Ann Oper Res (2011) 183: 125–142
DOI 10.1007/s10479-009-0546-z

Improved convergent heuristics for the 0-1 multidimensional knapsack problem

Saïd Hanafi · Christophe Wilbaut

Published online: 27 May 2009

Abstract At the end of the seventies, Soyster et al. (Eur. J. Oper. Res. 2:195–201, 1978) proposed a convergent algorithm that solves a series of small sub-problems generated by exploiting information obtained through a series of linear programming relaxations. This process is suitable for the 0-1 mixed integer programming problems when the number of constraints is relatively smaller when compared to the number of variables. In this paper, we first revisit this algorithm, once again presenting it and some of its properties, including new proofs of finite convergence. This algorithm can, in practice, be used as a heuristic if the number of iterations is limited. We propose some improvements in which dominance properties are emphasized in order to reduce the number of sub problems to be solved optimally. We also add constraints to these sub-problems to speed up the process and integrate adaptive memory. Our results show the efficiency of the proposed improvements for the 0-1 multidimensional knapsack problem.

Keywords Relaxation · Heuristic · Multidimensional knapsack problem

1 Introduction

This paper considers the 0-1 Multidimensional Knapsack Problem (MKP), which seeks to find a subset of items that maximizes a linear objective function while satisfying the capacity constraints. This problem can be formulated as follows:

$$(\text{MKP})\left[\begin{array}{lll} \max\ \sum_{j=1}^{n} c_j x_j & & \\ \text{subject to:} & \sum_{j=1}^{n} a_{ij} x_j \le b_i, & \forall i \in M = \{1,\dots,m\} \\ & x_j \in \{0,1\}, & j \in N = \{1,\dots,n\} \end{array}\right.$$

S. Hanafi (✉) · C. Wilbaut
LAMIH-SIADE, UMR 8530, Université de Valenciennes et du Hainaut-Cambrésis, Le Mont Houy, 59313 Valenciennes Cedex 9, France
e-mail: said.hanafi@univ-valenciennes.fr

C. Wilbaut
e-mail: christophe.wilbaut@univ-valenciennes.fr

where N is the set of items, c_j is the profit of item $j \in N$, M is the set of knapsack constraints, b_i is the capacity of knapsack $i \in M$, and a_{ij} is the consumption of resource $i \in M$ by item j. All values c_j, b_i and a_{ij} are non-negative integers. Without loss of generality, we can assume that $\max_{j \in N} a_{ij} \le b_i \le \sum_{j \in N} a_{ij}$, $\forall i \in M$. For reasons of simplicity, we use the following shortcut notation for the problem:

$$\text{(MKP)} \quad \max\{cx : Ax \le b, x \in \{0,1\}^n\}. \tag{1}$$

The MKP is a special case of 0-1 integer programming; it is known to be NP-hard, but not strongly NP-hard. This problem 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 and Fréville and Hanafi 2005 for a comprehensive annotated bibliography). In particular, the MKP has been shown to become significantly harder to solve as m increases. Some very efficient algorithms (Kellerer et al. 2004; Martello and Toth 1990) do exist when $m = 1$, but as m increases, exact methods, including the latest versions of CPLEX, usually fail to provide an optimal solution for even moderate-size instances. Very efficient approaches based on the tabu search technique have been developed for the MKP (e.g., Glover and Kochenberger 1996 or Hanafi and Fréville 1998), and most of the best-known solutions for the instances in the OR-Library (Beasley 1990) were obtained by Vasquez and Hao (2001) and Vasquez and Vimont (2005). Other recent methods have obtained encouraging results by making a compromise between solution quality and computational effort. For example, Puchinger et al. (2006) proposed an extension of the classic core concept Pisinger (1995) for the MKP and also described an extension of the metaheuristic variable neighborhood search (Hansen and Mladenovic 2001) used with a branch-and-cut algorithm. In addition, Lichtenberger (2005) applied the local branching framework (Fischetti and Lodi 2003) to the MKP, producing results that show that integrating this method into the open source software COIN-OR is a good way forward.

At the end of the seventies, Soyster et al. (1978) proposed a convergent algorithm for pure 0-1 integer programming that solves a series of small sub-problems generated by exploiting information obtained through a series of linear programming relaxations. Another approach for generating and exploiting small sub-problems was also suggested by Glover (1977), whose ideas we will also make use of in parts of our development. In this proposal, a heuristic approach is applied in a series of passes to identify variables that qualify as strongly determined and consistent, by reference to how frequently they attain particular values in good solutions and by how much disruption they would cause to these solutions if changed. Upon identifying a subset of variables of this form, the method fixes (or bounds) the selected variables to receive their associated values, and then repeats the process, thereby iteratively identifying additional variables to be fixed. The possibility of using differing criteria for identifying solutions that qualify as good, and for setting the thresholds that determine which variables qualify as strongly determined and consistent gives a basis for repeating the process and adapting it to different problem structures.

More recently, building in part on these ideas, Glover (2005) proposed an adaptive memory projection (AMP) method for pure and mixed integer programming, which combines the principle of projection techniques with the adaptive memory processes of tabu search to set some explicit or implicit variables to some particular values. This approach, whose ideas we also draw upon subsequently, gives a useful basis for unifying and extending a variety of other procedures. For example, it gives an opportunity to bring together innovative contributions underlying methods such as the Large Neighborhood Search (LNS) introduced by Shaw (1998), the large-scale neighbourhood search proposed by Ahuja et al. (2002), the local branching proposed by Fischetti and Lodi (2003), the relaxation induced neighbourhood search (RINS) proposed by Danna et al. (2005), or the global tabu search intensification using dynamic programming (TS-PD) proposed by Wilbaut et al. (2006). LNS and RINS have been applied successfully to solve large-scale mixed integer programming problems. TS-PD is a hybrid method, combining adaptive memory and sparse dynamic programming to explore the search space, in which a move evaluation involves solving a reduced problem through dynamic programming at each iteration.

Thus, exact methods designed to find the optimal value have been successfully applied to problems of small dimensions, but they have not been able to produce high quality solutions with a reasonable computational effort for problems of a moderate size. Large-scale problems require good approximations of the optimal value, and both heuristic and relaxation methods have proved useful for providing the upper and lower bounds of the optimal value for large and difficult optimization problems. In this paper, we propose several convergent algorithms, integrating both heuristic and relaxation methods, for solving the MKP. Section 2 of this paper briefly describes the iterative linear programming-based heuristic on which we base our approach. We provide a new proof of the algorithm's finite convergence in Sect. 3. Then in Sect. 4 we describe several enhancements for this algorithm based on the definition of the concept of dominance to solve fewer problems optimally. We also describe other ways to obtain a new algorithm with other improvements. The computational results are reported in Sect. 5, and our conclusions and suggestions for future research are offered in Sect. 6.

