Tabu search: global intensification using dynamic programming

Christophe Wilbaut, Saïd Hanafi, Arnaud Fréville, Stefan Balev

Control and Cybernetics

vol. **35** (2006) No. 3

Tabu search: global intensification using dynamic programming

by

Christophe Wilbaut[1], Saïd Hanafi[1], Arnaud Fréville[1] and Stefan Balev[2]

[1] LAMIH-ROI, Université de Valenciennes
Le Mont Houy, 59313 Valenciennes, Cedex 9, France

[2] Université du Havre
25 Rue Philippe Lebon, BP 1123, 76063 Le Havre Cedex, France

e-mail: christophe.wilbaut@univ-valenciennes.fr,
said.hanafi@univ-valenciennes.fr, a.freville@cr-npdc.fr,
Stefan.Balev@univ-lehavre.fr

Abstract: Tabu search has proven highly successful in solving hard combinatorial optimization problems. In this paper, we propose a hybrid method that combines adaptive memory, sparse dynamic programming, and reduction techniques to reduce and explore the search space. Our approach starts with a bi-partition of the variables, involving a small core problem, which never exceeds 15 variables, solved using the "forward" phase of the dynamic programming procedure. Then, the remaining subspace is explored using tabu search, and each partial solution is completed with the information stored during the forward phase of dynamic programming. Our approach can be seen as a global intensification mechanism, since at each iteration, the move evaluations involve solving a reduced problem implicitly. The proposed specialized tabu search approach was tested in the context of the multidimensional 0-1 knapsack problem. Our approach was compared to ILOG's commercial product CPLEX and to the corresponding "pure" tabu search (i.e., without a core problem) for various sets of test problems available in OR-libraries. The results are encouraging. In particular, this enhances the robustness of the approach, given that it performs better than the corresponding pure tabu search most of the time. Moreover, our approach compares well with CPLEX when the number of variables is large; it is able to provide elite feasible solutions in a very reasonable amount of computational time.

Keywords: tabu search, dynamic programming, global intensification, multidimensional 0-1 knapsack problem.

1. Introduction

We propose a hybrid algorithm that combines adaptive memory techniques and sparse dynamic programming for solving some combinatorial optimization problems. Dynamic programming is used on a subset of variables to solve exactly a family of sub-problems that differ one from another only in the right-hand side constraint values. Tabu search is then applied to the remaining variables in the subset, bringing into play information generated during the dynamic programming phase. The efficiency of the method lies in the way the variables are partitioned into two subsets as required by the dynamic programming memory.

We developed this hybrid approach for solving the well-known multidimensional 0-1 knapsack problem (*MKP*). The *MKP* consists of finding a subset of items that maximizes a linear objective function while respecting the capacity constraints. The *MKP* can be formulated as follows:

$$\begin{array}{rlll} z(MKP) := & max \sum_{j\in N} c_j x_j & & \\ & s.t. \sum_{j\in N} a_{ij} x_j \leq b_i & i \in M = \{1,2,\ldots,m\} & (1) \\ & x_j \in \{0,1\} & j \in N = \{1,2,\ldots,n\} & \end{array}$$

where N is the set of items; c_j, the profit $j \in N$; M, the set of knapsack constraints; b_i, the capacity of knapsack $i \in M$; and a_{ij} , the consumption of resource $i \in M$ when item j is selected. All data - c_j, b_i and a_{ij} - are non-negative integers. Without loss of generality, we can assume that $c_j > 0$, $j \in N$, $b_i > 0$, $i \in M$, and $max_{j\in N} a_{ij} \leq b_i \leq \sum_{j\in N} a_{ij}$, for $i \in M$. We use the following shortcut notation for the problem:

$$(MKP) \quad max\{c^T x : Ax \leq b, x \in \{0;1\}^n\}. \tag{2}$$

We use a_j to denote the j-th column of the matrix A, namely $a_j = (a_{1j}, a_{2j}, ..., a_{mj})$. Throughout the remainder of this paper, $z(P)$ denotes the optimal value of the optimization problem (P). *MKP* is known to be NP-hard, but not strongly NP-hard (Fréville, Hanafi, 2005). *MKP* has been widely discussed in the literature, and efficient exact and approximate algorithms have been developed for obtaining optimal and near-optimal solutions (see Fréville, 2004, for a comprehensive annotated bibliography). In particular, *MKP* has been shown to become significantly harder to solve as m increases. For $m = 1$, some very efficient algorithms (Kellerer, Pferschy, Pisinger, 2004; Martello, Toth, 1990) do exist, but as m increases, exact methods (e.g., the last versions of CPLEX, Osorio, Glover, Hammer, 2000) usually fail to provide an optimal solution for even moderate-size instances.

Dynamic programming (*DP*), Bellman (1957), is one of the seminal methods proposed in the literature for solving knapsack problems. Generally, dynamic programming-based algorithms are easy to implement and very efficient for small- and medium-sized instances. However, dynamic programming does not address all types of combinatorial optimization problems and fails on large-scale

instances due to prohibitive computational time and memory requirements. Several algorithms based on dynamic programming have been developed for both the simple knapsack problem, Toth (1980), and the multidimensional knapsack problem, Fréville (2004), with more or less conclusive results. In cases involving a single constraint, the recent approaches using dynamic programming hybrids and other techniques (Plateau, Elkihel, 1985; Viader, 1998) have proved very efficient for solving large-scale instances. The situation is not so idyllic when multiple constraints are involved, mainly because the amount of information that must be stored during the process increases sharply with m (see Berstimas, Demir, 2002, for an example of such an algorithm).

In the past 30 years, many researchers have studied heuristics and metaheuristics devoted to the *MKP* (Fréville, Hanafi, 2005). Tabu search (*TS*), introduced by Glover (1986) and Hansen (1986), is acknowledged to be well suited to solving large-scale *MKP* instances. The main concepts were developed by Glover (1989, 1990), and many efficient implementations and hybrids based on other suitable heuristics and metaheuristics have been proposed over the intervening years.

This paper is organized as follows. Section 2 describes the dynamic programming method applied to the first subset of variables, which yields an improved solution and fixes variables at their optimal values. Section 3 presents the tabu search procedure that is applied to the remaining subset of variables in order to improve the best current feasible solution. The global intensification procedure is explained in Section 4 and illustrated with a simple example. In Section 5, computational results are provided for various sets of problems, and Section 6 presents our conclusions and offers suggestions for future research.

