \text{ and } c \cdot x \geqslant (z^* + 1) \text{ and } x \in [0-1]^n.\end{cases}$$

The $1 + k_{\max} - k_{\min}$ fractional optima $\bar{x}_{[k]}$ are considered as promising points around which a tabu search is carried out. Then, to take into account the information provided by these optimal points, but also to reduce the search space which is explored by our tabu algorithm, we have to consider an additional geometric constraint to the neighborhood rule: local search is limited to a *sphere* of fixed radius around the point $\bar{x}_{[k]}$ which is the optimal solution of the fractional relaxed 01MDK(k). In summary, each x binary configuration reached by our local search process verifies the two following constraints:

1. $1 \cdot x = k$,
2. $|x, \bar{x}_{[k]}| = \sum_1^n |x_j - \bar{x}_{[k]j}| \leqslant \delta_{\max}$.

Hence a move consists in adding 1 item which was not into the knapsack and, simultaneously, dropping 1 item which was already into the knapscak. Such a move concerns 2 different items also called attributes.

An important detail of our tabu search algorithm is the tabu list implementation. Widely inspired by the works of Glover [10] and Dammeyer and Voß [4], we have implemented a dynamic tabu list management system by using a reverse elimination method. It consists in storing the attributes (pair of components) of all completed moves in a *running list*. Stating if a move is forbidden or not needs to trace back the running list. Doing so, one builds another list, the so-called residual cancellation sequence RCS in which attributes are either added, if they are not yet in this RCS, or dropped otherwise. The condition RCS $= \emptyset$ corresponds to a move leading to an already visited configuration. For more details on this method see [4,10,17–19]. The important characteristics, that must be underlined here for further understanding, is the one related to the size of the running list, for our 2-flips move; it is twice the maximum number of iterations allowed to the search process, and, each time the tabu search improves strictly the best found configuration, the running list is cleared.

3. Ordering the exploration of the hyperplanes

There is a drawback to examine the hyperplanes from $k_{\min}$ to $k_{\max}$ like it is pointed out in Section 2. Let x^* be the best local optimum found so far and $\bar{z}_{[k]}$ be the value of the solution of 01MDK(k) (i.e. $\bar{z}_{[k]} = c \cdot \bar{x}_{[k]}$). If it occurs that $\bar{z}_{[k]} \leqslant c \cdot x^*$ for x^* binary, $1 \cdot x^* = k' > k$, then exploring the hyperplanes $1 \cdot x = k$ for $k < k'$ is useless and a waste of CPU time (Fig. 1).

In this section we present a better ordering built on the fact that the function $\bar{z}(\mu) = c \cdot \bar{x}_\mu$ (optimum of 01MDK (μ) where $\mu \in \mathbb{R}$) is convex. This property is directly proved by applying the theorem 10.2 of the book Linear Programming [3] to this problem. Hence the curve drawn by the optima values $\bar{z}_{[k]}$ of 01MDK(k) for successive integer values $k, k+1, k+2, \ldots$ has the following shape (Fig. 1).

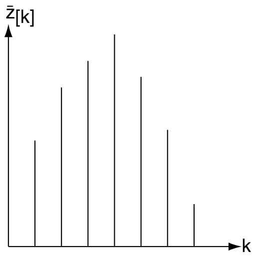

Fig. 1. "Convex" shape of $\bar{z}_{[k]}$.

A more realistic curve is given in Section 5.2 (Fig. 2).

From the convexity of the function $\bar{z}(\mu)$, we can deduce that the rounding value $[1 \cdot \bar{x}] = k_0$ (where $c \cdot \bar{x} = \max c \cdot x$ s.t. $A \cdot x \leqslant b,\ x \in [0-1]^n$) is the abscissa of one of the 2 highest bars. Starting from k_0 and scanning iteratively on the right and the left of this value ($k_0 - 1, k_0 + 1, k_0 - 2, k_0 + 2, \ldots$), our algorithm explores the hyperplanes $1 \cdot x = k$. Then, the process stops as soon as it meets the condition $\bar{z}_{[k]} \leqslant c \cdot x^*$ where x^* is the best integer configuration produced by the tabu search optimisation step. This ensures that we only explore the hyperplanes that are comprised between $k_{\min}$ and $k_{\max}$ (defined in Section 2).

4. Variables fixing

When we studied families of "good" solutions (obtained by any kind of means) to a given instance of the 01MDK problem, we were stricken to notice that many variables were fixed either to 0 or to 1 in all solutions. See also [7,15] for more recent works. Variable fixing heuristic using the notion of strongly determined variables is also described in [10].

That led us to the following scheme: for a given instance of the problem, generate several "good" solutions (i.e. near enough to the best known lower bound). For each variable x_j, fix it to zero if it is set to zero in all "good" solutions, fix it to 1 if it is set to 1 in all "good" solutions, let it free otherwise. It yields a reduced (i.e. with less variables) problem, which can then be tackled more efficiently by exact or heuristic methods. In a first approach, it seemed fairly natural to use a population based heuristic [2] to generate the set of "good" solutions and we chose the cooperative simulated annealing (COSA) heuristic (see [20] for a detailed description of the algorithm). Though our experimentation with

COSA was not exhaustive, preliminary results were deceiving: the populations were expensive (computationally speaking) to generate, and after applying tabu search as in [17–19] to the reduced problem, we didn't get any improvement on the lower bound obtained before.

