A Hybrid Approach for the 0–1 Multidimensional Knapsack problem

Michel Vasquez 1) and Jin-Kao Hao 2)
1) EMA–EERIE, Parc Scientifique G. Besse, F–30035 Nimes cedex 1, vasquez@eerie.fr
2) Université d'Angers, 2 bd Lavoisier, F–49045 Angers cedex 1, hao@info.univ-angers.fr
to appear in the Proc. of IJCAI-01, Seatle, Washington, August 2001

Abstract

We present a hybrid approach for the 0–1 multidimensional knapsack problem. The proposed approach combines linear programming and Tabu Search. The resulting algorithm improves significantly on the best known results of a set of more than 150 benchmark instances.

1 Introduction

The NP-hard 0–1 multidimensional knapsack problem (MKP01) consists in selecting a subset of n given objects (or items) in such a way that the total profit of the selected objects is maximized while a set of knapsack constraints are satisfied. More formally, the MKP01 can be stated as follows.

$$\text{MKP01} \left\{ \begin{array}{l} \text{maximize } c.x \text{ subject to} \\ A.x \leq b \text{ and } x \in \{0,1\}^n \end{array} \right.$$

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 object j is selected, 0 otherwise. c_j is the profit associated to j. Each of the m constraints $A_i.x \leq b_i$ is called a knapsack constraint.

The special case of the MKP01 with $m = 1$ is the classical knapsack problem (KP01), whose usual statement is the following. Given a knapsack of capacity b and n objects, each being associated a profit (gain) and a volume occupation, one wants to select k ($k \leq n$ and k not fixed) objects such that the total profit is maximized and the capacity of the knapsack is not exceeded. It is well known that the KP01 is not strongly NP-hard because there are polynomial approximation algorithms to solve it. This is not the case for the general MKP01.

The MKP01 can formulate many practical problems such as capital budgeting where project j has profit c_j and consume (a_{ij}) units of resource i. The goal is to determine a subset of the n projects such that the total profit is maximized and all resource constraints are satisfied. Other important applications include cargo loading [Shih, 1979], cutting stock problems, and processor allocation in distributed systems [Gavish and Pirkul, 1982]. Notice that the MKP01 can be considered as a general 0–1 integer programming problem with non-negative coefficients.

Given the practical and theoretical importance of the MKP01, it is not surprising to find a large number of studies in the literature; we give a brief review of these studies in the next section.

2 State of the Art

Like for many NP-hard combinatorial optimization problems, both exact and heuristic algorithms have been developed for the MKP01. Existing exact algorithms are essentially based on the branch and bound method [Shih, 1979]. These algorithms are different one from another according to the way the upper bounds are obtained. For instance, in [Shih, 1979], Shih solves, exactly, each of the m single constrained, relaxed knapsack problems and select the minimum of the m objective function values as the upper bound. Better algorithms have been proposed by using tighter upper bounds, obtained with other MKP01 relaxation techniques such as lagrangean, surrogate and composite relaxations [Gavish and Pirkul, 1985]. Due to their exponential time complexity, exact algorithms are limited to small size instances ($n = 200$ and $m = 5$).

Heuristic algorithms are designed to produce near-optimal solutions for larger problem instances. The first heuristic approach for the MKP01 concerns for a large part greedy methods. These algorithms construct a solution by adding, according to a greedy criterion, one object each time into a current solution without constraint violation. The second heuristic approach is based on linear programming by solving various relaxations of the MKP01. A first possibility is to relaxe the integrality constraints and solve optimally the relaxed problem with simplex. Other possibilities include surrogate and composite relaxations [Osorio *et al.*, 2000].

More recently, several algorithms based on metaheuristics have been developed, including simulated annealing [Drexl, 1988], tabu search [Glover and Kochenberger, 1996; Hanafi and Fréville, 1998] and genetic algorithms [Chu and Beasley, 1998]. The metaheuristic approach has allowed to obtain very competitive results on large instances compared with other methods ($n = 500$ and $m = 30$).

Most of the above heuristics use the so-called *pseudo–utility* criterion for selecting the objects to be added into a solution. For the single constraint case (the KP01), this criterion corresponds to the *profit/resource* ratio. For the general MKP01, the *pseudo–utility* is usually defined as $c_j/(u.a_j)$

where u is a multiplier for the column vector x_j. Notice that it is impossible to obtain "optimal" multipliers. Thus the *pseudo–utility* criterion may mislead the search.