2 Iterative linear programming-based heuristic

Soyster et al. (1978) proposed an exact algorithm to solve 0-1 integer programs. As we show in Sect. 3, this algorithm cannot be used as an exact algorithm for large instances in practice. We therefore use the same procedure but limit the number of iterations it is allowed to perform, and we call this heuristic the Iterative Linear Programming-based Heuristic (ILPH). The ILPH optimally solves a series of small sub-problems obtained from a series of linear programming relaxations. The approach proceeds as follows. At each iteration, the LP-relaxation of the current problem P is solved to generate one constraint. Then, a reduced problem induced from an optimal solution of the LP-relaxation is solved to obtain a feasible solution for the initial problem. If the stopping criterion is satisfied, then the best lower bound and the best upper bound are returned. Otherwise, a pseudo cut is added to P and the process is repeated. Our solution offers improvements on this algorithm.

To describe the ILPH, we introduce the notion of reduced problems, which are obtained from the original problem by setting several variables at given values. Given a binary solution $x^0 \in \{0,1\}^n$ and a subset $J \subseteq N$, the reduced problem associated with x^0 and J can be formally defined as follows:

$$P(x^0, J)\ \max\{cx : Ax \leq b, x_j = x_j^0, \forall j \in J, x \in \{0,1\}^n\}. \tag{2}$$

Obviously, $P(x^0, \oslash) = P$ and $v(P(x^0, J')) \leq v(P(x, J))$ for any subsets J and J' of N with $J \subseteq J'$, where $v(P)$ is the optimal value of the optimization problem P. The notion of reduced problem can be quite useful in a variety of different contexts. For example, at each iteration of the simplex method of linear programming, the pivot move solves a reduced problem where only a single variable is free, here $|J| = n - 1$.

Throughout the rest of the paper, the following notations are used. Let $x \in [0,1]^n$, $J^0(x) = \{j \in N : x_j = 0\}$, $J^1(x) = \{j \in N : x_j = 1\}$, $J^*(x) = \{j \in N : x_j \in]0,1[\}$ and

$J(x) = \{j \in N : x_j \in \{0, 1\}\}$ (i.e. $J(x) = J^0(x) \cup J^1(x)$). Let P be an optimization problem and Q be a set of constraints. The notation $(P|Q)$ denotes the optimization problem obtained from P by adding the set of constraints Q.

The ILPH restricts the search process to visiting optimal LP-solutions already generated by adding a pseudo-cut at each iteration according to the following propositions.

Proposition 1 *Let x^0 be a vector in $\{0, 1\}^n$. The following inequality*

$$\sum_{j \in J^1(x^0)} x_j - \sum_{j \in J^0(x^0)} x_j \le |J^1(x^0)| - 1 \tag{3}$$

cuts off solution x^0 without cutting off any other solution in $\{0, 1\}^n$.

Proof For any solution $x \neq x^0$ we have:

$$\|x - x^0\|_1 > 0. \tag{4}$$

Using the definition of the norm $\| . \|_1$, the strict inequality (4) can be expressed as follows:

$$\sum_{j \in J^1(x)^0} |1 - x_j| - \sum_{j \in J^0(x^0)} |x_j| > 0. \tag{5}$$

Since the solutions x are contained in $\{0, 1\}^n$, the inequality (5) is equivalent to (3). This completes the proof. □

Proposition 2 *Given a 0-1 integer program P, let $\bar{x}$ be an optimal solution of the LP-relaxation LP(P) and x^0 be an optimal solution for the reduced problem $P(\bar{x}, J(\bar{x}))$. Thus, an optimal solution for P is either the feasible solution x^0 or an optimal solution for the problem*

$$(P \mid \{fx \le |J^1(\bar{x})| - 1\}) \tag{6}$$

where the vector f of dimension n is defined for $j = 1, \dots, n$ as

$$f_j = \begin{cases} 2\bar{x}_j - 1 & \textit{if } \bar{x}_j \in \{0, 1\} \\ 0 & \textit{if } \bar{x}_j \in]0, 1[\end{cases}.$$

Proof It is easy to note that for each solution x in $\{0, 1\}^n$, $fx \le |J^1(\bar{x})|$. This inequality could be divided into two inequalities for every solution x for the problem P:

$$fx < |J^1(\bar{x})| \tag{7}$$

or

$$fx = |J^1(\bar{x})|. \tag{8}$$

Every solution satisfying the constraint (8) is considered in the reduced problem $P(\bar{x}, J(\bar{x}))$ and could be eliminated in the next iteration using Proposition 1. Since the variables are binary and the coefficients of the constraint (7) are integers, this constraint could be reformulated as $fx \le |J^1(\bar{x})| - 1$. This completes the proof. □

Algorithm 1 Iterative Linear Programming-based Heuristic (ILPH)

```
Require: Instance $P$ of the MKP.
Ensure: An optimal solution $x^*$ of $P$.
 1: Let $x^*$ be a feasible solution of $P$ if one is available;
 2: $Q = P$; stop = False;
 3: while stop = False do
 4:     Solve the LP-relaxation of $Q$ to obtain an optimal solution $\bar{x}$;
 5:     if $\bar{x} \in \{0,1\}^n$ then
 6:         $x^* = \bar{x}$; stop = True;
 7:     end if
 8:     Generate an optimal solution $x^0$ of the reduced problem $P(\bar{x}, J(\bar{x}))$;
 9:     Update the best known-solution: if $cx^0 > cx^*$ then $x^* = x^0$;
10:     Generate the current cut $\{fx \leq |J^1(\bar{x})| - 1\}$ according to (6)
11:     Update the current problem $Q$ by adding the above constraint:
                    $Q = (Q \mid \{fx \leq |J^1(\bar{x})| - 1\})$
12:     Check stopping criteria: if $\lfloor c\bar{x} - cx^* \rfloor < 1$ then stop = True;
13: end while
14: Return the best solution $x^*$ of $P$ if one is generated;
```

Balas and Jeroslow (1972) call the constraints added to the problem (6) canonical cuts on the unit hypercube $K = \{x \in R^n : 0 \leq x_j \leq 1, j = 1, \dots, n\}$. The inequalities in (6) have also been used, for example, to produce 0-1 "short hot starts" for branch-and-bound methods by Spielberg and Guignard (2000) and Guignard and Spielberg (2003) and were used by Glover (2005) in AMP in diversification and intensification phases.

Algorithm 1 shows the basic steps of the ILPH. At each iteration, the algorithm solves the LP-relaxation of the current problem, Q, generating the optimal solution $\bar{x}$ (line 4). From this optimal solution, the associated reduced problem $P(\bar{x}, J(\bar{x}))$ is solved exactly to generate a feasible solution x^0 for the original problem P (line 8). The current problem Q is then enriched by a pseudo-cut to avoid generating the optimal basis of the LP-relaxation more than once (line 11). The process stops if the difference between the upper and the lower bounds is less than 1 (line 12). Assuming that all data are integers, if the condition $\lfloor c\bar{x} - cx^* < 1 \rfloor$ is satisfied, then the solution x^* corresponding to the lower bound is an optimal solution for the problem P (where, for a real number α, $\lfloor \alpha \rfloor$ identifies the greatest integer $\leq \alpha$).