When we stopped thinking of those results, it seemed that, though the basic idea (fixing variables using the characteristics of a set of "good" solutions) was sound, we had lost sight of the guidelines of our previous work:

(1) Working on 01MDK(k) instances for $k_{\min} \leqslant k \leqslant k_{\max}$;
(2) Using the information contained in the fractional optimum of these problems.

Bearing that in mind, we devise the following algorithm:

(1) let us start from the problem 01MDK(k);
(2) let $\bar{x}_{[k]}$ and $\bar{z}_{[k]} = c \cdot \bar{x}_{[k]}$ be the solution to this problem. If $\bar{x}_{[k]}$ is integer (i.e. $\bar{x}_{[k]j} = 0$ or 1 for $1 \leqslant j \leqslant n$), the problem is solved. Otherwise, let
$\mathscr{F}_k = \{j, 1 \leqslant j \leqslant n \text{ and } 0 < \bar{x}_{[k]j} < 1\}$;
(3) let j_0 be the most fractional element of $\mathscr{F}_k$ (which minimizes $|0.5 - \bar{x}_{[k]j}|$);
(4) we then consider recursively the two problems:

$$\begin{cases} \text{maximize } c \cdot x \\ A \cdot x \leqslant b \\ 1 \cdot x = k \\ x_{j_0} = 1 \end{cases} \quad \text{and} \quad \begin{cases} \text{maximize } c \cdot x \\ A \cdot x \leqslant b, \\ 1 \cdot x = k, \\ x_{j_0} = 0. \end{cases}$$

In other words, we begin seeing what a branch and bound tree looks like, branching on the most fractional variable, as it is often recommended. By applying recursively the same process d times to those two new problems, we are able to generate 2^{d+1} problems, obtained by fixing conveniently chosen variables of the original problem either to zero or to one. We call this procedure *limited branch and bound*.

We then fix the variables of the original 01MDK(k) problem according to the following rule:

- let x_j be fixed to zero (resp. one) if and only if it is equal to zero (resp. one) in all the fractional optima of the problems generated above;
- let x_j be free otherwise.

In short we have used many fractional optima as "good" points that share interesting information for the variables fixing heuristic. We will show in the following section that these points are more promising than the local optima given by another heuristic (*like those evoked just above*).

In our experimental study, we use a slight variation of the above *limited branch and bound* algorithm: at each level, instead of selecting the most fractional variable, we can select the $w \leqslant |\mathscr{F}_k|$ most fractional variables. With only one *simplex* calculation, we are thus able to generate 2^w different elements. This leads to great savings in terms of simplex computations. Practically we have used $w = 4$ for the width parameter value of our limited branch and bound and $d = 2$ for the depth parameter. Hence we obtained 256 fractional points for each 01MDK(k).

We used 256 points to determine the variables to be fixed. Obviously, using more points would lessen the risk of fixing a variable at a wrong value but the size of the induced sub problem (and thus the computational cost) would be greater.

5. Computational results

The experimental phase of this study concerns the 90 largest 01MDK benchmarks of the OR-Library. The choice of these problems was driven more by the concern of comparing our computational results to results already published, than by their effective hardness. Indeed these instances are available at http://mscmga.ms.ic.ac.uk and results have been published by Chu and Beasley [2]. They were generated using the procedure proposed by Fréville and Plateau [7]. This one was intended to create more difficult instances. People interested in such techniques may refer to [13] about a study of the effects of coefficient correlation structure. First part of this section details the main results obtained when exploring 5 hyperplanes for each problem. Then, in a second part, we give some information about what is happening when exploring more hyperplanes. Eventually we give a synthesis on the quality we can get and the cost (in time) of this approach.

5.1. Main testing bench

The testing programme is the following:

(1) solve with the simplex the problem $\max cx\ s\cdot t\cdot\ Ax \leqslant b,\ 0 \leqslant x_i \leqslant 1$ fractional: that gives us $\bar{x}$ and then $k^0 = [\sum_1^n \bar{x}_i]$ (*the rounded value of the fractional optimum components sum*);
(2) for each of the 5 integer values $k^0 - 2 \leqslant k \leqslant k^0 + 2$ solve the problem $\max cx$ s.t. $Ax \leqslant b$, $\sum_1^n x_i = k,\ 0 \leqslant x_i \leqslant 1$ fractional: that gives us 5 root node points $\bar{x}_{[k]}$ for the limited branch and bound algorithm;
(3) from each of these 5 points compute 256 fractional optima by separating on the $w = 4$ most fractional basis variables twice from the root node $\bar{x}_{[k]}$. That needs 1+16+256 = 273 simplex runs per point;
(4) fix the common 0 and 1 variables of each of the 5×256 *leave* points;
(5) use tabu search to find a *good* solution for each of these 5 hyperplanes. In that purpose, 30 values of the radius $\delta_{\max}$ are tried ($\delta_{\max} = \frac{m+i}{2}$, $i \in [0 \cdot\cdot 29]$). This radius represents the *geometric constraint* that controls the exploration of the search space around $\bar{x}_{[k]}$. For each of these radius values we use 30 random seeds (*seed* $\in [0 \cdot\cdot 29]$) of the standard *srand( )* C function. At last, the *running list* size (*used for the dynamic tabu list management*) is fixed to 100,000. This means that the maximum number of moves without improvement is 50,000.