In this paper, we propose an alternative way to explore the search space. We use fractional optimal solutions given by linear programming to guide a neighborhood search (Tabu Search or TS) algorithm. The main idea is to exploit around a fractional optimal solution with additional constraints. We experiment this approach on a very large variety of MKP01 benchmark instances and compare our results with the best known ones. We show that this hybrid approach outperforms previously best algorithms for the set of tested instances.

The paper is organized as follows. Next section presents the general scope of our approach. Section 4 describes the algorithm used to determine *promising* sub-space of the whole search space from where we run a TS algorithm. This algorithm is presented in section 5. Finally we give computational results on a large set of MKP01 instances in section 6.

3 A Hybrid Approach for the MKP01

The basic idea of our approach is to search around the fractional optimum $\bar{x}$ of some relaxed MKP. Our hypothesis is that the search space around $\bar{x}$ should contain high quality solutions[1].

This general principle can be put into practice via a two phase approach. The first phase consists in solving exactly a relaxation MKP of the initial MKP01 to find a fractional optimal solution $\bar{x}$. The second phase consists in exploring carefully and efficiently some areas around this fractional optimal point.

Clearly, there are many possible ways to implement this general approach. For this work, we have done the following choices. The first phase is carried out by using simplex to solve a relaxed MKP. The second phase is ensured by a Tabu Search algorithm.

Let us notice first that relaxing the integrality constraints alone may not be sufficient since its optimal solution may be far away from an optimal binary solution. To be convinced, let us consider the following small example with five objects and one knapsack constraint:

$$sample \left\{ \begin{array}{lcl} c & = & (12 \quad 12 \quad 9 \quad 8 \quad 8) \\ A & = & (11 \quad 12 \quad 10 \quad 10 \quad 10) \quad b = 30 \end{array} \right.$$

This relaxed problem leads to the fractional optimal solution $\bar{x} = (1, 1, \frac{7}{10}, 0, 0)$ with an optimal cost value $\bar{z} = 30.3$ while the optimal binary solution is $x^{opt} = (0, 0, 1, 1, 1)$ with an optimal cost value $z^{opt} = 25$.

However, the above relaxation can be enforced to give more precise information regarding the discrete aspect of the problem. To do this, let us notice that all solutions of MKP01 verify the property: $\sum_{j=1}^{n} x_j = k \in \mathbb{N}$ with k an integer. Now if we add this constraint to the relaxed MKP, we obtain a series of problems:

$$\text{MKP}[k] \left\{ \begin{array}{l} \text{maximize } c.x \text{ s.t.} \\ A.x \leq b \text{ and } x \in [0\text{–}1]^n \text{ and} \\ \sigma(x) = k \in \mathbb{N} \end{array} \right.$$

where $\sigma(x)$ is the sum of the components of x, defining a hyperplane. We have thus several (promising) points $\bar{x}_{[k]}$ around which a careful search will be carried out.

To show the interest of this extra constraint, take again the previous example with $k = 1, 2, 3$, leading to three relaxed problems:

$$\begin{array}{lllll} \text{MKP}[1] & \rightarrow & \bar{z}_{[1]} = 12 & \text{et} & \bar{x}_{[1]} = (1,0,0,0,0) \\ \text{MKP}[2] & \rightarrow & \bar{z}_{[2]} = 24 & \text{et} & \bar{x}_{[2]} = (1,1,0,0,0) \\ \text{MKP}[3] & \rightarrow & \bar{z}_{[3]} = 25 & \text{et} & \bar{x}_{[3]} = (0,0,1,1,1) \end{array}$$

where $k = 3$ gives directly an optimal binary solution $x_{[3]} = x^{opt}$ without further search.

In general cases, search is necessary to explore around each fractional optimum and this is carried out by an effective TS algorithm (Section 5). In order not to go far away from each fractional optimum $\bar{x}_{[k]}$, we constraint TS to explore only the points $x \in \mathcal{S}$ such that $|x, \bar{x}_{[k]}|$, the distance between x and $\bar{x}_{[k]}$ is no greater than a given threshold δ_{max}[2].

To sum up, the proposed approach is composed of three steps:

1. determine interesting hyperplanes $\sigma(x) = k$;
2. run simplex to obtain the corresponding fractional optimum $\bar{x}_{[k]}$;
3. run Tabu Search around each $\bar{x}_{[k]}$, limited to a sphere of fixed radius.

