An iterative variable-based fixation heuristic for the 0-1 multidimensional knapsack problem

Christophe Wilbaut, Saïd Salhi, Saïd Hanafi

European Journal of Operational Research 199 (2009) 339–348

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

An iterative variable-based fixation heuristic for the 0-1 multidimensional knapsack problem

Christophe Wilbaut [a,*], Saïd Salhi [b], Saïd Hanafi [a]

[a] LAMIH-SIADE, UMR CNRS 8530, Université de Valenciennes, Le Mont Houy, 59530 Valenciennes Cedex 9, France
[b] Centre for Heuristic Optimisation, Kent Business School, University of Kent, UK

ARTICLE INFO

Article history:
Received 4 October 2007
Accepted 18 November 2008
Available online 6 December 2008

Keywords:
Heuristics
Exact methods
Fixation techniques
Knapsack

ABSTRACT

An iterative scheme which is based on a dynamic fixation of the variables is developed to solve the 0-1 multidimensional knapsack problem. Such a scheme has the advantage of generating memory information, which is used on the one hand to choose the variables to fix either permanently or temporarily and on the other hand to construct feasible solutions of the problem. Adaptations of this mechanism are proposed to explore different parts of the search space and to enhance the behaviour of the algorithm. Encouraging results are presented when tested on the correlated instances of the 0-1 multidimensional knapsack problem.

1. Introduction

The 0-1 multidimensional knapsack problem (**MKP**) is a generalization of the well-known knapsack problem with the presence of more than one constraint. The MKP can be formulated as follows:

$$(\text{MKP})\begin{cases}\max & \sum_{j\in N} c_j x_j,\\ \text{subject to}: & \sum_{j\in N} a_{ij}x_j \leqslant b_i, \quad \forall i\in M=\{1,\dots,m\},\\ & x_j\in\{0,1\}, \quad j\in N=\{1,\dots,n\},\end{cases}$$

where $c_j, \forall j\in N$, $a_{ij}, \forall i\in M$ and $j\in N$ and $b_i, \forall i\in M$ are positive integers. Without loss of generality, we can assume that the following constraints, as defined by (1), are satisfied. If this is not the case, one or more variables could be fixed to 0 or 1

$$\max\{a_{ij}: j\in N\}\leqslant b_i<\sum_{j\in N} a_{ij} \quad \forall i\in M. \tag{1}$$

For simplicity we also use the following matrix notation for the MKP:

$$(\text{MKP}) \quad \max\{c^T x: Ax\leqslant b, x\in\{0,1\}^n\}.$$

Many applications of this NP-hard problem are resource allocation-based, see for instance the first references of Lorie and Savage (1955) and Weingartner (1966). Recently, Meier et al. (2001) used the MKP as a subproblem in a new capital budgeting model. There are other applications such as cutting stock (Gilmore and Gomory, 1966), loading problems (Shih, 1979), and the daily management of a satellite (Vasquez and Hao, 2001a). Efficient algorithms have been proposed for solving the MKP. Several of those are metaheuristic-based algorithms like tabu search (see for instance Glover and Kochenberger, 1996 and Hanafi and Fréville, 1998), and genetic algorithm (Chu and Beasley, 1998). The review article by Fréville and Hanafi (2005) and the book by Kellerer et al. (2004) are informative and provide interesting and useful references. Very recently Wilbaut et al. (2008) produced a survey paper in this area with an emphasis to effective heuristics and their applications.

Several preprocessing techniques are often used to develop efficient integer programming-based approaches. It is well-known that if we are able to reduce the size of the problem to a reasonable level, even NP-hard problems can then be solved optimally with reasonable computational effort. Such a reduction process can be achieved by setting variables, identifying infeasibility and constraint redundancy, and tightening the linear programming (**LP**) relaxation. The latter includes modifying coefficients and generating strong valid inequalities. Some of these tools are described in Nemhauser and Wolsey (1999) and also in Savelsberg (1994). For instance, for the MKP techniques, the idea to reduce the number of variables was exploited by Babayev and Mardanov (1994) and also Zhu and Broughan (1997). Though the above techniques help to reduce the size of the problem in several cases, there is however no guarantee of their efficiency when tested on a given instance. For example, Wilbaut et al. (2006) applied a classical technique to fix variables in a global intensification algorithm, including a dynamic programming method for the MKP, and showed empirically

* Corresponding author. Tel.: +33 327511945.
E-mail addresses: christophe.wilbaut@univ-valenciennes.fr (C. Wilbaut), s.salhi@kent.ac.uk (S. Salhi), said.hanafi@univ-valenciennes.fr (S. Hanafi).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2008.11.036

that it is difficult to fix any variable for those instances with large values of m.