In short each benchmark of the OR-Library needs 4500 runs of tabu search during an average of 100,000 moves. Note that the fractional point around which the search is completed corresponds to the projection of $\bar{x}_{[k]}$ onto the sub space defined by the remaining (*not fixed*) variables. Heuristic solving (tabu search), linear programming (*simplex*) and enumeration (limited branch and bound) procedures have been coded in C programming language. Our algorithm has been executed on a P4 2 GHz computer. The three following tables give the detailed results obtained by our approach. The description of the data, per column is:

- Pb: row number r of the instance. The whole name of the problem is $cbm \cdot n\_r$ where m is the number of constraints and n the number of items. Each set of 30 instances is divided into 3 series with tightness ratio $\alpha = b_i / \sum_{j=1}^n A_{ij} = 1/4$ for $0 \leqslant r \leqslant 9$, $\alpha = 1/2$ for $10 \leqslant r \leqslant 19$ and $\alpha = 3/4$ for $20 \leqslant r \leqslant 29$;
- $\bar{z}$: the optimum value of the integrity relaxed version of the original 01MDK;
- $\mathrm{GA_{CB}}$: results obtained by Chu and Beasley [2] with their Genetic Algorithm;
- LP + TS: results obtained by the first version of our algorithm [17,18];
- *Fix* + LP + TS: new results;
- z^*: best integral objective value found by each of the three methods;
- k^*: number of items in the best solution found by our algorithms;
- dist*: distance $\sum_1^n |x_i^* - \bar{x}_{[k^*]i}|$ between the best solution x^* and the fractional optimum $\bar{x}_{[k^*]}$ of 01MDK in the hyperplane $\sum_1^n x_i = k$;
- t^*: time in seconds to find the best solution for *Fix* + LP + TS;
- *rk*: exploring order of the hyperplane $1 \cdot x = k^*$ (*as defined in Section 3*);
- #0: number of variables fixed at the value 0;
- #1: number of variables fixed at the value 1;
- κ^*: number of items to be chosen in the sub problem divided by the total number of remaining variables: $\kappa^* = (k^* - (\#1))/(500 - (\#0 + \#1))$.

Bold face text highlights the first best values found by one of these three compared methods.

We begin by the problems that have most constraints. They have the heaviest cost for the neighborhood evaluation of the tabu search. Fixing variables is considered as a way to decrease the computational complexity and so to have more time to find better solutions. Moreover, for these 30 instances, given the upper bounds and the best known lower bounds, it is not possible to fix variables using the reduced costs (as shown in [1]

Table 1
New results on **cb30.500** problems

Pb.	GA_{CB}		LP + TS				*Fix* + LP + TS						
	$\bar{z}$	z^*	z^*	k^*	dist*	z^*	t^*	k^*	rk	dist*	#0	#1	κ^*
0	116,619.0	115,868	115,991	130	18.61	**116,056**	15,771	130	0	18.21	288	47	0.50
1	115,370.1	114,667	**114,810**	128	17.56	114,810	120,414	128	1	17.56	278	46	0.47
2	117,342.5	116,661	116,683	128	17.72	**116,712**	121,769	128	1	16.02	270	41	0.46
3	115,946.4	115,237	115,301	128	19.38	**115329**	85,670	127	1	19.02	274	50	0.44
4	117,079.3	116,353	116,435	127	22.04	**116525**	603	129	0	17.67	304	64	0.49
5	116,377.6	115,604	115,694	131	17.54	**115,741**	616	131	0	17.59	277	45	0.48
6	114,689.7	113,952	114,003	128	23.28	**114,181**	110,873	128	1	16.14	280	48	0.47
7	114,847.8	114,199	114,213	129	21.43	**114,348**	282,523	128	2	19.00	290	59	0.46
8	115,902.6	115,247	115,288	127	18.78	**115,419**	112,849	128	1	15.18	281	48	0.47
9	117,668.8	116,947	117,055	129	18.21	**117,116**	121,248	128	1	20.63	281	55	0.45
10	218,601.5	217,995	218,068	251	13.03	**218,104**	96,952	251	0	18.27	161	162	0.50
11	215,074.7	214,534	214,562	251	18.74	**214,648**	167,224	251	1	22.11	160	166	0.49
12	216,401.1	215,854	215,903	250	19.48	**215,978**	178,701	250	1	22.78	164	172	0.48
13	218,350.5	217,836	**217,910**	251	14.14	217,910	308,469	251	3	14.14	161	167	0.49
14	216,094.5	215,566	215,596	251	15.22	**215,689**	144,793	251	1	20.65	164	169	0.49
15	216,327.4	215,762	215,842	253	17.64	**215,890**	117,102	253	1	17.76	172	188	0.46
16	216,376.3	215,772	215,838	252	13.04	**215,907**	1637	253	0	18.78	141	159	0.47
17	217,014.1	216,336	216,419	253	13.90	**216,542**	4775	254	0	19.88	160	162	0.52
18	217,839.2	217,290	217,305	253	19.72	**217340**	4742	253	0	17.13	162	166	0.51
19	215,218.5	214,624	214,671	252	17.49	**214739**	109,785	252	1	17.86	163	167	0.50
20	302,038.8	301,627	301,643	375	11.77	**301,675**	143,430	375	1	16.21	45	301	0.48
21	300,455.0	299,985	**300,055**	374	17.36	300,055	218,994	374	2	17.36	53	289	0.54
22	305,501.2	304,995	305,028	375	14.25	**305,087**	118,929	375	1	16.70	44	292	0.51
23	302,456.2	301,935	302,004	375	16.38	**302,032**	156,377	375	1	21.43	48	293	0.52
24	304,901.4	304,404	304,411	376	13.69	**304,462**	238,811	375	2	18.66	44	279	0.54
25	297,409.4	296,894	296,961	374	12.48	**297,012**	12,191	374	0	15.91	48	288	0.52
26	303,765.9	303,233	303,328	373	15.35	**303,364**	213,316	373	2	15.78	55	290	0.54
27	307,402.5	306,944	306,999	376	17.07	**307,007**	45,781	377	0	17.08	38	289	0.51
28	303,605.9	303,057	303,080	374	12.94	**303,199**	235,005	375	2	21.05	52	290	0.54
29	301,020.6	300,460	300,532	376	14.49	**300,572**	82,949	376	0	23.35	50	284	0.55