4 Determine Hyperplanes $\sigma(x) = k$ and Simplex Phase

Given an n size instance of MKP01, it is obvious that the $n+1$ values $0 \leq k \leq n$ do not have the same interest regarding the optimum. Furthermore exploring all the hyperplanes $\sigma(x) = k$ would be too much time consuming. We propose, in this section, a simple way to compute good values of k.

Starting from a 0–1 lower bound z, the method consists in solving, with the *simplex* algorithm, the two following problems:

$$\text{MKP}\sigma min[z] \left\{ \begin{array}{l} \text{minimize } \sigma(x) \text{ s.t.} \\ A.x \leq b \text{ and } x \in [0\text{–}1]^n \text{ and} \\ c.x \geq (z+1) \end{array} \right.$$

Let σmin be the optimal value of this problem. A knapsack that holds fewer items than $k_{min} = \lceil \sigma min \rceil$ respects no longer the constraint $c.x \geq (z+1)$.

$$\text{MKP}\sigma max[z] \left\{ \begin{array}{l} \text{maximize } \sigma(x) \text{ s.t.} \\ A.x \leq b \text{ and } x \in [0\text{–}1]^n \text{ and} \\ c.x \geq (z+1) \end{array} \right.$$

Let σmax be the optimal value of this problem. It is not possible to hold more items than $k_{max} = \lfloor \sigma max \rfloor$ without violate one of the m constraints $A_i.x \leq b_i$.

[1] This hypothesis is confirmed by experimental results presented later. Of course, the hypothesis does not exclude the possibility that other areas may contain good solutions.

[2] This point is discussed more formally in section 5.2.

Consequently, local search will run only in the hyperplanes $\sigma(x) = k$ for which k is bounded by k_{min} and k_{max}. Hence we compute, using once again the *simplex* algorithm, the $(k_{max} - k_{min} + 1)$ relaxed MKP[k] which give us the continuous optima $\bar{x}_{[k]}$ which are used by the TS phase.

5 Tabu Search Phase

A comprehensive introduction of Tabu Search may be found in [Glover and Laguna, 1997]. We give briefly below some notations necessary for the understanding of our TS algorithm TS^{MKP}.

5.1 Notations

The following notations are specifically oriented to the MKP01:

- a **configuration** x is a binary vector with n components;
- the unconstrained **search space** $\mathcal{S}$ is defined to equal the set $\{0, 1\}^n$, including both feasible and unfeasible configurations;
- a **move** consists in changing a small set of components of x giving x' and is denoted by $mvt(x, x')$;
- in this binary context, the flipped variables of a move can be identified by their indexes in the x vector: These indexes are the **attributes** of the move;
- the **neighborhood** of x, $\mathcal{N}(x)$, is the subset of configurations reachable from x in one move. In this binary context, a move from x to $x' \in \mathcal{N}$ can be identified without ambiguity by the attribute j if x' is obtained by flipping the j_{th} element of x. More generally, we use $mvt(i_1, i_2, ..., i_k)$ to denote the move $mvt(x, x')$ where $\forall j \in [1, k]\, x'_{i_j} = 1 - x_{i_j}$. Such a move is called a k_change.

5.2 Search Space Reduction

Regarding the knapsack constraints ($A.x \le b$), $\mathcal{S} = \{0, 1\}^n$ is the completely relaxed search space. It is also the largest possible search space. We specify in this section a reduced search space $\mathcal{X} \subset \mathcal{S}$ which will be explored by our Tabu Search algorithm. To define this reduced space, we take the ideas discussed previously (Section 3):

1. limitation of $\mathcal{S}$ to a *sphere* of fixed radius around the point $\bar{x}_{[k]}$, the optimal solution of the relaxed MKP[k]. That is: $|x, \bar{x}_{[k]}| \le \delta_{max}$;
2. limitation of the number of objects taken in the configurations x, that are visited by the tabu search algorithm, to the constant k (intersection of $\mathcal{S}$ and the hyperplane $\sigma(x) = k$).