2. Dynamic programming with reduction

In this section we first formulate the *DP* functional equation for *MKP*. Then we show how the dynamic programming, combined with the LP-relaxation information and the best-known solution, allows us either to prove that the best feasible solution obtained is optimal, or to fix a subset of variables at their optimal values.

2.1. Dynamic programming phase

Let us consider the family of sub-problems $MKP[k, \beta]$ which involves k variables, $k \in [0; n]$, according to residual capacities β, with $0 \leq \beta \leq b$:

$$\begin{array}{rrll} f(k,\beta) := & \max \sum_{j\in\{1,\ldots,k\}} c_j x_j & & \\ & s.t. \sum_{j\in\{1,\ldots,k\}} a_{ij} x_j \leq \beta_i & i \in M = \{1,2,\ldots,m\} & (3) \\ & x_j \in \{0,1\} & j \in \{1,2,\ldots,k\}. & \end{array}$$

As $f(k,\beta)$ (respectively $f(n,b)$) denotes the value of the sub-problem $MKP[k,\beta]$ (resp. MKP), the following recurrence relations hold:

$$f(k,\beta) = \max\{f(k-1,\beta), c_k + f(k-1,\beta-a_k)\}, if a_k \leq \beta \quad (4)$$

$$f(k,\beta) = f(k-1,\beta), if a_k \nleq \beta \quad (5)$$

for $k = 1, 2, ..., n$, with the initial conditions $f(0,\beta) = 0$, $0 \leq \beta \leq b$. In the worst case, this approach requires a total memory space in $O(n \times b_1 \times b_2 \ldots \times b_m)$. Even given a single constraint, this memory requirement becomes too demanding as soon as the size n, and/or the second member b, increases. To save memory space, we use a sparse representation based on the step-wise growth feature of the function $f(k,\beta)$. This sparse representation has already been used for the knapsack problem (Viader, 1998), and has been generalized to the MKP. The main idea is to maintain a list L of $(m+2)$-tuples, in which each element of L represents a state (v,β',k), where v denotes the objective function value; β', the amount of resource available for the remaining $n-k$ variables; and k, the number of the step. The last component k of each element in the list is used in the backtracking phase to restore an optimal solution.

In the continuation, elements of L are denoted (v,β',k), or simply by (v,β') when specifying k does not facilitate comprehension. In spite of the sparse representation, the DP application is not able to solve large instances of MKP due to its spatial and temporal requirements. However, the initial steps are very fast and can provide information that can be used to set variables (see Section 2.2). So, for a given MKP instance, the DP algorithm is only performed on a limited number of steps, noted n'. This parameter n' is determined experimentally in terms of the machine capacities and the size of the instance being solved (typically, the number of constraints), with the main goal being to apply dynamic programming to the maximum number of variables, while still respecting memory and computational requirements.

The algorithm proves fast enough when the memory needed to store the list L does not exceed a threshold of Mem units, where Mem reflects the characteristics of the computer used for running the code. In fact, during the forward phase, the number of elements in the list doubles at each iteration in the worst case. So, after the first n' iterations, the list L has at most $2^{n'}$ elements, since each tuple of L corresponds to a feasible solution and since, for small value of n', almost all solutions are feasible. Dynamic Programming is used here to enumerate the search space $\{0,1\}^{n'}$. Because each element is composed of $m+2$ values, the size of L is, at most, $2^{n'} \times (m+2)$ after n' iterations. For this reason, the value $n' = log_2(Mem) - log_2(m+2)$ would seem to be a good choice. In our experiments, the value of n' is small compared to n - between 0.3% and 15% (see Section 5 for more details). The choice of the first n' variables solved using DP is guided by a reduction technique logic.

The DP algorithm could be enhanced by integrating dominance techniques, which would eliminate some elements in the list L during the forward process.

Such enhancement has been shown to be very efficient for single-constraint knapsack problem. Unfortunately, this is not true for *MKP*; when $m > 1$, detecting dominance is very time consuming. In addition, dominance occurs very rarely in the solution process, since for a set of 270 instances (described in Section 5), the percentage of dominated elements in the list at the end of the forward process varies between 0% and 3%, with an average of 0.04%. Thus, the high price paid to detect dominance is not compensated by the elimination of a large number of elements in L. For these reasons, we chose not to use the dominance principle.

2.2. Reduction technique

Preprocessing techniques play a fundamental role in developing efficient integer programming approaches. The basic tools involve setting variables, identifying infeasibility and constraint redundancy, and tightening the LP-relaxation by modifying coefficients and by generating strong valid inequalities (see Savelsberg, 1994, for a framework of basic techniques). In our method, the reduction technique is only used to set variables to their optimal values.

Given a feasible solution x^0 and a subset $J \subseteq N$, the reduced problem denoted by $MKP(x^0, J)$ is extracted from the original problem *MKP* by fixing each x_j for $j \in J$ to its value in x^0 (i.e. $x_j = x_j^0$, $j \in J$). Obviously, $MKP(x^0, \oslash)$ corresponds to the initial *MKP*, and its value is equal to $c^T x^0$. As soon as the size of set J decreases, the reduced problem $MKP(x^0, J)$ becomes more difficult to solve. Thus, for any subset $J' \subseteq J$, we have:

$$z(MKP(x^0, J')) \geq z(MKP(x^0, J)). \tag{6}$$

In the following, shortcut notations are used: $MKP(x_j^0)$ for the reduced problem $MKP(x^0, \{j\})$ and $MKP(1 - x_j^0)$ for $MKP(e - x^0, \{j\})$, where e is the vector whose components are all set at 1. The common variable fixation techniques are based on knowing a good lower bound associated with a feasible solution x^0, and rely on the following property:

LEMMA 2.1 *For any $j \in N$, and for any feasible solution $x^0 \in \{0,1\}^n$, if $z(MKP(1 - x_j^0)) \leq c^T x^0$ then either $x_j = x_j^0$ in any optimal solution, or x^0 is optimal.*

Although the problem $MKP(1 - x_j^0)$ is as difficult to solve as the original *MKP*, the above property remains valid if $z(MKP(1 - x_j^0))$ is replaced by any upper bound from the problem $MKP(1 - x_j^0)$ (refined upper bounds are proposed in Fréville, Plateau, 1994). In the following, we propose an alternative approach based on dynamic programming and *LP* upper bounds that relies on the interdependence between two upper bound $\{u_j\}_{j \in N}$ and lower bound $\{l_j\}_{j \in N}$ sequences. For convenience, we recall the main results given in Andonov et al. (2001) (note that the proofs for the propositions referred to below are also given in Andonov et al., 2001).