for instance). Most of our previous results have been improved. Including the whole optimization process, it takes an average of 100 hours for 4500 runs per benchmark i.e. about 80 seconds by run. Note that, as we will illustrate it in Section 5.2, roughly the same results would be obtained with only 10 random seeds rather than 30: that would divide computing time by 3. The variable fixing process spends an average of 20 seconds per benchmark. The size of the remaining sub problems is $n \simeq 167$ variables. Although we did not investigate it exhaustively, we could not solve these sub problems with CPLEX7.0 running on a SunBlade1000 with 2 GBytes memory.

For these 30 problems with 10 constraints, the results are a little less satisfactory. Indeed four results are not improved (compared to three in the previous table), and the result for cb10.500_14 is worse. There is one wrongly fixed variable regarding the previous solution x^* obtained without fixing variables. In that case tabu search is not able to find this solution. Of course this is not a sufficient explanation of this bad result since the tabu search procedure is not a complete one. Nevertheless the majority of results are improved. The whole optimization process takes an average of 70 hours per benchmark. The remaining sub problems have a size of $n \simeq 95$ variables. Again, we have not been able to solve these small sub problems to optimality.

For these last 5 constraint problems we have still obtained less improvement. We have also four

Table 2
New results on **cb10.500** problems

Pb.	GA_{CB}		LP + TS				*Fix* + LP + TS						
	$\bar{z}$	z^*	z^*	k^*	dist*	z^*	t^*	k^*	*rk*	dist*	#0	#1	κ^*
0	118,019.5	117,726	117,779	134	12.96	**117,811**	21,845	135	0	12.77	317	98	0.44
1	119,437.3	119,139	119,190	134	11.04	**119,232**	7163	135	0	13.01	318	88	0.50
2	119,405.7	119,159	119,194	135	11.48	**119,215**	19205	136	0	13.32	317	97	0.45
3	119,066.1	118,802	**118,813**	137	8.49	118,813	288	137	0	8.49	305	90	0.45
4	116,698.0	116,434	116,462	134	4.59	**116,509**	58,353	136	1	11.70	322	94	0.50
5	119,710.0	119,454	**119,504**	137	11.19	119,504	18,720	137	0	11.19	316	90	0.50
6	120,033.3	119,749	119,782	139	11.89	**119,827**	82,719	138	1	14.42	299	94	0.41
7	118,545.7	118,288	118,307	135	10.26	**118,329**	98,291	135	1	13.93	316	83	0.51
8	118,001.6	117,779	117,781	136	12.42	**117,815**	12,558	136	0	10.67	314	84	0.51
9	119,440.6	119,125	119,186	138	8.40	**119,231**	30,501	138	0	14.59	319	88	0.54
10	217,552.9	217,318	217,343	256	9.01	**217,377**	64,165	256	1	13.41	193	216	0.44
11	219,255.2	219,022	219,036	259	8.08	**219,077**	750	259	0	12.06	197	216	0.49
12	217,987.8	217,772	217,797	256	12.07	**217,806**	99,480	256	1	15.67	197	217	0.45
13	217,040.7	216,802	216,836	258	8.34	**216,868**	476	259	0	10.34	200	215	0.52
14	214,010.3	213,809	**213,859**	256	7.83	213,850	9252	257	0	11.49	192	212	0.47
15	215,261.3	215,013	215,034	257	8.88	**215,086**	4953	257	0	10.76	193	203	0.52
16	218,109.2	217,896	217,903	261	9.89	**217,940**	41,803	260	0	14.53	190	213	0.48
17	220,175.6	219,949	219,965	256	9.79	**219,984**	646	257	0	11.11	200	208	0.53
18	214,561.0	214,332	214,341	258	8.75	**214,375**	6978	257	0	14.40	195	214	0.47
19	221,083.6	22,0833	22,0865	255	9.83	**220,899**	34,838	254	0	14.76	194	207	0.47
20	304,555.0	304,344	304,351	378	9.60	**304,387**	830	379	0	12.69	71	334	0.47
21	302,553.0	302,332	302,333	380	8.94	**302,379**	59,226	380	1	14.01	69	324	0.52
22	302,581.5	302,354	302,408	379	7.80	**302,416**	3235	380	0	12.29	78	333	0.53
23	300,956.7	300,743	**300,757**	378	10.79	300,757	569	379	0	10.49	67	331	0.47
24	304,584.7	304,344	304,344	381	7.50	**304,374**	17,251	380	0	12.33	75	332	0.52
25	301,952.5	301,730	301,754	375	7.73	**301,836**	67,073	375	1	13.27	81	333	0.49
26	305,139.7	304,949	304,949	378	10.83	**304,952**	27,970	378	0	15.24	72	326	0.51
27	296,636.6	296,437	296,441	379	10.01	**296,478**	22,862	380	0	13.38	61	334	0.44
28	301,547.6	301,313	301,331	379	12.39	**301,359**	11,508	379	0	12.36	74	332	0.50
29	307,250.0	307,014	307,078	378	13.90	**307,089**	1037	378	0	14.87	82	328	0.56