For the first point, we use $\sum_{j=1}^{n} |x_j - x'_j|$ to define the distance $|x, x'|$ where x et x' may be binary or continuous. We use the following heuristic to estimate the maximal distance δ_{max} authorized from the starting point $\bar{x}_{[k]}$. Let $(1, 1, \cdots, 1, r_1, \cdots, r_q, 0, \cdots, 0)$ be the elements of the vector $\bar{x}_{[k]}$ sorted in decreasing order. r_j are fractional components of $\bar{x}_{[k]}$ and we have: $1 > r_1 \ge r_2 \ge \cdots \ge r_q > 0$. Let u be the number of the components having the value of 1 in $\bar{x}_{[k]}$. In the worst case, we select r_j items rather than the u components. With $\sigma(\bar{x}_{[k]}) = k$ that gives $\delta_{[k]} = 2 \times (u + q - k)$. If $u = k$, it follows $\delta_{[k]} = 0$ that corresponds to the case where $\bar{x}_{[k]}$ is a binary vector. Depending on the MKP01 instances, we choose $\delta_{max} \approx \delta_{[k]}$. Hence, each tabu process running around $\bar{x}_{[k]}$ has its own search space $\mathcal{X}_k$:

$$\mathcal{X}_k = \{x \in \{0, 1\}^n \,|\, \sigma(x) = k \wedge |x, \bar{x}_{[k]}| \le \delta_{max}\}$$

Note that all $\mathcal{X}_k$ are disjoint, this is a good feature for a distributed implementation of our approach.

Finally, in order to further limit TS to interesting areas of $\mathcal{X}_k$, we add a qualitative constraint on visited configurations:

$$c.x > z_{min}$$

where z_{min} is the best value of a feasible configuration found so far.

5.3 Neighborhood

A neighborhood which satisfies the constraint $\sigma(x) = k$ is the classical add/drop neighborhood: we remove an object from the current configuration and add another object to it at the same time. This neighborhood is used in this study. The set of neighboring configurations $\mathcal{N}(x)$ of a configuration x is thus defined by the formula:

$$\mathcal{N}(x) = \{x' \in \mathcal{X}_k / \, |x, x'| = 2\}$$

It is easy to see that this neighborhood is a special ase of 2_change (cf. sect. 5.1). We use in the following sections indifferently $mvt(x, x')$ or $mvt(i, j)$ with $x'_i = 1 - x_i$ and $x'_j = 1 - x_j$. As TS evaluates the whole neighborhood for a move, we have a time complexity of $O\left((n - k) \times k\right)$ for each iteration of TS^{MPK}.

5.4 Tabu List Management

The reverse elimination method (REM), introduced by Fred Glover [Glover, 1990], leads to an exact tabu status (*i.e.* $mvt(x, x')$ tabu $\Leftrightarrow$ x' has been visited). It consists in storing in a *running list* the attributes (pair of components) of all completed moves. Telling 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 the RCS, or dropped Otherwise. The condition $RCS = \emptyset$ corresponds to a move leading to a yet visited configuration. For more details on this method see [Dammeyer and Voß, 1993; Glover, 1990]. REM has a time complexity $O(iter^2)$ ($iter$ is the current value of the move counter).

For our specific 2_change move, we need only one trace of the *running list*. Each time we meet $|RCS| = 2$, the move with RCS_0 and RCS_1 attributes is said tabu. Let $iter$ be the move counter. The following algorithm updates the tabu status of the whole neighborhood of a configuration x and corresponds to the equivalence

$$iter = tabu[i][j] \Longleftrightarrow mvt(i, j)\, tabu$$

Algorithme 1: UPDATE_TABU

$i \leftarrow erl$ % end of *running list*
repeat
  $i \leftarrow i - 1$
  $j \leftarrow$ *running list*$[i]$
  if $j \in RCS$ **then**
    $RCS \leftarrow RCS \ominus j$
  else
    $RCS \leftarrow RCS \oplus j$
  if $|RCS| = 2$ **then**
    $tabu[RCS_0][RCS_1] \leftarrow iter$
    $tabu[RCS_1][RCS_0] \leftarrow iter$
until $i = 0$

This algorithm traces back the *running list* table from its $erl - 1$ entry to its 0 entry.