3 Convergence of the iterative linear programming-based heuristic

The following theorem states that, when the ILPH terminates, the best solution found is an optimal solution for the initial problem.

Theorem 1 *Given an instance P of MKP, the final best solution obtained by the ILPH is an optimal solution for P. Moreover, if the optimal solution for the LP-relaxation $LP(P^k)$ is an integer solution or if the problem $LP(P^k)$ is infeasible, then the ILPH terminates (where P^k denotes the current problem at iteration k).*

Proof Let F^k be the union of all the feasible sets of the reduced problems solved exactly up to the current iteration k. More specifically, $F^k = \bigcup_{i=1}^{k} F(P(\bar{x}^i, J^*(\bar{x}^i)))$, where $F(P)$ is the feasible set of the optimization problem (P).

The set F^k and the set of the feasible solutions for the problem P^k constitute a partition of the feasible set of P (i.e., $F(P) = F^k \cup F(P^k)$ and $F^k \cap F(P^k) = \oslash$). Let x^{*k} be the final solution returned by the algorithm ILPH when it stops at iteration k. Thus, $cx^{*k} = \max\{cx : x \in F^k\}$. Moreover, $v(P) = \max\{cx : x \in F(P)\} = \max\{cx : x \in F^k \cup F(P^k)\}$, yielding

$$v(P) = \max\{cx^{*k}; \max\{cx : x \in F(P^k)\}\}. \tag{9}$$

Since $v(LP(P^k)) \geq \max\{cx : x \in F(P^k)\}$ and since all the data for P are integers, the stopping condition of the ILPH (i.e. $v(LP(P^k)) - cx^{*k} < 1$) implies that the final solution x^{*k} is an optimal solution for P. Moreover, according to (9), if the optimal solution of the LP-relaxation $LP(P^k)$ is an integer or the problem $LP(P^k)$ is infeasible, then the solution x^{*k} is an optimal solution for P. This completes the proof. □

Theorem 2 states the finite convergence of the ILPH.

Theorem 2 *The ILPH converges to an optimal solution for the problem or indicates that the problem is infeasible in a finite number of iterations.*

Proof Let us define a partial solution as a vector in which some of the components have not yet been assigned. Such partial solutions are assumed to include incomplete or complete solutions. More specifically, a partial solution x of order k is a vector for which exactly $n-k$ of the variables are assigned to values 0 or 1. There are 2^n partial solutions of order zero (i.e., 0-1 solution in $\{0, 1\}^n$) and there is one partial solution of order n (i.e., null solution). More specifically, there are $\binom{n}{k}2^{n-k}$ partial solutions of order k. Thus the total number of partial solutions is $3^n = (1+2)^n = \sum_{k=0}^{n}\binom{n}{k}2^{n-k}$. At each iteration, the ILPH generates a partial solution from the optimal solution $\bar{x}$ of the current LP-relaxation $LP(P)$, which is completed by solving the corresponding reduced problem exactly. The next iteration cuts off this partial solution by adding the constraint (6). Since there is a finite number of partial solutions, the number of iterations of the ILPH is bounded by 3^n iterations. Moreover, according to Theorem 1, the ILPH terminates as soon as a partial solution of order zero is found, so the number of iterations cannot exceed $3^n - 2^n$. This completes the proof. □

The proofs of Theorems 1 and 2 remain valid for the general 0-1 integer programs, which implies the finite convergence of the ILPH for the 0-1 integer programs. This validity can be extended to general 0-1 mixed integer programs, assuming that the optimal value of the LP-relaxation $v(LP(P^k)) = -\infty$ if the problem $LP(P^k)$ is infeasible.

The ILPH was first tested on moderate size instances of MKP to evaluate its convergence. We report in Table 1 the results obtained for two MKP problems. We give for some iterations (*iter*) the value of the linear programming relaxation $v(LP(P))$; the value of the feasible solution generated cx^0; the number of fractional variables associated in the solution of the linear programming relaxation which corresponds to the size of the reduced problem, $|J^*(\bar{x})|$. The first problem has 30 variables and 10 constraints. Its optimal value is equalled to 376. The process is very fast and an optimal solution is obtained with 13 iterations (see Table 1). This solution is proved to be optimal at iteration 13 since the optimal condition expressed at line 12 of Algorithm 1 is satisfied. When the algorithm was launched on the second problem with 100 variables and 5 constraints (OR-100-5.3 in the OR-Library), an optimal solution was obtained after 41 iterations; after 100 iterations, the difference between the lower and the upper bounds was about 0.5%. In the end, 292 iterations were necessary to prove the solution's optimality. The process is distinctly slower. The results obtained for

Table 1 Convergence illustration

GK9				OR-100-5.3			
iter	$v(LP(P))$	cx^0	$\lvert J^*(\bar{x})\rvert$	*iter*	$v(LP(P))$	cx^0	$\lvert J^*(\bar{x})\rvert$
1	380.3	336	6	1	23724.1	22554	5
2	379.7	368	7	2	23722.7	22983	5
3	379.6	360	8	3	23720.3	23056	7
4	379.5	368	9	4	23715.5	22606	6
5	378.7	368	10	40	23687.4	23447	18
8	377.9	368	10	41	23687.3	**23534**	20
9	377.6	372	10	42	23686.8	23402	16
10	377.2	368	11	98	23652.3	23497	23
11	**376.9**	372	13	99	23651.6	23486	23
12	376.5	364	7	100	23651.1	23497	24
13	376.4	**376**	9	292	**23534.6**	23497	46

this problem clearly show that the convergence is not easily reachable in practice. The total running time of the algorithm was about 400 seconds on our computer (see Sect. 5 for a description of the computer's characteristics), whereas the branch-and-bound algorithm required only a fraction of this time, a mere handful seconds, to obtain an optimal solution. This example seems to indicate that the ILPH can not be used easily as an exact method. Notice also that the size of the reduced problem does not necessarily increase between two iterations and that the increase is usually observed to be slow. Finally note that the ILPH could generate the same feasible solution several times but it has the advantage of generating good lower bounds in a small number of iterations.

In order to increase it's effectiveness, we decided to use the ILPH as a heuristic, replacing the stopping criterion with a maximum number of iterations. In the following section, we propose enhancements for the ILPH designed to accelerate the process and to reduce the gap between the lower bound and the upper bound of the problem when the ILPH is used as a heuristic. These enhancements are based on dominance properties and the use of adaptive memory.