bad results on the instances CB5.500_5, 10, 14 and 15. Once again, we have checked with the solutions coming from the LP + TS algorithm and have found one mistakenly fixed variable for each of these four problems. However, the average improvement value is positive. Each problem required about 50 hours CPU but the solutions were found in an average time of 8.5 hours. Remaining sub problems have an average size of $n = 65$ variables. Using CPLEX7.0, we find optimal integer solutions for most of them. That shows, firstly, that our tabu search algorithm has also produced the best possible results, and secondly, that there are mistakenly fixed variables for the problems 5, 10, 14 and 15. Tables 1–3 show that the best solutions found by our algorithm are almost always (89 times out of 90) located in the first three hyperplanes (see column *rk*). A more detailed study on this topic is conducted in the next sub section.

5.2. Exploring more hyperplanes

In this section, we give the distribution of the best values $z^*_{[k]}$ found when exploring nine hyperplanes rather than five. Table 4 details also the $z^*_{[k]}$ obtained in each hyperplane $1 \cdot x = k$, considering separately the intervals $[0 \cdots 9]$, $[10 \cdots 19]$ and $[20 \cdots 29]$ of the random seed values, in order to highlight the behavior of the algorithm regarding randomness. This experimentation is carried out over nine problems.

$ub_{[k]}$ represents the upper bound computed from the 256 fractional optima of the hyperplane

Table 3
New results on **cb5.500** problems

Pb.	GA_{CB}		LP + TS				*Fix* + LP + TS						
	$\bar{z}$	z^*	z^*	k^*	dist*	z^*	t^*	k^*	*rk*	dist*	#0	#1	κ^*
0	120,234.9	120,130	120,134	146	9.10	**120,148**	89,256	147	2	7.90	322	119	0.47
1	117,955.2	117,837	117,864	148	8.21	**117,879**	7218	148	0	10.29	317	114	0.49
2	121,213.3	121,109	121,112	145	9.58	**121,131**	18,336	144	0	10.63	327	114	0.51
3	120,888.5	120,798	**120,804**	149	10.50	120,804	566	149	0	10.50	310	122	0.40
4	122,426.5	**122,319**	122,319	147	6.90	122,319	117	147	0	6.90	314	112	0.47
5	122,126.0	122,007	**122,024**	153	7.14	122,011	221,029	151	3	12.63	313	121	0.45
6	119,218.8	119,113	**119,127**	145	10.71	119,127	598	145	0	10.71	320	114	0.47
7	120,643.1	**120,568**	120,568	150	7.47	120,568	146	150	0	7.47	319	122	0.47
8	121,663.3	121,575	121,575	148	9.68	121,575	56,360	148	1	9.68	307	121	0.38
9	120,800.7	120,699	120,707	151	11.63	**120,717**	81,062	151	2	11.74	312	119	0.46
10	218,500.1	218,422	**218,428**	267	9.26	218,426	25	267	0	5.80	202	239	0.47
11	221,272.4	221,191	221,191	265	6.98	**221,202**	36,742	265	0	11.71	202	233	0.49
12	217,615.8	217,534	217,534	264	6.04	**217,542**	30,597	264	0	9.22	202	237	0.44
13	223,653.2	223,558	223,558	264	7.97	**223,560**	47,175	263	1	12.00	202	233	0.46
14	219,067.5	218,962	**218,966**	267	8.56	218,965	305	267	0	8.26	196	232	0.49
15	220,617.0	220,514	**220,530**	262	7.76	220,527	617	261	0	11.46	211	225	0.56
16	220,076.6	219,987	**219,989**	266	7.45	219,989	164	266	0	7.45	201	235	0.48
17	218,282.7	218,194	218,194	266	8.00	**218,215**	40,799	265	1	7.99	205	235	0.50
18	217,059.9	**216,976**	216,976	262	7.26	216,976	138	262	0	7.26	207	223	0.56
19	219,812.8	219,693	219,704	267	7.23	**219,719**	42,149	267	1	9.23	196	237	0.45
20	295,896.4	**295,828**	295,828	383	7.17	295,828	45,199	383	1	7.17	78	346	0.49
21	308,157.6	308,077	308,083	383	6.39	**308,086**	253	384	0	7.98	85	357	0.47
22	299,878.6	**299,796**	299,796	385	8.33	299,796	299	385	0	8.33	82	348	0.53
23	306,554.1	306,476	306,478	385	11.26	**306,480**	344	384	0	8.66	85	355	0.48
24	300,412.6	**300,342**	300,342	385	8.43	300,342	39,613	385	1	8.43	86	357	0.49
25	302,661.8	302,560	302,561	384	7.29	**302,571**	457	385	0	9.14	81	353	0.48
26	301,400.3	301,322	301,329	385	9.93	**301,339**	542	385	0	10.65	85	353	0.52
27	306,517.3	306,430	**306,454**	383	8.63	306,454	370	383	0	8.63	75	350	0.44
28	302,896.8	302,814	302,822	382	8.75	**302,828**	84,497	384	2	8.72	84	350	0.52
29	299,973.7	299,904	299,904	379	4.24	**299,910**	77,254	378	2	11.06	88	355	0.40