5.5 Evaluation Function

We use a two components evaluation function to assess the configurations $x \in S$: $v_b(x) = \sum_{i\,|\,a_i x > b_i}(a_i.x - b_i)$ and $z(x) = c.x$. To make a move from x, we take among $x' \in \mathcal{N}(x)$ the configuration x' defined by the following relation:

$$x' \in \mathcal{N}(x) \text{ such that } \begin{cases} \forall\, x'' \in \mathcal{N}(x) \\ v_b(x') < v_b(x'') \text{ or} \\ v_b(x') = v_b(x'') \text{ and } c.x' \geq c.x'' \end{cases}$$

Random choice is used to break ties.

Each time $v_b(x) = 0$ the *running list* is reset and z_{min} is updated. Hence the tabu algorithm explores areas of the search space where $z > z_{min}$ trying to reach feasibility.

5.6 Tabu Algorithm

Based on the above considerations, our TS algorithm for MKP01 (TS^{MPK}) works on a series of problems like:

$$\text{find } x \in \mathcal{S} = \{0,1\}^n \text{ such that}$$
$$A.x \leq b \wedge \textstyle\sum_1^n x_i = k \wedge |x, \bar{x}_{[k]}| \leq \delta_{max} \wedge c.x > z_{min}$$

where z_{min} is a strictly increasing sequence. Let $|R.L.|$ be the *running list* size. Considering the last feature of our algorithm given just above (sect. 5.5), $|R.L.|$ is twice the maximum number of iterations without improvement of the cost function. The starting configuration x_{init} is built with the following simple greedy heuristic: Choose the k items $\{i_1, \cdots, i_k\}$, which have the largest $\bar{x}_{[k]i_j}$ value.

6 Computational Results

All the procedures of TS^{MPK} have been coded in C language. Our algorithm has been executed on different kind of *CPU* (up to 20 distributed computers like PII350, PIII500, ULTRASPARC5 and 30). For all the instances solved below, we run TS^{MPK} with 10 random seeds (0..9) of the standard *srand()* C function.

We have first tested our approach on the 56 classical problems used in [Aboudi and Jörnsten, 1994; Balas and Martin, 1980; Chu and Beasley, 1998; Dammeyer and Voß, 1993; Drexl, 1988; Fréville and Plateau, 1986; 1993; Glover and Kochenberger, 1996; Shih, 1979; Toyoda, 1975]. The size of these problems varies from n=6 to 105 items and from m=2 to 30 constraints. These instances are easy to solve for state-of-the-art algorithms. Indeed, our approach finds the optimal value (known for all these instances) in an average time

Algorithme 2: TS^{MKP}

$iter \leftarrow 0$ % Move counter
$(erl, tabu[][]) \leftarrow (0, (-1, \ldots, -1))$
$x \leftarrow x_{init}$
if $v_b(x) = 0$ **then**
  $z_{min} \leftarrow c.x; x^* \leftarrow x$
else
  $z_{min} \leftarrow 0; x^* \leftarrow (0)$
repeat
  $(v_{min}, z_{max}) \leftarrow (\infty, -\infty)$
  for $1 \leq i < j \leq n$ **do**
    if $tabu[i][j] \neq iter$ **then**
      $(x_i, x_j) \leftarrow (1 - x_i, 1 - x_j)$ %$mvt(i,j)$ evaluation
      if $(|x, \bar{x}_{[k]}| \leq \delta_{max}) \wedge (c.x > z_{min})$ **then**
        if $(v_b(x) < v_{min}) \vee (v_b(x) = v_{min} \wedge c.x > z_{max})$ **then**
          $(i', j') \leftarrow (i, j)$
          $(v_{min}, z_{max}) \leftarrow (v_b(x), c.x)$
      $(x_i, x_j) \leftarrow (1 - x_i, 1 - x_j)$ % Restore old values
  if $v_{min} \neq \infty$ **then**
    $(x_{i'}, x_{j'}) \leftarrow (1 - x_{i'}, 1 - x_{j'})$ % Complete move
    if $v_{min} = 0$ **then**
      $erl \leftarrow 0$ %Reset the *running list*
      $z_{min} \leftarrow c.x$
      $x^* \leftarrow x$
    else
      $iter \leftarrow iter + 1$
      running list $\leftarrow$ *running list* $\oplus\, i' \oplus\, j'$
      $erl \leftarrow erl + 2$
      UPDATE_TABU
until $v_{min} = \infty \vee erl \geq |R.L.|$

of 1 second (details are thus omitted). The *running list* size ($|R.L.|$) is fixed to 4000.

For the next two sets of problems, the *running list* size ($|R.L.|$) is fixed to 100000. Hence the maximum number of moves without improvement is 50000.