4 Enhancements of the ILPH

In practice, most of the time needed to execute the ILPH is consumed by the exact method for solving the reduced problems. The linear programming relaxation does not, in fact, require much effort thanks to advances in the commercial solvers (as long as the size of the problem remains reasonable). However, neither the commercial solvers nor the existing exact algorithms in the literature are able to solve exactly moderate-sized reduced problems with reasonable computational effort. This is the case both for the MKP and for other optimization problems. For example, the reduced problems obtained from instances of the MKP with $n = 500$ and $m = 30$ are very difficult to solve with CPLEX v9, even at the first iterations of the ILPH. In order to reduce the total number of exact optimization procedures over reduced problems and hence to accelerate the search, we propose the use of dominance properties.

4.1 Reducing the number of reduced problems

Our dominance properties depend on the subsets $J(\bar{x})$ associated with the solutions for the linear programming relaxations obtained during the solution process.

Definition 1 Let x^1 and x^2 be two solutions in $[0,1]^n$, we say that solution x^1 dominates solution x^2 if, for all $j \in J(x^1)$, $x^1_j = x^2_j$.

For example, solution $x^1 = (1100{*}{*})$ dominates solution $x^2 = (11001{*})$, where "*" indicates a fractional variable in a solution. Note that given two solutions x^1 and x^2, one of them will not always dominate the other (e.g., $x^1 = (1110{*}{*})$ and $x^2 = (0001{*}{*})$). The following proposition describes the dominance property associated with Definition 1.

Proposition 3 *Let x^1 and x^2 be two solutions in $[0,1]^n$. If solution x^1 dominates solution x^2 according to Definition 1, then*

$$v(P(x^1, J(x^1))) \geq v(P(x^2, J(x^2))). \tag{10}$$

Proof According to Definition 1, if solution x^1 dominates x^2, then $J(x^1) \subseteq J(x^2)$. In addition, the integer values in solution x^1 are the same as in solution x^2. Thus, the inequality (10) is obtained due to the fact that the problem $P(x^1, J(x^1))$ is a relaxation of $P(x^2, J(x^2))$. □

An obvious implication of Proposition 3 is that it is not necessary to solve the reduced problems corresponding to the dominated optimal solutions of the LP-relaxation.

To integrate the dominance property into the ILPH, we propose a new two-phase algorithm described in Algorithm 2. In the first phase, only one series of LP-relaxations is solved (lines 3 to 8); and in the second phase, the reduced problems associated with the undominated LP-solutions are solved exactly (lines 9 to 12). To implement this algorithm, a list L, containing only the solutions associated with the undominated reduced problems, is maintained. However, all the optimal solutions of the LP-relaxations are not stored; only the indices corresponding to the fractional variables and the variables set at 1 are included in the

Algorithm 2 Iterative Linear Programming-based Heuristic with dominance

```
Require: Instance $P$ of the MKP; The maximum number of iterations, $Max\_Iter$
Ensure: A feasible solution $x^*$ of $P$.
1:  Let $x^*$ be a feasible solution of $P$ if one is available;
2:  $v^* = cx^*$; $L = \emptyset$; $Q = P$; $iter = 1$;
3:  while $iter \leq Max\_Iter$ do
4:      Let $\bar{x}$ be an optimal solution of $LP(Q)$;
5:      $Q = (Q \mid \{fx \leq |J^1(\bar{x})| - 1\})$
6:      if $D^-(\bar{x}) = \emptyset$ then $L = L + \bar{x} - D^+(\bar{x})$
7:      $iter = iter + 1$;
8:  end while
9:  for every solution $\bar{x} \in L$ do
10:     Let $x$ be an optimal solution of the reduced problem $P(\bar{x}, J(\bar{x}))$
11:     if $cx > cx^*$ then $x^* = x$; $v^* = cx$;
12: end for
13: Return the best solution $x^*$ of $P$ if one is generated;
```

list. This list can be sorted according to the size of the fractional subsets of the LP-solutions in order to accelerate the detection of the dominated solutions. The dominated solutions are detected using the set $D^+(\bar{x})$ (resp. $D^-(\bar{x})$), representing the set of the elements in the list L dominated by (resp. which dominate) the solution $\bar{x}$:

$$D^+(\bar{x}) = \{y \in L : \bar{x} \text{ dominates } y\},$$
$$D^-(\bar{x}) = \{y \in L : y \text{ dominates } \bar{x}\}.$$

Thus, a solution $\bar{x}$ is added to L if and only if $D^-(\bar{x}) = \emptyset$ (line 6).

This version of the ILPH stops after a maximum number of iterations (the algorithm parameter *Max_Iter*). As shown in Sect. 5, integrating the dominance property results in a smaller number of reduced problems to be solved exactly for most of the instances tested.

The following proposition extends this dominance property.

Proposition 4 *Let x^1 and x^2 be two solutions in $[0,1]^n$; the solution $y = \frac{x^1+x^2}{2}$, dominates solutions x^1 and x^2, thus*

$$v(P(y, J(y))) \geq \max\{(v(P(x^1, J(x^1))), v(P(x^2, J(x^2)))\}. \tag{11}$$

Proof It is easy to check that solution y dominates solutions x^1 and x^2. Inequality (11) follows directly from Proposition 3. □

From a practical standpoint to use the Proposition 4, we define a parameter t corresponding to the maximum number of fractional variables in the solution y. This parameter must be adjusted experimentally; for example, it can depend on the total number of iterations and/or the number of constraints of the problem. The results presented in Sect. 5 provide an idea of this parameter's influence.

Another way to avoid solving exactly reduced problems is based on the following proposition.

Proposition 5 *Let $\bar{v}(Q)$ be an upper bound on the optimal value of the reduced problem Q to be solved and v^* the value of the best known solution found. Therefore there is no need to solve exactly the reduced problem Q if $\bar{v}(Q) \leq v^*$.*

Proof The proof is based on the classical pruning by bound used in branch-and-bound (see e.g. Nemhauser and Wolsey 1999). □

The upper bound $\bar{v}(Q)$ can be computed by different alternatives using relaxation such LP-relaxation, Lagrangean or surrogate relaxation strategies (see, e.g. Rardin and Karwan 1984). Note that the LP-relaxation affords the most method used for computing the upper bound value strategies (see Rardin and Karwan 1984).

In the next subsection, other ways to accelerate the solving of the reduced problems are described. Constraints are added to these sub-problems only to cut off the search, and a classic reduction rule is used to reduce the size of the initial problem and to accelerate the search.

4.2 Accelerating the solving of the reduced problems

Various techniques can be used to reduce the computational time needed to solve the reduced problems exactly. Our implementation adds four constraints to the reduced problems to reduce the search space to be explored. The following 2 constraints (12) set the lower and upper bounds on the optimal value of the reduced problem. More specifically, these constraints are expressed as:

$$\underline{v} \le cx \le \bar{v} \tag{12}$$

where $\underline{v}$ is the best-known lower bound and $\bar{v}$ is the best-known upper bound. Note that these bounds are improved during the running of the ILPH.