Let x^0 be a feasible solution of *MKP*. Let u_j, $j \in N$, be an upper bound of $z(MKP(1 - x_j^0))$ (i.e. $u_j \geq z(MKP(1 - x_j^0)), \forall j \in N$). Throughout the remainder of this paper, we assume that the decision variables x_j are sorted in decreasing order according to the upper bounds u_j:

$$u_1 \geq u_2 \geq \ldots \geq u_n \,. \tag{7}$$

PROPOSITION 2.1 *Let x^0 be a feasible solution to MKP and define*

$$l_j = z(MKP(x^0, \{j+1, \ldots, n\})), j \in N \tag{8}$$

If $l_k \geq u_{k+1}$ for some $k \in \{1, 2, \ldots, n-1\}$ then $z(MKP) = l_k$.

In our experiments, the upper bounds u_j correspond to the value of the LP-relaxation of the reduced problem $MKP(1 - x_j^0)$. The next proposition explains the mechanism for setting the variables, using the upper and lower bounds sequences.

PROPOSITION 2.2 *Assume that there exists a pair of indices (h, k) such that $h < k$ and $l_h \geq u_k$. Then there exists an optimal solution x^*, such that $x_j^* = x_j^0$ for $j = k, k+1, \ldots, n$.*

Let $N' = \{1, \ldots, n'\}$. The lower bounds associated to N' (i.e. $\{l_k\}_{k \in N'}$) are calculated during the dynamic programming process as follows:

$$l_k = \sum_{j=k+1}^{n} c_j x_j^0 + \max\{v | \beta' \geq \sum_{j=k+1}^{n} a_j x_j^0, (v, \beta') \in L\} \,. \tag{9}$$

The value l_k is optimal according to the initial feasible solution x^0 in that it achieves the best feasible completion of the partial solution $(x_{k+1}^0, \ldots, x_n^0)$. In fact, each element $(v, \beta') \in L$ (for the first k iterations) corresponds to a feasible solution, such that $v = \sum_{j=1}^{k} c_j x_j$ and $\beta' = b - \sum_{j=1}^{k} a_j x_j$. Since $\beta' \geq \sum_{j=k+1}^{n} a_j x_j^0$ holds, all the solutions are feasible and (8) ensures that the value of l_k is optimal.

The results of the previous propositions are embedded in Algorithm 1. It is not necessary to know the lower bounds l_k at each iteration, which reduces the computational time required by the algorithm. In fact, in large instances, the probability of fixing variables during the early iterations is small, so our method computes the lower bound only at the last iteration, n'. The process is repeated on the reduced problem until all the variables are set, or until no variable can be set.

3. Tabu search

TS is primarily designed to overcome local optimality by implementing intelligent and reactive tools during a neighborhood search approach. *TS* has aroused

Algorithm 1 Sparse dynamic programming algorithm with reduction

Procedure DP(MKP)

STEP 1: *Pre-processing phase*
  Compute an initial feasible solution x^0 of *MKP* and the sequence $\{u_j\}_{j\in N}$
  Sort the sequence $\{u_j\}_{j\in N}$ according to (7)
  Compute n' and let N'=$\{1,...,n'\}$
  Set $L = \{(0, b, 0)\}$ and $k = 1$;
STEP 2: *Construct the Dynamic Programming List*
while $k \leq n'$ **do**
  Set $L_1 = \emptyset$;
  for all $(v, \beta') \in L$ **do**
    if $\beta' \geq a_k$ **then**
      $L_1 = L_1 \cup \{(v + c_k, \beta' - a_k, k)\}$;
      $L = L \cup L_1; k = k + 1$;
    end if
  end for
end while
STEP 3: *Compute the lower bound*
  Compute $l_{n'}$ according to (9)
STEP 4: *Reduction and optimality test*
if $l_{n'} \geq u_{n'+1}$ **then**
  stop (all the variables are fixed and $l_{n'}$ is the optimal value of *MKP*)
else
  let k be an index such that $u_{k-1} > l_{n'} \geq u_k$
  if k exists **then**
    Add the constraints $x_j = x_j^0$ for $j = k, \ldots, n$ to *MKP*
    Repeat the process by calling *DP*(*MKP*)
  else
    stop (no more fixation is possible)
  end if
end if

great interest in the research community over the last two decades and has proven highly successful at solving hard, large-scale combinatorial optimization problems (See Glover, Laguna, 1997, for a state-of-the art of the *TS* method and its applications[1]).

3.1. Global intensification

The set of decision variables N is partitioned into two subsets, N' and N'' (i.e. $N = N' \cup N''$ and $N' \cap N'' = \oslash$). The subset N' defines the sub-problem solved by using dynamic programming ($N' = \{1, 2, \ldots, n'\}$, where $n' = |N'|$ and $N'' = N - N' = \{n'+1, \ldots, n\}$ with $n'' = |N''| = n - n'$). Let $x = (x', x'')$ be a partition of the vector solution x, where $x' = (x_j)_{j \in N'}$ and $x'' = (x_j)_{j \in N''}$. Similarly, let $A = (A',\ A'')$ and $c = (c',\ c'')$.

The main new feature of our *TS* method is to involve a global intensification mechanism. Because of this mechanism, the tabu search focuses only on the region of the search space defined by the subset N'', each partial solution generated during the process being completed by the solution of a reduced problem. The integration of *DP* in the tabu search process is justified by exploiting the list L generated during the forward phase of *DP*. In fact, each partial solution x'' in $\{0,1\}^{n''}$ visited by *TS* is completed to obtain a feasible solution of the original *MKP* x in $\{0,1\}^n$ (see Algorithm 2). It corresponds to the backtracking phase of *DP*.