The second set of tested instances is constituted of the last seven (also the largest ones with n =100 to 500 items, m = 15 to 25 constraints) of 24 benchmarks proposed by Glover and Kochenberger [Glover and Kochenberger, 1996]. These instances are known to be particularly difficult to solve for *Branch & Bound* algorithms. Table **1** shows a comparison between our results (columns 4 to 7) and the best known ones reported in [Hanafi and Fréville, 1998] (column T_{HF}).

GK	$n \times m$	T_{HF}	z^*	k^*	$iter^*$	$sec.^*$
18	100×25	4524	**4528**	61	3683	10
19	100×25	3866	**3869**	51	3144	9
20	100×25	5177	**5180**	70	2080	5
21	100×25	3195	**3200**	42	1465	4
22	100×25	2521	**2523**	34	512	2
23	200×15	9231	**9235**	123	16976	131
24	500×25	9062	**9070**	119	9210	268

Table 1: *Comparative results on the 7 largest* GK *pb.*

Column k^* shows the number of items of the best solution x^* with cost z^* found by TS^{MKP}. From the table, we observe that all the results are improved. Columns $iter^*$ and $sec.^*$ give the number of moves and the time elapsed to reach x^*. We know that the algorithm runs 50000 moves after $iter^*$. The search process takes thus an average of 380 seconds for the test problems GK18...GK22, 600 seconds for GK23 and 1100 seconds for GK24.

The third set of test problems concerns the 30 largest benchmarks (n = 500 items, m = 30 constraints) of OR-

Library[3], proposed recently by Chu and Beasley [Chu and Beasley, 1998].

CB	AG_{CB}	z^*	k^*	$iter^*$	$sec.^*$
30.500.0	115868	**115950**	130	17841	397
30.500.1	114667	**114810**	128	104866	2264
30.500.2	116661	**116683**	128	73590	1203
30.500.3	115237	**115301**	128	71820	1587
30.500.4	116353	**116435**	127	75909	1784
30.500.5	115604	**115694**	131	33391	684
30.500.6	113952	**114003**	128	107994	2851
30.500.7	114199	**114213**	129	87593	1503
30.500.8	115247	**115288**	127	75243	1495
30.500.9	116947	**117055**	129	39044	869
30.500.10	217995	**218068**	251	6828	116
30.500.11	214534	**214562**	251	89201	2478
30.500.12	215854	**215903**	250	60074	1311
30.500.13	217836	**217910**	251	50732	1121
30.500.14	215566	**215596**	251	62524	1262
30.500.15	215762	**215842**	253	34201	633
30.500.16	215772	**215838**	252	54476	1003
30.500.17	216336	**216419**	253	40683	947
30.500.18	217290	**217305**	253	64489	1475
30.500.19	214624	**214671**	252	18531	368
30.500.20	301627	**301643**	375	1298	17
30.500.21	299985	**300055**	374	78278	1532
30.500.22	304995	**305028**	375	64926	1161
30.500.23	301935	**302004**	375	26901	1110
30.500.24	304404	**304411**	376	20483	333
30.500.25	296894	**296961**	374	31403	462
30.500.26	303233	**303328**	373	43398	757
30.500.27	306944	**306999**	376	33810	1366
30.500.28	303057	**303080**	374	17647	350
30.500.29	300460	**300532**	376	6948	150

Table 2: *Comparative results on the 30 largest* CB *pb.*

Table **2** compares our results with those reported in [Chu and Beasley, 1998] (AG_{CB}), which are among the best results for these instances. From the table, we see that our approach improves significantly on all these results. The average time of the 50000 last iterations is equal to 1200 seconds for these instances.

These results can be further improved by giving more *CPU* time (iterations) to our algorithm. For example, with $|R.L.| = 300000$, TS^{MKP} finds $z^* = 115991$ after 6000 seconds for the instance CB30.500.0.