The second two constraints (13) impose bounds on the sum of problem variables. Glover (1965) has already proposed ways to exploit these kinds of constraints, and both Fréville and Plateau (1993) and Vasquez and Hao (2001) have shown that these constraints can improve the efficacy of algorithms for the MKP. More specifically, we add the two constraints:

$$\underline{\sigma} \le \sum_{j \in N} x_j \le \bar{\sigma} \tag{13}$$

where $\underline{\sigma}$ (resp. $\bar{\sigma}$) is a lower (resp. an upper) bound on the sum of the problem variables. These constraints of (13) are valid for the problem P for values of $\underline{\sigma}$ and $\bar{\sigma}$ equal to the optimal values of the following linear programs $\underline{P}$ and $\bar{P}$ (i.e., $\underline{\sigma} = v(\underline{P})$ and $\bar{\sigma} = v(\bar{P})$), respectively:

$$(\underline{P}) \left[\begin{array}{ll} \min & \sum_{j=1}^{n} x_j \\ \text{s.t.:} & Ax \le b, \\ & \underline{v} \le cx \le \bar{v}, \\ & x_j \in [0,1], \quad \forall j \in N, \end{array} \right. \qquad (\bar{P}) \left[\begin{array}{ll} \max & \sum_{j=1}^{n} x_j \\ \text{s.t.:} & Ax \le b, \\ & \underline{v} \le cx \le \bar{v}, \\ & x_j \in [0,1], \quad \forall j \in N. \end{array} \right.$$

The constraints (12 and 13) are adapted to the reduced problems. The bounds $\underline{\sigma}$, $\bar{\sigma}$, $\underline{v}$, and $\bar{v}$ are updated every time a change occurs. For example, improving $\underline{v}$ provokes an update of $\underline{\sigma}$ and $\bar{\sigma}$. Practically speaking, calculating the values $\underline{\sigma}$ and $\bar{\sigma}$ does not require much effort since it involves solving just two linear programs.

Another way to accelerate the ILPH is to reduce the size of the instance. Thus, the following common property is applied to verify whether or not several variables can be set to their optimal values during the algorithm.

Proposition 6 *For any $j \in N$ and for any feasible solution $x^0 \in \{0,1\}^n$, if $v(P(e - x^0, \{j\})) \le cx^0$ (where e is a vector whose components are all set to 1), then either $x_j = x_j^0$ in any optimal solution of the problem, or x^0 is an optimal solution of the problem.*

Although the problem $P(e - x^0, \{j\})$ is as difficult to solve as the original one, the property above remains valid if $v(P(e - x^0, \{j\}))$ is replaced by any of the problem's upper bounds. In our experiments, we used LP-relaxation to apply Proposition 6 quickly. We also tried to set variables every time the lower bound was improved. Applying Proposition 6 requires that only linear programming relaxations be solved, and in many cases doing so helps to reduce the size of the initial problem.

In the next subsection, we propose another version of the ILPH, in which all the constraints generated during the process are not added. The size of the reduced problems is kept under control by using surrogate constraints.

4.3 Controlling the size of the reduced problems

The fact that the ILPH adds one constraint to the current problem at each iteration can increase the number of variables of the reduced problem. In order to prevent the number of constraints from becoming too large, it is worthwhile to use adaptive memory of tabu search (memory based on recency and frequency) in such processes to decide when to drop previously introduced inequalities (see, e.g., Glover and Laguna 1997). Another way consists to replace older constraints by one or several surrogate constraints. For example a simple mechanism that uses surrogate constraints to control the size of the reduced problems is as follows. Let p be a parameter corresponding to the maximum number of variables in the reduced problems. Constraints can be added to the current problem as long as the total number of constraints is less than p. When the number of constraints becomes greater than p, the oldest constraints are replaced by surrogate constraints (Glover 1975). Typically, at iteration $k > p$, the problem P^k contains the following additional constraints:

$$f^l x \leq |J^1(\bar{x}^l)| - 1 \quad \text{for } l = k-p+2, \ldots, k, \tag{14}$$

$$\sum_{l=1}^{k-p+1} \mu_l f^l x \leq \sum_{l=1}^{k-p+1} \mu_l |J^1(\bar{x}^l)| - (k-p+1), \tag{15}$$

where f^l is defined as follows:

$$f_j^l = \begin{cases} 2\bar{x}_j^l - 1 & \text{if } \bar{x}_j^l \in \{0,1\} \\ 0 & \text{if } \bar{x}_j^l \in]0,1[\end{cases}.$$

The constraints (14) are generated between iteration $k - p + 2$ and iteration k, and the surrogate constraint (15) is created as a non-negative linear combination of the constraints generated between iteration 1 and iteration $k-p+1$ (with multiplier μ). When $k \leq p$, all the constraints generated are added to the problem. This version of the ILPH helps to decrease the total computational time of the algorithm, but it doesn't guarantee that the same lower bounds will be obtained. As shown in the following section, this version of the algorithm provides a good compromise between the computational effort expended and the quality of the lower bounds.

4.4 Improving upper bound

In order to reduce the gap between the upper bound and the lower bound it is possible to use the mixed integer programming (MIP) relaxation rather than the linear programming (LP) relaxation. A MIP relaxation of a problem P relative to a subset J of N is defined as follows:

$$\text{MIP}(P, J) \quad \max\{cx : Ax \leq b, x \in [0,1]^n, x_j \in \{0,1\}, \forall j \in J\}. \tag{16}$$

In this relaxation, a subset of variables is forced to be binary for the current problem P, which is modified after adding pseudo-cuts. In practice, the size of the subset is kept small compared to n, and the remaining variables are continuous. We describe in Wilbaut and Hanafi (2009) several ways to integrate this relaxation into the ILPH. For instance the mixed integer programming relaxation could simply replace the linear programming relaxation. The MIP-relaxation can also be used as a technique to diversify the search by forcing the algorithm to explore disjoint reduced problems during the search. It is also possible to use the two relaxations conjointly in other algorithms. More details on the integration of LP and MIP relaxations into the ILPH are given in Wilbaut and Hanafi (2009).

4.5 Intensification and diversification phases

Intensification and diversification methods are the main strategies of tabu search (TS) which significantly improve the efficiency of TS algorithms when solving difficult problems. In this section we propose some ways to incorporate intensification and diversification methods in the ILPH.

The use of adaptive memory in optimization algorithms often increases the quality of the solutions generated. In this paper we use intensification and diversification methods based on the frequency memory of the search. We use two kinds of frequency memory. The first one consists of keeping the number of times a variable was free in the reduced problems. We denote this memory by RedFreq. The second one consists in keeping the sum of the solutions generated by the algorithm for all the variables, which is denoted by SumFreq.