Algorithm 2 Global intensification mechanism for completing the solutions in tabu search

Procedure Complete(MKP,L,x'')

STEP 1: *Search Phase*
  Find (v^*, β'^*, k^*) in L such that $v^* = \max\{v : (v, \beta', k) \in L \text{ and } \beta' \geq A''x''\}$
STEP 2: *Construction Phase*
$x' = 0$;
while $k^* > 0$ **do**
  $x'_{k^*} = 1$;
  find $(v', \gamma', k') \in L$ such that $v' = v^* - c_{k^*}, \gamma' = \beta'^* - a_{k^*}$
  $(v^*, \beta'^*, k^*) = (v', \gamma', k')$;
end while

Let x'' be a partial solution in $\{0,1\}^{n''}$, the procedure complete generates a solution x' in $\{0,1\}^{n'}$ such that:

$$c'^T x' = \max\{c'^T y : A'y \leq b - A''x'', y \in \{0,1\}^{n'}\}. \tag{10}$$

[1]see also special issue of *Control and Cybernetics*, *"Tabu Search for Combinatorial Optimization"*, **29**, 3, 2000, Guest Edited by R.V.V. Vidal and Z. Nahorski.

Note that x' is an optimal solution of *MKP* once x'' is set. The list L is ordered to accelerate the backtracking process (the sort is realized during the forward phase, more precisely when the two lists merge).

Although infeasibility during the process has proved interesting in other circumstances (Glover, Kochenberger, 1996; Hanafi, Fréville, 1998), this approach only takes feasible solutions into account in order to complete each partial solution. Thus, to maintain feasibility, an adding or dropping move flips one variable to obtain its neighbor. Specifically, the neighborhood $V(x'')$ of a current solution x'' on N'' is defined such that:

$$V(x'') = \{y \in \{0,1\}^{n''} | \sum_{j \in N''} a_j y_j \leq b, \sum_{j \in N''} |x''_j - y_j| = 1\}. \tag{11}$$

The size of the neighborhood in (11) is equal to $|N''|$. The value of any neighbor y in $V(x'')$ is given by:

$$c^T y = \sum_{j \in N''} c_j y_j + \max\{v : \beta' \geq \sum_{j \in N''} a_j y_j, (v, \beta') \in L\}. \tag{12}$$

The value of the second term in (12) is obtained by the application of the *Complete* procedure.

3.2. Initial solution

The initial feasible solution x^0 used in our approach is generated using an LP-based heuristic with local search. Our procedure is the first step of the method proposed by Soyster, Lev, Slivka (1978). Starting with the LP-relaxation solution, non-basic variables are set to the values 0 or 1, and the remaining subproblem corresponding to the fractional variables is solved to produce a feasible solution x^0. Both the LP-relaxation and the reduced problem are solved with CPLEX. Note that the number of constraints m is such that the reduced *MKP* can be solved exactly with enhanced versions of CPLEX; at worst, good quality lower bounds are generated.

A local search, called *complement heuristic*, is applied to improve the feasible solution x^0 that was generated by the LP-based heuristic. The variables set at 1 in x^0, except k of them, define a partial solution that is completed by a greedy heuristic working only with the remaining "free" variables. The procedure is repeated for each subset with a cardinality equal to k, and the best complete solution x^0 is selected (in our experiments, the value $k = 2$ was selected).
The tabu search procedure starts with the final solution provided by the dynamic programming phase.

3.3. Other TS ingredients

The tabu list is designed to prevent the search from short term cycling. In our method, since a move consists of flipping a single variable at each iteration,

the index of the flipped variable, denoted j, is classified tabu for the next t iterations. The tabu tenure t is determined empirically. Our computational results show that a value t equal to m is a suitable choice.

Generally, *TS* becomes significantly more efficient when long-term memory is incorporated into its two main components: intensification and diversification. Although the moves of our *TS* are defined for the search space $\{0,1\}^{n''}$ of the reduced problem on N'', the intensification and diversification strategies are defined for the search space $\{0,1\}^n$ of the original problem *MKP*. The long-term frequency information, a collection of all the local optima visited during the search space exploration, is stored in a vector, denoted *Elite*. Frequency memory was introduced in order to influence the direction of the search. A penalty term is associated with the frequency information F_j associated with variable x_j (corresponding to the number of times the variables has been set to '1' during the search process) when evaluating the ratio $c_j/\mu a_j$ for each variable x_j where μ is a multiplier. In fact, variables are sorted according to the $(\delta c_j/\mu a_j + (1-\delta)F_j)$ value with δ parameter.

Intensification strategies record and exploit elite solutions, or specific features of such solutions. The intensification procedure starts when the best solution is improved, or when no improvement has occurred during n/m consecutive iterations. Choice rules can be modified in many ways so that they intensify the area around attractive regions, where good elite solutions have historically been found. The intensification procedure tries to improve one of the K best solutions stored in *Elite* during the search through the use of the modified complement heuristic described in Section 3.2.

Diversification strategies, on the other hand, are used to drive the search towards unexplored regions by periodically introducing infrequently used attributes into the solution. Our diversification algorithm considers the variables in terms of a frequency value associated with the number of times the variable has been set to '1' during the search process. This technique diversifies the search by favoring variables with the smallest values. Starting with an initial null solution, variables are added according to the increased frequency value as long as the solution is feasible. If the frequency does not really evolve between two iterations, this kind of diversification runs the risk of producing identical solutions at each iteration. However, it is easy to avoid this situation by, for example, modifying a random number of bits in the solution. A diversification phase is activated as soon as K consecutive intensification phases do not improve the incumbent solution, where K denotes the number of best solutions recorded.

To evaluate the positive impact of the global intensification mechanism, we also developed a "pure tabu search" algorithm without dynamic programming. The pure tabu search process has the same characteristics as the *TS* hybrid; however, the set of decision variables associated is N, and the neighborhood of

the current solution x is defined as follows:

$$V(x) = \{y \in \{0,1\}^n | \sum_{j \in N} a_j y_j \leq b, \sum_{j \in N} |x_j - y_j| = 1\} \tag{13}$$

where the size of $V(x)$ is $|N|$. The intensification phase applies the local search heuristic. The diversification phase follows the principles described above. The initial solution of the tabu search algorithm is determined using the LP-based heuristic.