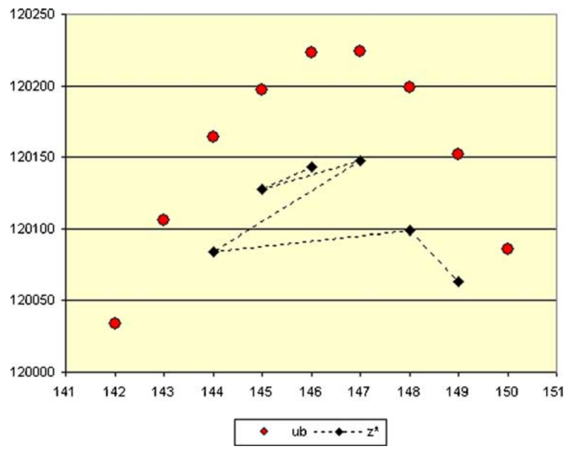

Fig. 2. Shapes of $ub_{[k]}$ and $z^*_{[k]}$ for CB5.500_0.

$1 \cdot x = k$; note that this upper bound is tighter than $\bar{z}$. This table shows that the best values are found in the first three hyperplanes and also that, for these nine instances, trying 30 random seed values rather than 10 does not improve the solution significantly.

The Fig. 2 illustrates the evolution of the $ub_{[k]}$ and $z^*_{[k]}$ values during the whole optimization algorithm. We can see that the hyperplanes $k = 142$, $k = 143$ and $k = 150$ are not candidate for the local search phase of our algorithm because $ub_{[k]} < z^*_{[147]}$. The points corresponding to z^* are linked by a dashed line. That permits to show in what order the hyperplanes (from $k = 146$ to $k = 149$) are explored.

Table 4
Results distribution over 9 hyperplanes and 3×10 random seeds

cb5.500_0					**cb5.500_10**					**cb5.500_20**				
k	$ub_{[k]}$	$0\cdots9$	$10\cdots19$	$20\cdots29$	k	$ub_{[k]}$	$0\cdots9$	$10\cdots19$	$20\cdots29$	k	$ub_{[k]}$	$0\cdots9$	$10\cdots19$	$20\cdots29$
146	120,223	120,138	120,138	120,143	**267**	218,493	**218,426**	**218,426**	**218,426**	384	295,890	295,815	295,815	295,815
145	120,197	120,117	120,117	120,128	266	218,482	218,422	218,422	218,422	**383**	295,885	**295,828**	**295,828**	**295,828**
147	120,224	**120,148**	**120,148**	**120,148**	268	218,490	218,421	218,421	218,421	385	295,878	295,801	295,801	295,801
144	120,164	120,084	120,084	120,084	265	218,461	218,383	218,383	218,383	382	295,869	295,799	295,799	295,799
148	120,199	120,099	120,099	120,099	269	218,474	218,416	218,416	218,416	386	295,859	295,804	295,804	295,804
143	120,106	$ub_{[k]} \leqslant z^*$			264	218,415	$ub_{[k]} \leqslant z^*$			381	295,838	295,766	295,766	295,766
149	120,152	120,063	120,063	120,063	270	218,447	218,383	218,383	218,383	387	295,832	295,748	295,748	295,748
142	120,034	$ub_{[k]} \leqslant z^*$			263	218,360	$ub_{[k]} \leqslant z^*$			380	295,784	$ub_{[k]} \leqslant z^*$		
150	120,086	$ub_{[k]} \leqslant z^*$			271	218,408	$ub_{[k]} \leqslant z^*$			388	295,791	$ub_{[k]} \leqslant z^*$		
cb10.500_0					**cb10.500_10**					**cb10.500_20**				
135	118,008	117,809	**117,811**	117,809	257	217,540	217,367	217,353	217,346	**379**	304,544	**304,387**	**304,387**	**304,387**
134	117,996	117,790	117,790	117,790	**256**	217,526	**217,377**	217,357	**217,377**	378	304,535	304,363	304,363	304,363
136	117,987	117,809	117,809	117,809	258	217,516	217,335	217,347	217,350	380	304,526	304,350	304,350	304,350
133	117,944	117,776	117,776	117,776	255	217,481	217,335	217,335	217,335	377	304,502	304,333	304,333	304,333
137	117,932	117,743	117,736	117,743	259	217,471	217,299	217,286	217,299	381	304,491	304,284	304,284	304,284
132	117,844	117,641	117,641	117,641	254	217,420	217,259	217,280	217,280	376	304,415	304,279	304,279	304,279
138	117,855	117,666	117,666	117,666	260	217,406	217,233	217,225	217,230	382	304,413	304,249	304,249	304,249
131	117,708	$ub_{[k]} \leqslant z^*$			253	217,337	$ub_{[k]} \leqslant z^*$			375	304,309	$ub_{[k]} \leqslant z^*$		
139	117,759	$ub_{[k]} \leqslant z^*$			261	217,321	$ub_{[k]} \leqslant z^*$			383	304,301	$ub_{[k]} \leqslant z^*$		
cb30.500_0					**cb30.500_10**					**cb30.500_20**				
130	116,572	**116,056**	**116,056**	**116,056**	**251**	218,563	218,075	**218,104**	**218,104**	376	302,021	301,605	301,615	301,611
129	116,466	115,981	115,968	115,968	250	218,499	218,078	218,081	218,072	**375**	301,989	**301,675**	**301,675**	**301,675**
131	116,565	115,927	115,952	115,927	252	218,545	218,088	218,088	218,088	377	301,953	301,502	301,502	301,524
128	116,230	115,830	115,868	115,825	249	218,288	217,950	217,950	217,950	374	301,871	301,623	301,623	301,623
132	116,451	115,736	115,736	115,746	253	218,439	217,846	217,874	217,873	378	301,786	301,273	301,249	301,249
127	115,863	$ub_{[k]} \leqslant z^*$			248	217,967	$ub_{[k]} \leqslant z^*$			373	301,688	301,454	301,454	301,454
133	116,211	115,495	115,648	115,648	254	218,258	217,688	217,688	217,688	379	301,546	$ub_{[k]} \leqslant z^*$		
126	115,415	$ub_{[k]} \leqslant z^*$			247	217,577	$ub_{[k]} \leqslant z^*$			372	301,448	$ub_{[k]} \leqslant z^*$		
134	115,833	$ub_{[k]} \leqslant z^*$			255	218,030	$ub_{[k]} \leqslant z^*$			380	301,220	$ub_{[k]} \leqslant z^*$		