An intensification phase is launched when the best solution was improved during the ILPH. It corresponds to the following scheme in 3 steps:

1. Choose α variables to be free in the reduced problem according to the decreasing order of the values in RedFreq (with α parameter fixed at 40 in our experimentations).
2. Set the other variables to their values in the best solution (set a variable at value 1 if the feasibility of the solution is respected).
3. Solve the associated reduced problem.

The diversification phase consists in generating another reduced problem from the two structures of frequency memories described above and in a similar way as the one used for the intensification method. In a first step we choose α variables to be free in the reduced problem according the increasing order of the values in RedFreq. Then the other variables are fixed according to the decreasing values in SumFreq, and the associated reduced problem is solved. A diversification phase is applied when two consecutive intensification phases do not improve the best solution. As we show in the next section, the integration of all the methods described in the previous pages in the ILPH contributes to the efficiency of this heuristic for solving the MKP.

5 Computational results

The different versions of the ILPH proposed in this paper were tested on a wide set of MKP instances available in the OR-Library (Beasley 1990). It is a collection of 270 correlated, and thus difficult, instances generated using the procedure proposed by Fréville and Plateau (1994). The 270 instances were generated by varying combinations of constraints ($m = 5$, 10, 30) and variables ($n = 100$, 250, 500), with 30 instances being generated for each $n - m$ combination. We implemented all the algorithms presented in this paper in the ‘C’ language compiled with “gcc”, option -O2. We used the commercial solver CPLEX of ILOG to solve the reduced problems. The tests were carried out using a 3.4 GHz Pentium IV computer with 4 Gb RAM. In the following paragraphs, all times are expressed in CPU seconds. Our results are compared with those from CPLEX v9 and the best-known values (Vasquez and Vimont 2005).

We first compared the ILPH alone with CPLEX. We limited CPLEX to 2000 seconds of CPU time for the sets of instances with $m = 5$ and 10, and to 5000 seconds for the instances with $m = 30$. Table 2 presents the average results obtained for solving the 270 instances, with the maximum number of iterations (*Max_Iter*) being set to 50 and 100 for the ILPH (every line is an average over 30 instances). For every instance size, the average

Table 2 Average results for ILPH compared to CPLEX

n	m	$Max\_Iter = 50$		$Max\_Iter = 100$	
		%cx	CPU	%cx	CPU
100	5	0.02	1	0	10
	10	0.02	7	0	44
	30	<0.01	390	0	1586
250	5	0.03	2	0.01	31
	10	0.04	10	0.01	148
	30	0.03	1385	0.01	4284
500	5	0.02	2	0.01	65
	10	0.05	12	0.02	275
	30	0.04	2072	0.02	5108

Table 3 Results for the ILPH with the dominance properties

	n	$Max\_Iter = 25$		$Max\_Iter = 50$		$Max\_Iter = 100$	
		%R	%CPU	%R	%CPU	%R	%CPU
Prop.3	100	22.22	11.21	22.89	2.36	22.22	−9.01
	250	19.33	17.70	21.11	10.22	21.16	8.85
	500	19.20	12.75	21.20	8.40	20.58	4.91
	Average	20.25	13.89	21.73	6.99	21.32	1.58
Prop.4	100	88.08	7.18	71.63	−21.29	61.12	−4.83
	250	76.03	−1.04	50.33	14.01	42.77	22.17
	500	61.41	27.58	36.08	18.26	31.06	19.76
	Average	75.17	11.24	52.68	3.66	44.98	12.37

gap between the value of the best lower bound found by CPLEX and the one obtained by the ILPH is given in the column "%cx" (i.e., ((value_CPLEX - value_ILPH) / value_CPLEX * 100)). The column "CPU" reports the average CPU times in seconds for the ILPH.

As shown in Table 2, the quality of the solutions is better when the number of iterations increases. However, the computational effort becomes significantly larger when $m = 30$. For instances with a number of variables equal to 100, the ILPH finds almost all the optimal solutions within 100 iterations; for the other problems, the quality of the final solution found is comparable with those from CPLEX. These results confirm that the ILPH is efficient, particularly when m is small. The results obtained with our new algorithms are presented and discussed below.

Table 3 gives the results obtained when the dominance properties (Propositions 3 and 4) are introduced. We report in column % R (resp. %CPU) the average percentage of gain in the number of reduced problems solved exactly (resp. in run time) for three distinct values of *Max_Iter*: 25, 50 and 100 and for each value of n. Note that the values of the final lower bounds are the same as those presented in Table 2 since the dominance property only affects the number of reduced problems exactly solved. The results reported for Proposition 4 were obtained with the value of parameter $t = 45, 45, 50$ for $Max\_Iter = 25, 50, 100$ respectively.

Table 4 Results with the control of the size of the reduced problems

p	n	100			250			500		
	m	5	10	30	5	10	30	5	10	30
1	%	0.378	0.312	0.1	0.249	0.222	0.094	0.147	0.132	0.09
	CPU	0.1	0.7	57.5	0.1	0.8	134	0.1	0.8	257.6
5	%	0.144	0.17	0.047	0.161	0.159	0.077	0.087	0.107	0.069
	CPU	0.3	1.4	96.6	0.4	1.6	242.3	0.4	1.8	454.6
10	%	0.044	0.102	0.011	0.078	0.093	0.04 8	0.052	0.078	0.057
	CPU	0.7	3.5	218.6	0.8	3.7	709.5	0.9	4.1	1387.6
20	%	0.007	0.02	0.001	0.021	0.039	0.026	0.016	0.044	0.036
	CPU	3	11.7	659	3.5	15.5	2089	3.5	16	4352

Table 3 shows that the gain in the number of reduced problems solved exactly is significant, and that the gain in run time varies sharply. It appears that adding the dominance properties can increase the execution time of the ILPH, especially for problems with few constraints. The inefficiency of the dominance properties with such problems can be explained by the fact that the dominated reduced problems are generally small and can be solved very quickly. Therefore, the contribution of the dominance property cannot compensate for the cost of detecting the dominated elements in the first place. Table 3 also shows that the gain in the number of reduced problems solved exactly can be more important when we apply Proposition 4. However, this increase depends on the value of the parameter t. In fact, for the same value of t, the values of %R and %CPU generally decrease for two consecutive values of *Max_Iter*. The gain becomes more important for $Max\_Iter \geq 50$ and $t = 50$. When $Max\_Iter = 100$, the total computational time of our algorithm is reduced by about 12%, which is a significant improvement with respect to the initial CPU times of the ILPH, in particular when $m = 30$ (i.e., about 3600 seconds; see Table 2).