In this paper, we propose an algorithm which is based on a heuristic fixation of the variables for solving the MKP. This is achieved by using an iterative scheme to generate useful information from LP-relaxation. This knowledge is then used to fix iteratively a subset of the decision variables and hence to generate feasible solutions. This method has the advantage of generating both lower and upper bounds of the problem. To explore the diversity of the solutions efficiently, we put forward three variants of the algorithm.

The remainder of this paper is organised as follows: in Section 2 we present the iterative scheme which we use in our algorithm to generate memory information. We describe in Section 3 the different strategies we implemented to fix the variables and to generate bounds of the problem. Section 4 is devoted to the computational results. We summarize our conclusions and point out some research avenues in the last section.

2. An iterative scheme

Several exact methods designed to find the optimal value of the problem were successfully applied to small sized instances. Such a success is unfortunately not repeated for problems with moderate and large size due to memory and computational time requirements. It is well-known that heuristics that use relaxation-based techniques are among the efficient ways to provide both upper and lower bounds for large and difficult combinatorial optimization problems. Glover (2005) proposed a general iterative method for pure and mixed integer programming. This method, which is referred to as the Adaptive Memory Projection (**AMP**), consists of four steps: (i) from an initial solution apply a heuristic to define a subset of the free variables; (ii) use an exact method to solve the sub problem associated with these remaining variables; (iii) re-launch the heuristic used in (i) from the solution obtained in (ii) with the introduction of restrictions generated from the memory; and finally (iv) introduce diversification processes to visit unexplored regions of the search space.

Some of the ideas of this general method do also exist in other efficient algorithms that are based on the exploration of small neighborhoods around the incumbent solution. For instance in Fischetti and Lodi (2003), a constraint, called local branching constraint, is added at every iteration to define a k-OPT neighborhood of the incumbent solution. The motivation is to explore better solutions quickly during the search. In the relaxation induced neighborhood search of Danna et al. (2005), variables which happen to have the same values in both the incumbent solution and in the solution of the current linear programming relaxation are made fixed, and the corresponding remaining sub problem is then optimally solved. For the MKP, Volgenant and Zwiers (2007) recently used partial enumeration. This approach can be viewed as a particular case of the AMP in which steps (i) and (ii) are only applied. Even if this method generates lower bounds of the MKP quickly (i.e. a few seconds/minutes by instance in general), it is clearly outperformed by other methods.

The method described in this paper is based on the scheme proposed by Wilbaut and Hanafi (2008) for solving the 0-1 mixed integer programming problem. For completeness this is shown in Fig. 1.

This interesting method though it guarantees an optimal solution, it was observed that its convergence can be really difficult to achieve in practice especially for larger sized instances. In other words, the results obtained for the MKP showed that even if this scheme was the basis in developing efficient algorithms for generating good bounds, the computational effort associated with these techniques can be significantly high. This can be due to either the large number of reduced problems to solve or the high level of difficulty in solving some of the reduced problems.

In this paper we propose a new algorithm that attempts to overcome the above drawbacks by using an iterative-phase to generate useful information in fixing some variables of the problem heuristically. This iterative process is described in Fig. 2.

The addition of a new constraint in the current problem in Step 2 appears to be useful in generating a better solution of the LP-relaxation. The following two propositions explain the construction of the constraint and show how it only cuts off the current

Step 1: Solve one or more relaxation(s) of the current problem (P) and record the corresponding optimal solution(s).

Step 2: Generate and solve one or more reduced problem(s) induced from the previous solution(s) to obtain one or more feasible solution(s) of (P).

Step 3: Update the best lower bound $\underline{v}^*$ of (P) if necessary and the best upper bound $\bar{v}$ of (P).

Step 4: If a stopping criterion is satisfied then return $\underline{v}^*$ and $\bar{v}$, else add one or more constraint(s) generated from the solution(s) of the relaxation(s) to (P) and return to Step 1.

Fig. 1. A general iterative scheme.

Step 1: Solve the LP-relaxation of the current problem (P) and keep an optimal solution $\bar{x}$ of this relaxation. Update the upper bound $\bar{v}$ of (P).

Step 2: Generate a constraint from $\bar{x}$ which eliminates this solution without eliminating any other solution of the initial problem and add this constraint to (P).

Step 3: If a chosen number of iterations is reached then return $\bar{v}$, otherwise go to Step 1.