The Chu and Beasley *benchmark* contains 90 instances with 500 variables: 30 instances with m=5 constraints, 30 with $m = 10$ and 30 with $m = 30$ (results detailed just above). Each set of 30 instances is divided into 3 series with $\alpha = b_i / \sum_{j=1}^{n} A_{ij} = 1/4$, $\alpha = 1/2$ and $\alpha = 3/4$. Table **3** compares, for each subset of 10 instances, the averages of the best results obtained by AG_{CB}, those obtained more recently by Osorio, Glover and Hammer [Osorio *et al.*, 2000] (columns 4 and 5) and those by TS^{MPK} (column 6). The new algorithm of [Osorio *et al.*, 2000] uses advanced techniques such as cutting and surrogate constraint analysis (see column *Fix+Cuts* for results). We reproduce also from [Osorio *et al.*, 2000], in column $CPLEX$, the best values obtained by the *MIP solver CPLEX v6.5.2* alone.

m	α	AG_{CB}	*Fix+Cuts*	*CPLEX*	z^*	gap
5	1/4	120616	120610	120619	**120623**	0.08%
	1/2	219503	219504	219506	**219507**	0.04%
	3/4	302355	**302361**	302358	302360	0.02%
10	1/4	118566	118584	118597	**118600**	0.20%
	1/2	217275	217297	217290	**217298**	0.09%
	3/4	302556	302562	302573	**302575**	0.07%
30	1/4	115470	115520	115497	**115547**	0.55%
	1/2	216187	216180	216151	**216211**	0.24%
	3/4	302353	302373	302366	**302404**	0.15%

Table 3: *Average performance over the 90 largest* CB *pb.*

The column gap indicates the average gap values in percentage between the continuous relaxed optimum and the best cost value found: $(\bar{z} - z^*)/\bar{z}$. $Fix{+}cuts$ and $CPLEX$ algorithms were stopped after 3 hours of computing or when the tree size memory of 250M bytes was exceeded. Our best results were obtained with $|R.L.| = 300000$ and the algorithm never requires more than 4M bytes of memory. Except for the instances with $m = 5$ and $\alpha = 3/4$ our approach outperforms all the other algorithms.

To finish the presentation on Chu and Beasley benchmarks, Table **4** and **5** show the best values obtained by our algorithm on the 30 CB5.500 and the 30 CB10.500 instances.

XX	z^*	k^*	XX	z^*	k^*	XX	z^*	k^*
0	120134	146	10	218428	267	20	295828	383
1	117864	148	11	221191	265	21	308083	383
2	121112	145	12	217534	264	22	299796	385
3	120804	149	13	223558	264	23	306478	385
4	122319	147	14	218966	267	24	300342	385
5	122024	153	15	220530	262	25	302561	384
6	119127	145	16	219989	266	26	301329	385
7	120568	150	17	218194	266	27	306454	383
8	121575	148	18	216976	262	28	302822	382
9	120707	151	19	219704	267	29	299904	379

Table 4: *Best values for* CB5.500.XX

XX	z^*	k^*	XX	z^*	k^*	XX	z^*	k^*
0	117779	134	10	217343	256	20	304351	378
1	119190	134	11	219036	259	21	302333	380
2	119194	135	12	217797	256	22	302408	379
3	118813	137	13	216836	258	23	300757	378
4	116462	134	14	213859	256	24	304344	381
5	119504	137	15	215034	257	25	301754	375
6	119782	139	16	217903	261	26	304949	378
7	118307	135	17	219965	256	27	296441	379
8	117781	136	18	214341	258	28	301331	379
9	119186	138	19	220865	255	29	307078	378

Table 5: *Best values for* CB10.500.XX

To conclude this section, we present our results on the 11 latest instances proposed very recently by Glover and Kochenberger (available at: `http://hces.bus.olemiss.edu/tools.html`.) These benchmarks contain up to n=2500 items and m=100 constraints, thus are very large. Table **6** compares our results (columns 4 and 5) and the best known results taken from the above web site. Once again, our approach gives improved solutions for 9 out of 11 instances. Let us mention that the experimentation on

[3]Available at `http://mscmga.ms.ic.ac.uk`.

these instances showed some limits of our approach regarding the computing time for solving some very large instances ($n > 1000$). Indeed, given the size of our neighborhood ($k \times (n-k)$, see Section 5.3), up to 3 days were needed to get the reported values.