4. Tabu search using dynamic programming

This section presents our hybrid approach, combining dynamic programming with tabu search. The hybrid consists of three inter-connected phases:

Phase 1 - Initialization: An initial feasible solution is generated and further improved using heuristic procedures. The set of decision variables N is partitioned into two subsets, N' and N''.

Phase 2 - Dynamic programming: A sparse dynamic programming algorithm solves all the sub-problems associated with the subset N'. During the process, reduction rules are applied to set variables to their optimal values, and a list L containing all the optimal values of the sub-problems is stored. The process is repeated in order to set as many variables as possible.

Phase 3 - Tabu search: Tabu search is performed on the search space defined by the subset N'', and uses the list L to complete each partial solution.

Both phases 2 and 3 are repeated until no improvement is possible. This method can be applied to any combinatorial optimisation problems for which an efficient dynamic programming approach exists. The execution of this algorithm is illustrated using the following example $MKP(n = 10, m = 3)$.

$$\begin{aligned}
\max \quad & 20x_1 + 18x_2 + 15x_3 + 14x_4 + 12x_5 + 9x_6 + 7x_7 + 5x_8 + 3x_9 + 2x_{10} \\
s.t. \quad & 15x_1 + 16x_2 + 12x_3 + 12x_4 + 10x_5 + 10x_6 + 8x_7 + 5x_8 + 4x_9 + 3x_{10} \leq 45 \\
& 22x_1 + 21x_2 + 16x_3 + 14x_4 + 15x_5 + 7x_6 + 5x_7 + 2x_8 + 4x_9 + 4x_{10} \leq 50 \\
& 18x_1 + 20x_2 + 15x_3 + 10x_4 + 9x_5 + 8x_6 + 2x_7 + 6x_8 + 2x_9 + 5x_{10} \leq 40 \\
& x_j \in \{0,1\}, j \in N
\end{aligned}$$

Phase 1 - Initialization:

The value of the LP-relaxation is $z(LP) = 51.60$ that corresponds to the LP-solution: $\bar{x} = (0.90, 0, 0, 1, 0.58, 0.07, 1, 1, 0, 0)$.

According to $\bar{x}$, the LP-based heuristic solves the following reduced problem $MKP1$:

$$\begin{aligned} \max \quad & 20x_1 + 12x_5 + 9x_6 \\ s.t. \quad & 15x_1 + 10x_5 + 10x_6 \leq 20 \\ & 22x_1 + 15x_5 + 7x_6 \leq 29 \\ & 18x_1 + 9x_5 + 8x_6 \leq 22 \\ & x_j \in \{0,1\}, j = 1,5,6. \end{aligned}$$

The value $z(MKP1)$ is equal to 21 with the solution (0,1,1). Thus, we obtain the first initial feasible solution:
x^0 = (0, 0, 0, 1, 1, 1, 1, 1, 0, 0) with $c^T x^0 = 47$. The *complement* heuristic is used to improve x^0, producing a new feasible solution x^0:
x^0 = (0, 0, 1, 0, 1, 1, 1, 1, 0, 0) with $c^T x^0 = 48$. Then, the variables x_j are sorted in the decreasing order of the upper bounds u_j, where u_j is the optimal value of the LP-relaxation of $MKP(1 - x_j^0)$.

u_3	u_4	u_1	u_6	u_5	u_8	u_9	u_7	u_{10}	u_2
51.60	51.60	51.60	51.50	51.36	51.11	50.82	49.13	49.06	48.65

Phase 2 - Dynamic programming:

In our experiments we fix n' =4, with $N' = \{3, 4, 1, 6\}$. The following table gives the list L generated during the fourth iteration of the *DP* procedure (recall that v denotes the objective value and $b_i - a_i x$ the slack of the i-th constraint). To simplify the description we omit the element "k" in the list entries.

k	$L = (v; b_1 - a_1 x, b_2 - a_2 x, b_3 - a_3 x)$
1	(0; 45, 50, 40), (15; 33, 34, 25)
2	(0; 45, 50, 40), (14; 33, 36, 30), (15; 33, 34, 25), (29; 21, 20, 15)
3	(0; 45, 50, 40), (14; 33, 36, 30), (15; 33, 34, 25), (20; 30, 28, 22), (29; 21, 20, 15), (34; 18, 14, 12), (35; 18, 12, 7)
4	(0; 45, 50, 40), (9; 35, 43, 32), (14; 33, 36, 30), (15; 33, 34, 25), (20; 30, 28, 22),(23; 23, 29, 22), (24; 23, 27, 17),(29; 20, 21, 14), (29; 21, 20, 15), (34; 18, 14, 12), (35; 18, 12, 7), (38; 11, 13, 7), (43; 8, 7, 4)

The lower bound l_4 on the value of *MKP* is calculated according to (9):
$l_4 = 12x_5^0 + 5x_8^0 + 3x_9^0 + 7x_7^0 + 2x_{10}^0 + 18x_2^0 + \max\{v | \beta' \geq \sum_{j \in N''} a_j x_j^0, (v, \beta') \in L\}$
$l_4 = 24 + \max\{v | \beta' \geq (23, 22, 17), (v, \beta') \in L\} = 24 + 24 = 48$,
since $\max\{v | \beta' \geq (23, 22, 17), (v, \beta') \in L\}$ corresponds to the tuple (24; 23, 27, 17). Moreover, since $l_4 = 48 \geq \lfloor u_{10} \rfloor = 48$, the variable x_2 is set to 0 ($x_2^0 = 0$). When the *DP* algorithm is recalled on the reduced problem, there is no change.

Phase 3 - Tabu search:

The tabu search algorithm is launched on the subset $N'' = N - N' = \{5, 7, 8, 9, 10\}$ and provides the following feasible solution in $\{0,1\}^n$ obtained in four moves:

$x^* = (1, 0, 0, 1, 0, 1, 1, 0, 0, 0)$ with $c^T x^* = 50$.

To illustrate the complete procedure, described in Algorithm 2, let $x'' = (x_5, x_7, x_8, x_9, x_{10}) = (1, 1, 1, 0, 0)$ be the current partial solution. After the drop move $x''_5 = 0$, x'' becomes (0, 1, 1, 0, 0). This move releases the capacities (10, 15, 9). Step 1 of the complete procedure yields the element (29; 21, 20, 15) in the list L, and step 2 generates the optimal solution $x' = (x_3, x_4, x_1, x_6) = (1, 1, 0, 0)$. Thus the complete solution visited is

$x = (x_1, x_3, \ldots, x_{10}) = (0, 1, 1, 0, 0, 1, 1, 0, 0)$

where $c^T x = 41$ $(= 29 + c''^T x'')$.