To summarize the above results we can say that the added dominance properties improve the performance of the ILPH considerably. The efficiency of the first property is generally higher when the number of iterations is small. If the number of iterations is increased, then the gain in run time decreases, though the gain in problems solved remains on the same order. The situation is different when the second property is integrated. The influence of parameter t on the results is clear. It appears to be difficult to identify a value for t that will insure the best results for the MKP. Here, the value of t increases with the value of *Max_Iter*. However, the efficiency of the algorithm may decrease with increasing values of t.

In Table 4 we present the results obtained when the size of the reduced problems is controlled as described in Sect. 4.3. Like in Table 2, the values of our lower bounds are compared with those obtained with CPLEX. Each column is the average for 30 instances. For each class of instances, the results are given for the quality of the lower bound (row "%") and the amount of run time (row "CPU"). To measure the impact of aggregating the constraints, the algorithm was tested with several possible values for the number of constraints p added to the problem: 1, 5, 10 and 20. In order to obtain a real idea of the efficiency of this version of the algorithm, *Max_Iter* was set to 100.

As Table 4 shows, for a given number of constraints added, the run time increases with n and m and so does the solution quality. In particular, the increase in run time between $m = 10$ and 30 is very high, irrespective of the value of the other parameters. Comparing the values for the quality of the lower bounds and the running times in Tables 2 and 4 reveals a strong decrease in run times and a less abrupt difference between the running times for each value of n. This approach appears to provide solutions that are close in quality to those generated by the ILPH while requiring relatively less computational effort.

The last part of this section provides the results obtained by our algorithm with the first dominance property (Proposition 3) and all the accelerating components (adding constraints, reduction rule and adaptive memory) for the 90 largest instances with $n = 500$ (Table 5). The algorithm was run through 120, 120 and 60 iterations for the instances with $m = 5$, 10 and 30, respectively. For every instance, the best-known lower bound reported by Vasquez and Vimont (2005) is given in column v^*. The absolute and the relative differences between this lower bound and our lower bound is given in columns $v^* - \underline{v}$ and $\%v^*$; the difference between our final upper bound and our final lower bound appears in column $\bar{v} - \underline{v}$, and the time needed to obtain our best lower bound in seconds appears in column CPU^*. Finally column $v^* - \underline{v}^+$ gives the absolute difference between the best-known lower bound and our lower bound when we apply the ILPH during 3600 (resp. 7200) seconds for $m = 5$, 10 (resp. 30).

According to the data reported in Table 5, our algorithm obtained 23, 6 and 2 best-known values for $m = 5$, 10 and 30, respectively. The ILPH also visits one best solution than the one reported in column "v^*" for the instance number 12 with 10 constraints. This improvement was obtained when integrating the use of adaptive memory in the ILPH. To give an idea of the contribution of this integration, note that the ILPH obtains 5 better solutions for $m = 5$ when using it. On the contrary, the use of this integration clearly increases the average CPU time of the ILPH. However for the instances with $m = 5$ this increase is not excessive since it is on average about 57 seconds. The average value of CPU^* is also increased by about 27 seconds (when the final solution is the same with and without the use of adaptive memory). The difference between our lower bounds and the best-known ones is, on average, about 0.01%. The difference between our upper and lower bounds is not very large when $m = 5$, but this difference increases greatly for $m = 10$ and especially $m = 30$. Obviously, using the ILPH as an exact method is not recommended for these instances. The execution time of our algorithm is, on average, about 190, 730 and 3600 seconds for $m = 5$, 10 and 30, respectively. Such a result confirms that our algorithm generates good lower bounds for the MKP within a reasonable time frame.

Table 5 also shows that the ILPH obtains better values for several instances when we fixed the total CPU time (see column $v^* - \underline{v}^+$). Our algorithm then obtained 27, 17 and 8 best-known values for $m = 5$, 10 and 30, respectively. The ILPH also visits one new best solution when $m = 5$ and one more solution when $m = 10$. Finally note that the execution time observed remains reasonable, even when compared to the execution times reported by Vasquez and Vimont (2005) who reported an average execution time of several hours for each of these instances, using a 2 GHz Pentium IV (the RAM was not specified). In addition, using the constraints (12) and (13) helps decrease the computational effort (between 5% and 10%) for the largest instances (with $m = 30$).

6 Conclusions

In this paper, we proposed several enhanced versions of an exact algorithm proposed at the end of the 1970s. This algorithm was designed to generate a non-increasing sequence of upper bounds and a sequence of lower bounds by solving a series of linear programming relaxations and a series of reduced problems. First, we proposed a new proof of the finite

Table 5 Final results for the 90 largest instances ($n = 500$)

Problem	$m = 5$						$m = 10$						$m = 30$					
	v^*	$v^* - \underline{v}$	$\%v^*$	$\bar{v} - \underline{v}$	CPU^*	$v^* - \underline{v}^+$	v^*	$v^* - \underline{v}$	$\%v^*$	$\bar{v} - \underline{v}$	CPU^*	$v^* - \underline{v}^+$	v^*	$v^* - \underline{v}$	$\%v^*$	$\bar{v} - \underline{v}$	CPU^*	$v^* - \underline{v}^+$
0	120148	19	0.02	83	119	0	117811	2	<0.01	186	449	2	116056	109	0.09	651	4366	80
1	117879	0	0	69	54	0	119232	15	0.01	197	119	15	114810	88	0.08	624	1026	30
2	121131	0	0	59	73	0	119215	4	<0.01	173	569	0	116712	51	0.04	662	123	8
3	120804	5	<0.01	71	33	5	118813	0	0	218	105	0	115329	86	0.07	682	3427	19
4	122319	0	0	77	4	0	116509	0	0	170	70	−5	116525	51	0.04	593	2183	0
5	122024	0	0	86	147	0	119504	35	0.03	214	64	0	115741	0	0	612	2277	0
6	119127	0	0	68	19	0	119827	39	0.03	216	378	17	114181	72	0.06	566	1659	72
7	120568	0	0	55	17	0	118329	6	0.01	207	373	0	114348	132	0.1	613	3303	49
8	121575	0	0	67	11	−11	117815	34	0.03	200	65	34	115419	0	0	467	1119	0
9	120717	0	0	61	146	0	119231	34	0.03	251	318	9	117116	93	0.08	624	2769	0
10	218428	2	<0.01	53	253	0	217377	0	0	154	784	0	218104	36	0.02	514	2795	31
11	221202	0	0	58	1	0	219077	11	0.01	167	693	0	214648	98	0.05	508	1359	3
12	217542	8	<0.01	63	47	6	217806	−41	−0.02	168	330	−41	215978	75	0.03	482	1810	33
13	223560	0	0	69	10	0	216868	17	0.01	172	196	0	217910	83	0.04	506	13	35
14	218966	4	<0.01	74	1	0	213859	13	0.01	149	1114	13	215689	76	0.04	465	1047	54
15	220530	3	<0.01	67	66	0	215086	0	0	153	238	0	215890	53	0.02	478	1232	0
16	219989	0	0	69	7	0	217940	14	0.01	159	529	9	215907	107	0.05	549	846	24
17	218215	0	0	68	81	0	219984	0	0	167	96	0	216542	64	0.03	514	3772	32
18	216976	0	0	62	1	0	214375	24	0.01	185	203	0	217340	27	0.01	503	392	7
19	219719	0	0	78	27	0	220899	45	0.02	206	62	13	214739	59	0.03	516	3264	0
20	295828	0	0	45	1	0	304387	24	0.01	164	161	24	301675	48	0.02	399	121	0
21	308086	0	0	51	66	0	302379	8	<0.01	193	60	0	300055	41	0.01	423	1919	0
22	299796	0	0	54	9	0	302416	0	0	143	201	0	305087	7	<0.01	399	1731	7
23	306480	4	<0.01	62	3	0	300757	10	<0.01	179	160	0	302032	27	0.01	433	4302	24
24	300342	0	0	48	1	0	304374	17	0.01	194	546	0	304462	49	0.02	467	724	46
25	302571	0	0	70	54	0	301836	40	0.01	136	315	40	297012	53	0.02	433	1384	47
26	301339	0	0	51	3	0	304952	3	<0.01	167	7	0	303364	60	0.02	444	2162	29
27	306454	0	0	41	14	0	296478	22	0.01	157	22	0	307007	63	0.02	443	280	17
28	302828	0	0	47	42	0	301359	28	0.01	187	42	2	303199	56	0.02	443	2585	21
29	299910	0	0	47	78	0	307089	11	<0.01	154	72	0	300572	36	0.01	465	3865	36