5.3. Results synthesis

Our approach is able to improve many lower bounds on the OR-LIBRARY 500 variables benchmarks. Some result performances can not be guaranteed, however this is not the goal of a heuristic approach. Objective of such a work is two fold; first it provides better lower bounds,very useful in the framework of an exact method; then, it gives practical "good" solutions in a real world problems.

For estimating the benefit provided by our algorithm, Table 5 compares, for each subset of 10 instances, the averages given by different approaches; it combines best results obtained by Chu and Beasley (column GA_{CB}) [2], those obtained by Osorio, Glover and Hammer [15] (columns *Fix* + Cuts and CPLEX), those produced by our first tabu search [17–19] (column LP + TS) and those obtained by this last work (column *Fix* + LP + TS).

Columns $\bar{t}^*$ give the average time (in hours) required to reach the best solutions. This information is very approximative since the CPU's are not the same for activating separately each method. It is clear enough to point out that our testing bench requires much more CPU time resource than the others approaches. However on the one hand, we have noticed that it was possible to reduce the number of runs (*i.e. seeds values*) without degrading the solutions quality in a too significative way. On the other hand, we can try to identify in a safer way the promising hyperplanes. More precisely, the ranking order in exploring the hyperplanes presented in Section 3 does not always lead to decreasing values of the upper bounds. This is typically true for benchmarks submitted to 5 constraints for which the *best* hyperplane can be located at the third position: for example, the best solution of CB5.500_0 can be found in the hyperplane for which the upper bound is the greatest, whereas it is explored only at the third step (*see* Table 4 or Fig. 2). The time required to get this solution would have been reduced from 89256 (see Tables 3) to 191 seconds if we had searched in this hyperplane at first. Moreover, it may also explain

Table 5
Average performance over the 90 largest OR-LIBRARY problems

m	GA_{CB}			Fix + Cuts		CPLEX		LP + TS		Fix+LP+TS	
	α	z^*	$\bar{t}^*$	z^*	$\bar{t}^*$	z^*	$\bar{t}^*$	z^*	$\bar{t}^*$	z^*	$\bar{t}^*$
5	1/4	120,616	0.1	120,610	3	120,619	3	120,623	5	**120,628**	8.5
	1/2	219,503	0.1	219,504	3	219,506	3	219,507	5	**219,512**	8.5
	3/4	302,325	0.1	302,361	3	302,358	3	302,360	5	**302,363**	8.5
10	1/4	118,566	0.2	118,584	3	118,597	3	118,600	9	**118,629**	7.6
	1/2	217,275	0.2	217,297	3	217,290	3	217,298	9	**217,326**	7.6
	3/4	302,556	0.2	302,562	3	302,573	3	302,575	9	**302,603**	7.6
30	1/4	115,474	0.4	115,520	3	115,497	3	115,547	12	**115,624**	33
	1/2	216,157	0.4	216,180	3	216,151	3	216,211	12	**216,275**	33
	3/4	302,353	0.4	302,373	3	302,366	3	302,404	12	**302,447**	33

Table 6
Best average lower bounds for CB.500

α	$m = 5$			$m = 10$			$m = 30$		
	$\frac{\sum z^*}{10}$	$\frac{\sum ub}{10}$	$\frac{\sum \bar{z}}{10}$	$\frac{\sum z^*}{10}$	$\frac{\sum ub}{10}$	$\frac{\sum \bar{z}}{10}$	$\frac{\sum z^*}{10}$	$\frac{\sum ub}{10}$	$\frac{\sum \bar{z}}{10}$
1/4	120,629	120,709	120,717	118,629	118,822	118,836	115,624	116,151	116,184
1/2	219,513	219,589	219,596	217,327	217,493	217,504	216,275	216,703	216,730
3/4	302,363	302,429	302,435	302,603	302,762	302,776	302,447	302,828	302,856

the reason why average times are worse for benchmarks with 5 constraints than for those with 10 constraints.