Fig. 2. The iterative-phase.

solution of the LP-relaxation. For convenience, we only recall these propositions as their proofs can be found in Wilbaut (2006).

Proposition 1. *Let y be a solution of the MKP. Let* $J^1(y) = \{j \in N : y_j = 1\}$ *and* $J^0(y) = \{j \in N : y_j = 0\}$. *Inequality (2) cuts off solution y without cutting off any other solution in* $\{0,1\}^n$.

$$\sum_{j \in J^1(y)} x_j - \sum_{j \in J^0(y)} x_j \leqslant |J^1(y)| - 1. \tag{2}$$

Let us introduce the following notion of a reduced problem. A reduced problem, noted $P(y, J(y))$, is obtained from a solution y in $[0,1]^n$ of an instance P of MKP and the set $J(y) = J^0(y) \cup J^1(y) = \{j \in N, y_j \in \{0,1\}\}$, by fixing all the variables x_j with $j \in J(y)$ to their value in y (y_j).

Proposition 2. *Let P be an instance of MKP,* $\bar{x}$ *an optimal solution of the LP-relaxation LP(P) and y an optimal solution of the reduced problem* $P(\bar{x}, J(\bar{x}))$, *then an optimal solution of P is either the feasible solution y or an optimal solution of P in which the following constraint is added:*

$$f^T x \leqslant |J^1(\bar{x})| - 1, \tag{3}$$

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

$$f_j = \begin{cases} 2\bar{x}_j - 1 & \text{if } \bar{x}_j \in \{0,1\}, \\ 0 & \text{if } \bar{x}_j \in]0,1[\end{cases} \tag{4}$$

In the next section, we present some methods to exploit the iterative-phase to generate feasible solutions and to reduce the problem by fixing variables heuristically.

3. Memory-based strategies for fixing variables

One of the major questions in variable-fixing methods is related to the choice of the variables to fix at a given step as this affects the behaviour of the algorithm. Based on the previous iterative-phase we propose three versions of our algorithm. In the first subsection we only fix variables temporarily to calculate lower bounds of the MKP, whereas in the other two we examine the permanent fixation of the variables as well.

3.1. Algorithm 1: temporary variables fixation

Here, we first apply the iterative-phase to generate memory information. During this phase, we record the number of times every variable has been fixed to 1, 0 or was found free in the LP-solutions. At the end of this iterative-phase, we propose a scheme to fix the variables that have the most important frequency values associated with value 1 and 0. In other words, we aim to use the memory information to generate a feasible solution of the problem. This is achieved by solving exactly the resulting reduced problem. The algorithm consists in re-launching this process until a stopping condition is satisfied. For example in this work we use a maximum number of passes. The algorithm is described in Fig. 3. We use a long term memory to generate a reduced problem by defining two variables LT^0 and LT^1, which are two n-dimensional vectors, to record the number of times a given variable is found at 0 and 1 in the LP-solutions. In the following we also use the shortcut notation LT to refer to the long term memory.

When the iterative-phase is terminated, we construct the reduced problem by applying the following rules to fix the variables:

$$\text{if } LT_j^1 \geqslant \beta^1 \times n\_iter \text{ then } x_j = 1, \tag{5}$$

$$\text{if } LT_j^0 \leqslant \beta^0 \times n\_iter \text{ then } x_j = 0, \tag{6}$$

where β^1 and β^0 are two parameters in $[0,1]$, and n_iter is the total number of iterations achieved in the iterative-phase.

At the beginning of the algorithm we use $\beta^1 = 1$ and $\beta^0 = 0$. Then, the values of the parameters β^1 and β^0 are updated. This adjustment is necessary as the probability that a variable has all the time the same value in the LP-solutions may decrease when n_iter increases. We implement a simple adaptive mechanism that automatically updates the value of β^1 and β^0 according to the size of the problem and the number of times the iterative-phase has been performed. Note that when we cannot fix a sufficient number of variables according to (5) and (6), a simple mechanism that decreases the values of the parameters β^1 and β^0 until a preset lower limit is introduced.

When repeating the algorithm in the iterative-phase, it is possible that the LP-relaxation may become difficult to solve due to the number of constraints added to the problem. To overcome this drawback, we propose a second version in which we fix variables definitively in the problem to reduce its size and also to simplify its LP resolution.

3.2. Algorithm 2: permanent and temporary variables fixation

3.2.1. Description of the method

In this version of the algorithm we use two types of fixation. The first one consists in fixing temporarily variables to construct a reduced problem as in the first approach. However we also apply a permanent fixation on variables in the original problem to reduce its size iteratively and hence to overcome the difficulty in solving the LP-relaxations. The corresponding algorithm is described in Fig. 4.