convergence of this algorithm. Then, we proposed a new two-phase heuristic algorithm that uses dominance properties to decrease the number of reduced problems to be solved exactly. This heuristic first solves a series of LP-relaxations and then solves the undominated reduced problems exactly. We also proposed a second heuristic designed to control the size of the reduced problems. We finally introduced adaptive memory to enhance the process. Both heuristics were tested on the 0-1 multidimensional knapsack problem. The results obtained on a set of available and difficult instances show the efficiency of our methods.

This research remains valid for the general 0-1 mixed integer program, and it will be interesting to apply it to particular MIP problems. We also would like to implement new hybrid heuristics that integrate the adaptive memory processes of tabu search to explore the neighbourhoods induced by the relaxations. The idea would be to guide the search by integrating more flexible memories than the ones used in the heuristics presented in this paper.

Acknowledgements The present research work has been supported by International Campus on Safety and Intermodality in Transportation the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the Ministry of Higher Education and Research, and the National Center for Scientific Research. The authors gratefully acknowledge the support of these institutions. We also would like to thank the referees for their valuable suggestions in improving this paper.

References

Ahuja, R. K., Ergun, O., Orlin, J. B., & Punnen, A. P. (2002). Survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, *123*, 75–102.

Balas, E., & Jeroslow, R. (1972). Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, *23*, 61–69.

Beasley, J. E. (1990). OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, *41*, 1069–1072.

Danna, E., Rothberg, E., & Le Pape, C. (2005). Exploring relaxations induced neighborhoods to improve MIP solutions. *Mathematical Programming*, *102*, 71–90.

Fischetti, M., & Lodi, A. (2003). Local Branching. *Mathematical Programming*, *98*, 23–47.

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

Fréville, A., & Hanafi, S. (2005). The multidimensional 0-1 knapsack problem – Bounds and computational aspects. *Annals of Operations Research*, *139*, 195–227. M. Guignard and K. Spielberg (Eds.).

Fréville, A., & Plateau, G. (1993). Sac-à-dos multidimensionnel en variables 0-1: encadrement de la somme des variables à l'optimum. *RAIRO Operations Research*, *27*, 169–187.

Fréville, A., & Plateau, G. (1994). An efficient preprocessing procedure for the multidimensional knapsack problem. *Discrete Applied Mathematics*, *49*, 189–212.

Glover, F. (1965). A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research*, *13*, 879–919.

Glover, F. (1975). Surrogate constraints duality in mathematical programming. *Operations Research*, *23*, 434–451.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, *8*, 156–166.

Glover, F. (2005). Adaptive memory projection methods for integer programming. In C. Rego & B. Alidaee (Eds.), *Metaheuristic optimization via memory and evolution* (pp. 425–440). Boston: Kluwer.

Glover, F., & Kochenberger, G. (1996). Critical event tabu search for multidimensional knapsack problems. In I. H. Osman & J. P. Kelly (Eds.), *Meta heuristics: theory and applications* (pp. 407–427). Boston: Kluwer.

Glover, F., & Laguna, M. (1997). *Tabu search*, (pp. 1–412). Boston: Kluwer.

Guignard, M., & Spielberg, K. (2003). Double contraction, double probing, short starts and BB-probing cuts for mixed (0,1) programming. Technical Report, Wharton School.

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

Hansen, P., & Mladenovic, N. (2001). Variable neighborhood search: principles and applications. *European Journal of Operational Research*, *130*, 449–467.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems* (pp. 1–572). Berlin: Springer.

Lichtenberger, D. (2005). An extended local branching framework and its application to the multidimensional knapsack problem. Master Thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms.

Martello, S., & Toth, P. (1990). *Knapsack problems: algorithms and computer implementations* (pp. 1–308). New York: Wiley.

Nemhauser, G. L., & Wolsey, L. A. (1999). *Integer and combinatorial optimization* (pp. 1–784). New York: Willey.

Pisinger, D. (1995). An expanding-core algorithm for the exact 0-1 knapsack problem. *European Journal of Operational Research*, *87*, 175–187.

Puchinger, J., Raidl, G., & Pferschy, U. (2006). The core concept for the multidimensional knapsack problem. In *Proceedings of evolutionary computation in combinatorial optimization, 6th European conference* (pp. 195–208). EvoCOP 2006.

Rardin, R., & Karwan, M. H. (1984). Surrogate dual multiplier search procedures in integer programming. *Operations Research*, *32*, 52–69.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *The 4th international conference on principles and practice of constraint programming*. Lecture Notes in Computer Science, vol. 1520 (pp. 417–431).

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

Spielberg, K., & Guignard, M. (2000). A sequential (quasi) hot start method for BB (0,1) mixed integer programming. Mathematical Programming Symposium, Atlanta.

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

Vasquez, M., & Vimont, Y. (2005). Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, *165*, 70–81.

Wilbaut, C., & Hanafi, S. (2009). New convergent heuristics for 0-1 mixed integer programming. *European Journal of Operational Research*, *195*, 62–74.

Wilbaut, C., Hanafi, S., Fréville, A., & Balev, S. (2006). Tabu search: global intensification using dynamic programming. *Control and Cybernetic*, *35*, 579–598.