At last, we can complete this presentation by gathering results and data in the following Table 6 intended to motivate researchers/challengers to improve these best average results (obtained by taking into account the best value in both LP + TS and *Fix* + LP + TS columns).

For each problem, the *ub* parameter represents the maximum value among the 5×256 fractional optima computed during the variables fixing process (it is also equal to the maximum of the five best values of $ub_{[k]}$ defined in Section 5.2). It is a slightly tighter value compared to the classical upper bound $\bar{z}$ of the relaxed linear program 01MDK. This table shows that it is left an average of 74 points to improve our values for the CB5.500 benchmarks, 173 points for the CB10.500 ones, and 445 for the CB30.500 ones.

All the problems and their associated best solutions presented in this paper are available in the following web site: www.01mdk.ema.fr.

6. Conclusion

In terms of qualitative results, the two aforementioned tables, clearly show the positive contribution of our heuristic approach that combines cutting planes ($1\cdot x = k$), geometric constraints ($|x - \bar{x}_{[k]}| \leqslant \delta_{\max}$) and limited branch and bound for fixing variables using fractional optima as high quality reference points.

The generated sub problems discussed in this paper are very difficult to solve even though they are small enough (about 100 variables). One of their significant characteristic is related to the ratio $\kappa = (k - (\#1))/(n - (\#0 + \#1))$ which is always close to 0.5. This value ($\kappa \simeq 0.5$) corresponds to the maximum number of sub sets of $k - (\#1)$ elements in a set of $n - (\#0 + \#1)$ elements. This means that the combinatory aspect of those sub problems is the greatest possible considering the $n - (\#0 + \#1)$ remaining variables.

Though very attractive, in terms of results, the above proposed limited branch and bound variable fixing heuristic is not fully satisfactory: it may happen (and it probably sometimes does !) that some variables are set-up at their wrong values, which prevents the search phase, that follows, to reach a good solution; we would be ready to accept a lower problem reduction (i.e. a bigger reduced problem) giving the certainty (mathematically proved) that no variable is fixed at its wrong value. That is the aim of further work we are planning to conduct.

References

[1] E. Balas, C.H. Martin, Pivot and complement a heuristic for 0–1 programming, Management Science 26 (1980) 86–96.
[2] P.C. Chu, J.E. Beasley, A genetic algorithm for the multidimensional knapsack problem, Journal of Heuristic 4 (1998) 63–86.
[3] V. Chvátal, Linear Programming, W.H. Freeman and Company, 1983.
[4] F. Dammeyer, S. Voß, Dynamic tabu list management using the reverse elimination method, Annals of Operations Research 41 (1993) 31–46.
[5] A. Drexl, A simulated annealing approach to the multiconstraint zero–one knapsack problem, Computing 40 (1988) 1–8.
[6] A. Fréville, G. Plateau, Sac à dos multidimensionnel en variable 0–1: Encadrement de la somme des variables à l'optimum, Recherche Opérationnelle 27 (2) (1993) 169–187.
[7] A. Fréville, G. Plateau, An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem, Discrete Applied Mathematics 49 (1994) 189–212.
[8] B. Gavish, H. Pirkul, Allocation of data bases and processors in a distributed computing system, Management of Distributed Data Processing 31 (1982) 215–231.
[9] B. Gavish, H. Pirkul, Efficient algorithms for solving multiconstraint zero–one knapsack problems to optimality, Mathematical Programming 31 (1985) 78–105.
[10] F. Glover, Tabu search, ORSA Journal of Computing 2 (1990) 4–32.
[11] F. Glover, G.A. Kochenberger, Critical event tabu search for multidimensional knapsack problems, in: I.H. Osman, J.P. Kelly (Eds.), Metaheuristics: The Theory and Applications, Kluwer Academic Publishers, 1996, pp. 407–427.
[12] S. Hanafi, A. Fréville, An efficient tabu search approach for the 0–1 multidimensional knapsack problem, European Journal of Operational Research 106 (1998) 659–675.
[13] R.R. Hill, C.H. Reilly, The effects of coefficient correlation structure in two dimensional knapsack problems on solution procedure performance, Management Science 46 (2) (2000) 302–317.
[14] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley, 1990.

[15] M.A. Osorio, F. Glover, P. Hammer, Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions, Technical Report HCES-08-00, Hearin Center For Enterprise Science, The University of Mississippi, August 2000.
[16] W. Shih, A branch and bound method for the multiconstraint zero–one knapsack problem, Journal of the Operational Research Society 30 (1979) 369–378.
[17] M. Vasquez, Résolution en variables 0–1 de problèmes combinatoires de grande taille par la méthode tabou, PhD thesis, Université d'Angers U.F.R. Sciences, December 2000.
[18] M. Vasquez, J.K. Hao, A hybrid approach for the 0–1 multidimensionnal knapsack problem, in: IJCAI-01 Proceedings of the 7th International Joint Conference on Artificial Intelligence, vol. 1, Seattle, Washington, USA, August 2001, pp. 328–333.
[19] M. Vasquez, J.K. Hao, Une approche hybride pour le sac à dos multidimensionnel en variables 0–1, RAIRO Operations Research 35 (December) (2001) 415–438.
[20] O. Wendt, W. König, Cooperative simulated annealing: How much cooperation is enough? Technical Report, Institute of Information Systems, Goethe University, Frankfurt, 1997.