Phase 2 - Dynamic programming:

As the solution has been improved by the *TS* phase, the process is re-launched, with x^* as the initial solution and $N := N$ - {2} since the variable x_2 is set to 0. The upper bounds u_j are now:

u_8	u_6	u_5	u_9	u_3	u_1	u_4	u_7	u_{10}
51.60	51.50	50.96	50.82	50.82	50.80	50.31	49.13	49.06

The *DP* algorithm is called on the subset $N' = \{8, 6, 5, 9\}$, where the lower bound l_4 is equal to 50. Since $l_4 \geq \lfloor u_5 \rfloor$ (where u_5 is associated to the variable x_3 of the original problem), the algorithm ends as stipulated in Proposition 2.1, and x^* is optimal.

The "pure tabu search" process described in Section 3.3 yields the optimal solution for this small problem in 35 iterations.

5. Computational results

All the implementations were coded in C language, then compiled with gcc option -03, and linked to the CPLEX 6.6 callable library, mainly the simplex and branch-and-bound subroutines. The tests were carried out on three sets of *MKP* instances, using a 300 MHz UltraSparc Solaris 2.6 Work Station with 128 Mo Ram.

The first set is a collection of 270 correlated and thus difficult instances that were generated using the procedure proposed by Fréville and Plateau (1996) (also available in the OR-library). In these instances the coefficients a_{ij} are integer numbers uniformly generated in $U(0, 1000)$. The right-hand side coefficients (b_i's) are set using the formula, $b_i = \alpha \sum_{j \in N} a_{ij}$ where α is the tightness ratio. The objective function coefficients (c_j's) are correlated to a_{ij} as follows: $c_j = \sum_{i \in M} a_i/m + 500\delta_j, j \in N$ where δ_j is a real number uniformly generated in $U(0, 1)$.

The 270 instances were generated by varying combinations of constraints (m = 5, 10, 30), variables (n = 100, 250, 500), and tightness ratios (α = 0.25, 0.5, 0.75), and ten instances have been generated for each of $n - m - \alpha$ combinations (thirty instances for each value of m).

The second set contains 360 large *MKP* instances, randomly generated with a fixed number of constraints ($m = 5$). This data set allows instances with a large number of variables ($n = 500$ up to 5000) and different tightness ratios (α =0.25, 0.50, 0.75). Ten instances were generated for each of $n - \alpha$ combinations. The coefficients a_{ij} are integer numbers uniformly generated in $U(0, 1000)$ and the objective function coefficients (c_j's) were correlated to a_{ij} as follows:

- uncorrelated (A): $c_j \in U(0, 1000)$
- weakly correlated (B): $c_j = 1/m \sum_{i=1}^{m} a_{ij} + \xi$ with $\xi \in U(-100, 100)$
- strongly correlated (C): $c_j = 1/m \sum_{i=1}^{m} a_{ij} + 100$.

Thirty instances were generated for each (n-correlation degree) combination.

The last experiments concern a set of 18 instances proposed by Glover and Kochenberger (1996), with $n = 100$ up to 2500 and $m = 15$ up to 100. These problems are known to be very hard to solve by branch and bound methods. The maximum number of iterations of the *TS* algorithm in the global intensification process is determined in relation to the size of $|N''|$ with a lower and an upper bound (typically between 1000 and 5000 iterations). We compare our solutions to those obtained using CPLEX and those obtained using the "pure" tabu search for the same CPU time as our method. For example, for the instances with $n =$ 100 and $m = 5$ presented in Table 1, the time limit for CPLEX and the "pure" tabu search was fixed at 25 seconds.

Table 1 presents the results obtained using our approach on the first two sets of instances. The first nine lines represent the results for the OR-Library instances, and the last twelve lines represent the results for the randomly generated instances. Each line is an average for 30 problems regrouped by categories. The characteristics of each class of problem (n, m, c_j correlation: or-library for the OR-Library instances, non, weakly or strongly for the randomly generated ones), and the size of the subset N' are given.

The performance of each phase of our method is specified: the initial heuristic phase (*Initial*), the dynamic programming phase (*Dynamic*) and the whole procedure (*Tabu Search*). The results of the reduction procedure (*Reduction*) are also provided. Column *%F1* (respectively *%F2*) presents the average percentage of reduction after the first application of the *DP* process (respectively after the whole procedure). Columns *%G1*, *%G2* and *%G3* show the relative quality gap between the value of the solution obtained by CPLEX and the value of the objective function found at the end of each phase (i.e. Quality_Value = (CPLEX value - our value)*100/(CPLEX value)). Column *%G4* presents the relative gap for the "pure" tabu search algorithm.

The results indicate that the quality of the initial solution is interesting with a reasonable computational effort. This solution is sometimes even better than

the final CPLEX solution for the larger randomly generated instances and the strongly correlated instances. The computational effort associated with each phase (1 and 2) increases with m and the degree of correlation of the instances. The total computational time of our approach increases logically with n, m, and also with the correlation of the data. However, the number of times that the *TS* process is applied influences the total running times of the algorithm, and it depends on the type of instance. On average, *TS* is applied 1.7 times for the OR-Library instances, and 2.1 times for the randomly generated problems. The *DP* process with the reduction technique improves the initial solution (from 0.1% to 0.06%), with an average reduction near 35% of the initial problem. The reduction technique is, however, clearly inefficient for the most correlated instances and for those instances when m increases. The performance of the reduction technique also increases with n for these sets of problems, as well as with the value of α for the OR-Library instances. The same conclusions can be drawn for the quality of our solutions at each step of the approach.