MK_GK	$n \times m$	z^{GK}	z^*	k^*	gap
01	100×15	3766	3766	52	0.26%
02	100×25	3958	3958	50	0.46%
03	150×25	5650	**5656**	78	0.26%
04	150×50	5764	**5767**	78	0.46%
05	200×25	7557	**7560**	104	0.23%
06	200×50	7672	**7677**	99	0.34%
07	500×25	19215	**19220**	259	0.06%
08	500×50	18801	**18806**	252	0.13%
09	1500×25	58085	**58087**	773	0.02%
10	1500×50	57292	**57295**	770	0.04%
11	2500×100	95231	**95237**	1271	0.04%

Table 6: *Comparison on the lastest 11* MK_GK *pb.*

7 Conclusion

In this paper, we have presented a hybrid approach for tackling the NP-hard 0–1 multidimensional knapsack problem (MKP01). The proposed approach combines linear progamming and Tabu Search. The basic idea consists in obtaining "promising" continuous optima and then using TS to explore carefully and efficiently binary areas close to these continuous optima. This is carried out by introducing several additional constraints:

1. hyperplane constraint:$\sum_1^n x_i = k \in \mathbb{N}$;
2. geometrical constraint:$|x, \bar{x}_{[k]}| \leq \delta_{max}$;
3. qualitative constraint: $c.x > z_{min}$.

Our Tabu Search algorithm integrates also some advanced features such as an efficient implementation of the reverse elimination method for a dynamic tabu list management in the context of a special *2-change* move.

This hybrid approach has been tested on more than 100 state-of-the-art benchmark instances, and has led to improved solutions for most of the tested instances.

One inconvenience of the proposed approach is its computing time for very large instances (n > 1000 items). This is partially due to the time complexity of the neighborhood used. Despite of this, this study constitutes a promising starting point for designing more efficient algorithms for the MKP01. Some possibilities are: 1) to develop a partial evaluation of relaxed knapsack constraints; 2) to study more precisely the relationship between δ_{max} and z^{opt} for a given instance; 3) to find a good crossover operator and a cooperative distributed implementation of the TS^{MKP} algorithm.

Finally, we hope the basic idea behind the proposed approach may be explored to tackle other NP-hard problems.

Acknowledgement: We would like to thank the reviewers of the paper for their useful comments.

References

[Aboudi and Jörnsten, 1994] R. Aboudi and K. Jörnsten. Tabu Search for General Zero-One Integer Programs using the Pivot and Complement Heuristic. *ORSA Journal of Computing*, 6(1):82–93, 1994.

[Balas and Martin, 1980] E. Balas and C.H. Martin. Pivot and a Complement Heuristic for 0-1 Programming. *Management Science*, 26:86–96, 1980.

[Chu and Beasley, 1998] P.C. Chu and J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristic*, 4:63–86, 1998.

[Dammeyer and Voß, 1993] F. Dammeyer and S. Voß. Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research*, 41:31–46, 1993.

[Drexl, 1988] A. Drexl. A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing*, 40:1–8, 1988.

[Fréville and Plateau, 1986] A. Fréville and G. Plateau. Heuristic and reduction methods for multiple constraints 0-1 linear programming problems. *European Journal of Operational Research*, 24:206–215, 1986.

[Fréville and Plateau, 1993] A. Fréville and G. Plateau. Sac à dos multidimensionnel en variable 0-1 : encadrement de la somme des variables à l'optimum. *Recherche Opérationnelle*, 27(2):169–187, 1993.

[Gavish and Pirkul, 1982] B. Gavish and H. Pirkul. Allocation of data bases and processors in a distributed computing system. *Management of Distributed Data Processing*, 31:215–231, 1982.

[Gavish and Pirkul, 1985] B. Gavish and H. Pirkul. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31:78–105, 1985.

[Glover and Kochenberger, 1996] F. Glover and G.A. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In I.H. Osman J.P. Kelly, editor, *Metaheuristics: The Theory and Applications*, pages 407–427. Kluwer Academic Publishers, 1996.

[Glover and Laguna, 1997] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[Glover, 1990] F. Glover. Tabu search. *ORSA Journal of Computing*, 2:4–32, 1990.

[Hanafi and Fréville, 1998] S. Hanafi and A. Fréville. An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 106:659–675, 1998.

[Osorio *et al.*, 2000] M.A. Osorio, F. Glover, and Peter Hammer. Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. Technical report, Hearin Center for Enterprise Science. Report HCES-08-00, 2000.

[Shih, 1979] W. Shih. A branch & bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society*, 30:369–378, 1979.

[Toyoda, 1975] Y. Toyoda. A simplified algorithm for obtaining approximate solutions to zero-one programming problem. *Management Science*, 21(12):1417–1427, 1975.