Overall, the solution quality increases from 0.1% at the end of the initial heuristic phase to 0.01% at the end of the tabu search. This underlines the positive impact of the global intensification on solution quality. Solutions obtained using CPLEX are better those obtained with *TS* when n is small, but *TS* provides better solutions when the size of instances increases. In fact, when the size of the problems grows, CPLEX has memory problems. The reduction process is also very efficient for some classes of problems (sometimes higher than 80%) which allows the tabu search algorithm to explore a small search space. Generally, for the 270 OR-Library instances, we obtain the same solution as CPLEX 99 times, and we improve the CPLEX solution for 26 problems (i.e. for 46.3% of the instances). For the 360 randomly generated instances, we obtain 268 solutions at least as good as the CPLEX solutions (about 74%). On average, the quality of our final solutions is very near the CPLEX solutions (0.01%), and the approach appears to be robust given the results obtained for the large instances. These results clearly show that our approach is very efficient for large instances, particularly when $m << n$.

The "pure" tabu search algorithm obtained generally worse solutions than the global intensification mechanism, except for 21 OR-Library instances (7.8%) and 38 randomly generated instances (10.6%). These 59 instances represent a small part of the data sets. Moreover, the results on the randomly generated instances were less robust than those obtained using the global intensification process.

Table 2 presents the computational results obtained on the last 18 instances, proposed by Glover et Kochenberger (1996). This set can be divided into two groups, the first one containing seven problems from GK18 to GK24 and the second containing eleven instances from MK_GK01 to MK_GK11. For all these instances, the reduction process is completely ineffective since no variable could be fixed. This confirms previous conclusions about the reduction process. The first three columns of Table 2 present the characteristics of each problem: name,

Table 1. Computational results on the OR-Library and randomly generated instances

Problem		c_j	$\|N'\|$	Initial solution		Dynamic programming		Reduction		Tabu search		Pure Tabu
n	m	*correlation*		*%G1*	*T1*	*%G2*	*T2*	*%F1*	*%F2*	*%G3*	*T3*	*%G4*
100	5		15	0.34	0	0.08	0.06	25.53	32.30	0	22.5	0.01
100	10		15	0.55	0.02	0.27	0.08	5.10	8.57	0.01	38.9	0.02
100	30		13	0.27	0.87	0.26	0.92	0.17	0.27	0.06	31.07	0.02
250	5		15	0.15	0	0.1	0.07	30.20	49.87	0.01	65.17	0.03
250	10	*or−library*	15	0.24	0.02	0.17	0.09	4.70	11.87	0.06	113.7	0.09
250	30		13	0.10	2.76	0.10	2.82	0.03	0.03	0.07	109.7	0.09
500	5		15	0.06	0.02	0.04	0.09	39.20	57.80	0.01	138.37	0.05
500	10		15	0.11	0.05	0.08	0.11	5.87	13.50	0.03	414.77	0.12
500	30		13	0.04	6.01	0.04	6.35	0	0	0.03	339.4	0.04
500	5	*non*	15	0.05	0.02	0.03	0.56	74.67	81.33	0.002	39.33	0.02
		weakly		0.06	0.03	0.06	0.91	17.33	37.33	0.01	244.33	0.06
		strongly		0.06	0.03	0.02	1.10	1	5.33	-0.002	307.67	0.02
1000	5	*non*		0.02	0.05	0.01	1.66	79.33	84	0.001	79.33	0.02
		weakly		0.02	0.08	0.01	2.65	22	46	-0.001	829	0.04
		strongly		0.02	0.09	0.001	3.80	1	4.67	-0.01	1463	-0.005
2000	5	*non*		0.01	0.16	0.007	6.00	81.00	87.33	0.000	211.67	0.01
		weakly		0	0.25	0.004	9.62	33.00	51.67	-0.002	1813.33	0.02
		strongly		0.01	0.28	-0.001	14.51	2.33	7.33	-0.006	1605	-0.006
5000	5	*non*		0.002	0.97	0.002	35.38	83.33	89	-0.02	803.67	0.005
		weakly		-0.002	1.53	-0.002	47.26	47	55.33	-0.003	1612	0.002
		strongly		-0.002	1.78	-0.003	86.86	2.67	4	-0.005	1428.67	-0.005
Average				0.1	0.7	0.061	10.5	26.45	34.64	0.012	557.6	0.031
Worst case				0.547	6.0	0.270	86.9	0	0	0.07	1813.3	0.117

number of variables and number of constraints. Then, the size of N', the value obtained by CPLEX (cx^*) and the results of our approach (*Initial Solution*, *Dynamic Programming* and *Tabu Search*) are presented, as in Table 1. The last column presents the solution quality of the "pure" tabu search algorithm.

The initial solution quality for these 18 instances varies quite clearly, ranging from far from the best solution (for example MK_GK001) to very close (for example MK_GK006). The total time needed to obtain this initial solution increases greatly with m (in particular for MK_GK010 and MK_GK011), so much so that we had to stop the process before the end. We chose to limit this process to 200 seconds since it is not the most important phase in the global intensification mechanism.

The dynamic programming phase is able to improve the initial solution for part of these problems and the time needed is not excessive, except for the last instance in which it became prohibitive due to the number of constraints. However, the *DP* process was clearly efficient in terms of computational time for instances with few constraints.

The quality of the final solution is good; it is better than CPLEX in eight instances, the same in one instance, and worse in the eight last instances. This confirms the idea that for the instances with high n values, the approach is more effective than CPLEX. The "pure" tabu search process was less effective than the global intensification mechanism, except for problem GK18 for which a better solution was obtained than with CPLEX. For problems MK_GK04 and MK_GK06, the "pure" tabu search obtained the same solutions as *TS*, and comes quite near for the larger problems.

In addition to the experimental results related above, we also applied our algorithm to randomly generated problems with other numbers of variables and constraints in order to study the evolution of our results. The results obtained tend to confirm those discussed above. We also compared our results to those of Vasquez and Hao (2001) for the last set of 18 instances. The average gap value is equal to 0.05%. However, the comparison is difficult to make given that the same computer was not used, and the computational times were not of the same order (their days compared to our hours) for the larger instances.

6. Conclusion

We have proposed a hybrid approach combining dynamic programming with tabu search to solve the zero-one multidimensional knapsack problem. Starting with a bi-partition of the variables, dynamic programming was applied to solve a family of sub-problems associated with the first subset of these partitions, using only the forward phase of the dynamic programming. Next, a tabu search was performed in the subspace associated to the remaining set of variables, where each partial feasible solution encountered is completed by calling the backtracking phase of dynamic programming. Reduction techniques were incorporated to enhance the dynamic prgramming phase. The overall process was repeated until

Table 2. Gap values and running times for the Glover and Kochenberger instances

Problem			$\|N'\|$	Cplex	Initial phase		Dynamic Programming		Tabu search		Pure tabu
Name	n	m		cx^*	*%G1*	*T1*	*%G2*	*T2*	*%G3*	*T3*	*%G4*
MK_GK01	100	15	14	3766	0.35	0.1	0.19	0.2	0.13	11.9	0.19
GK18	100	25	14	4522	0.04	0.9	0.04	1.5	0.02	16.2	-0.04
GK19	100	25	14	3867	0.18	0.5	0.18	0.7	0.10	15.1	0.18
GK20	100	25	14	5177	0.04	0.8	0.04	0.9	0.02	13.2	0.02
GK21	100	25	14	3199	0.16	0.6	0.16	0.8	0.13	14.0	0.16
GK22	100	25	14	2521	0.16	0.3	0.16	0.4	0.12	13.4	0.16
MK_GK02	100	25	14	3957	0.20	0.5	0.05	0.4	-0.03	11.2	0.20
MK_GK03	150	25	14	5649	0.16	1.4	0.14	0.7	0.00	23.3	0.16
MK_GK04	150	50	13	5763	-0.02	59.3	-0.02	1.9	-0.02	14.3	-0.02
GK23	200	15	14	9233	0.08	0.2	0.08	1.1	0.03	24.2	0.08
MK_GK05	200	25	14	7558	0.07	1.1	0.07	1.0	0.04	29.3	0.07
MK_GK06	200	50	13	7666	-0.07	177.4	-0.07	3.2	-0.07	21.1	-0.07
GK24	500	25	14	9064	0.08	0.8	0.08	2.5	0.07	139.7	0.11
MK_GK07	500	25	14	19209	0.02	0.6	0.02	5.3	-0.02	137.9	0.02
MK_GK08	500	50	13	18789	0.09	200	0.07	15.8	-0.05	111.9	-0.04
MK_GK09	1500	25	14	58080	0.01	1.5	0.00	36.1	-0.01	900.8	0.01
MK_GK10	1500	50	13	57277	0.24	200	0.24	126.4	-0.02	908.5	-0.01
MK_GK11	2500	100	12	95209	0.05	200	0.05	1943.3	-0.01	929.2	-0.01

no improvement is obtained. In the context of tabu search, this hybrid approach can be seen as a global intensification strategy, since each move involves solving a reduced problem. The effectiveness of the proposed approach was demonstrated through numerical experiments, and the results were compared to those obtained with a "pure tabu search" and CPLEX. These numerical experiments show that our approach is robust and able to produce high-quality solutions for large-scale instances in a reasonable amount of CPU time, particularly for problems with few constraints. This method seems to be appropriate for a class of combinatorial optimization problems, whose the structure is adapted to dynamic programming treatment. The tabu search algorithm could also be improved by introducing a *Reverse Elimination Method* strategy for tabu list management, for example.

Acknowledgement

We wish to thank the referees for their valuable suggestions and help in improving the paper. This work was partially supported by the Région Nord-Pas-de-Calais, and by the European funding program FEDER. The support is gratefully acknowledged.

References

Andonov, R. Balev, S. Fréville, A. and Yanev, N. (2001) A dynamic programming based reduction procedure for the multidimensional 0-1 knapsack problem. FRANCORO 2001, Québec, submitted to *European Journal of Operational Research*.

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

Bertsimas, D. and Demir, R. (2002) An Approximate Dynamic Programming Approach to Multidimensional Knapsack Problems. *Management Science* **48**, 550–565.

Chu, P. and Beasley, J. (1998) A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics* **4**, 63–86.

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

Fréville, A. and Hanafi, S. (2005) The multidimensional 0-1 knapsack problem - Bounds and computational aspects. IN: M. Guignard and K. Spielberg, eds., *Advances in Operations Research. Annals of Operations Research*, **139**(1), 195-197.

Fréville, A. and Plateau, G. (1990) Hard 0-1 multiknapsack test problems for size reduction methods. *Investigacion Operativa* **1**, 251–270.

Fréville, A. and Plateau, G. (1994) An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Applied Mathematics* **2** (2), 147–167.

Fréville, A. and Plateau, G. (1996) The 0-1 bidimensional knapsack problem: towards an efficient high-level primitive tool. *Journal of Heuristics* **2**, 147–167.

Glover, F. (1986) Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* **19**, 533–549.

Glover, F. (1989) Tabu Search-Part I. *ORSA Journal on Computing* **1** (3), 190–206.

Glover, F. (1990) Tabu Search-Part II. *ORSA Journal on Computing* **2** (1), 4–32.

Glover, F. and Kochenberger, G. (1996) Critical event tabu search for multidimensional knapsack problems. In: *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, 407–427.

Glover, F. and Laguna, M.(1997) *Tabu Search*. Kluwer Academic Publishers, Dordrecht.

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

Hansen, P. (1986) The steepest ascent mildest descent heuristic for combinatorial programming, *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.

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

Martello, S. and Toth, P. (1990) Knapsack Problems: Algorithms and Computer Implementations. *Series in Discrete Mathematics and Optimization*. Wiley Interscience.

Osorio, M.A. Glover, F. and Hammer, P. (2002) Cutting and Surrogate Constraint Analysis for Improved Multidimensional Knapsack Solutions. *Annals of Operational Research* **117**, 71-93.

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

Savelsberg, M.W.P. (1994) Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal of Computing* **6**, 445–454.

Soyster, A.L. Lev, B. and Slivka, W. (1978) Zero-one programming with many variables and few constraints. *European Journal of Operational Research* **2**, 195–201.

Toth, P. (1980) Dynamic programming algorithms for the zero-one knapsack problem. *Computing* **25**, 29–45.

Vasquez, M. and Hao, J.K. (2001) Une approche hybride pour le sac à dos multidimensionnel en variables 0-1. *RAIRO Operations Research* **35**, 415–438.

Viader, F. (1998) *Méthodes de Programmation Dynamique et de Recherche Arborescente pour l'Optimisation Combinatoire: Utilisation Conjointe des deux approches et Parallélisation d'Algorithmes.* Ph.D. Thesis, Université Paul Sabatier, Toulouse, France.