In Fig. 4, we denote by P the initial problem. Variable $\underline{v}^*$ refers to our best lower bound, and the set F contains the index of the variables currently fixed in the problem (initially $F = \emptyset$). Problem Q is the problem with the free variables (initially $Q = P$). During the iterative-phase we add the constraint, generated according to Proposition 2, to problem Q^k that refers to the problem at iteration k. Formally speaking, at iteration k, we define $Q^{k+1} = (Q^k \mid \{f.x \leqslant |J^1(x^k)| - 1\})$. Parameter n_iter refers to the number of iterations in the iterative-phase. We define the short term memory ST by two n-dimensional vectors which records the number of

Step 0: Initialize the long term memory (LT) and the best lower bound at 0.

Step 1: Apply the iterative-phase during a given number of iterations, and update the long term memory.

Step 2: Generate and solve a reduced problem according to the long term memory. Update the best lower bound if necessary.

Step 3: If a stopping condition is satisfied then return the final upper bound and the best lower bound, otherwise go to Step 1.

Fig. 3. Iterative algorithm with temporary fixation.

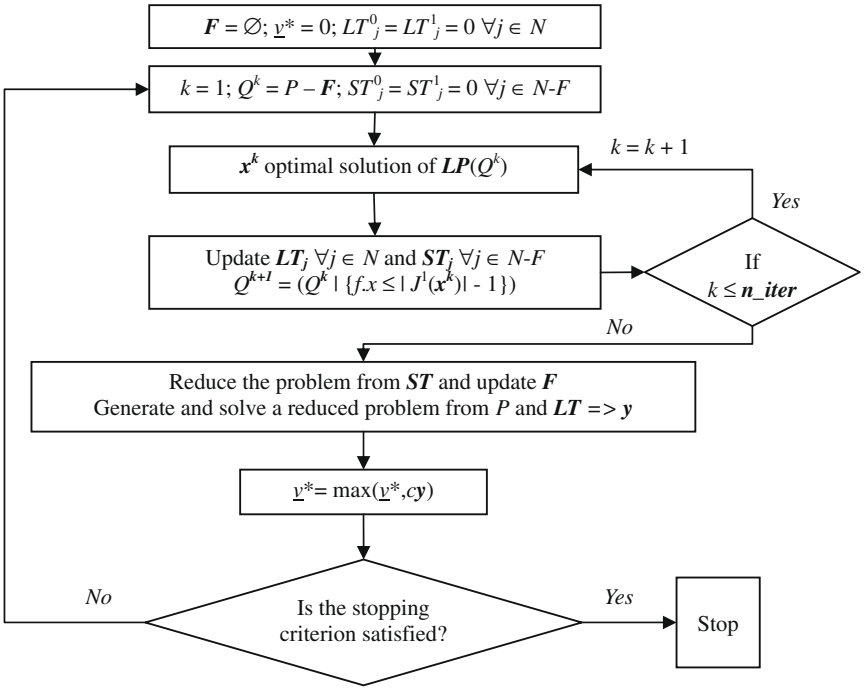

Fig. 4. A schematic description of the iterative algorithm with permanent and temporary fixation.

times a given variable is found at 0 or 1 at the LP stage during one iterative-phase only. Solution y in Fig. 4 is obtained by solving a reduced problem which is constructed from the long term memory, according to the rules described by (5) and (6). If the stopping condition is not satisfied, the algorithm is applied to the problem $P - F$, that consists of the initial problem in which all the variables in F have been fixed. In this version, at the end of every application of the iterative-phase, we apply the rules described in (5) and (6) with ST instead of LT and with $\beta^1 = 1$ and $\beta^0 = 0$, to fix permanently a part of the free variables in the problem. The stopping condition checks whether the remaining problem is sufficiently small to be solved exactly.

We illustrate the progress of this algorithm in Fig. 5 where parameter s represents the number of free variables in the reduced problems. We set the value of s to 30 as we have found, in our earlier preliminary experiments, that this value represents instances for the MKP with a large enough size to be solved exactly and reasonably quickly for the MKP. When generating the reduced problems, the $n - |F| - s$ remaining variables are fixed according to LT as described above. If it is not possible to add a variable due to feasibility, it is simply fixed at value 0. Note in this figure that when applying the iterative-phase for the second time, variables in F are considered neither in the LP-relaxations nor in the reduced problems. We use the size of the final problem to solve exactly

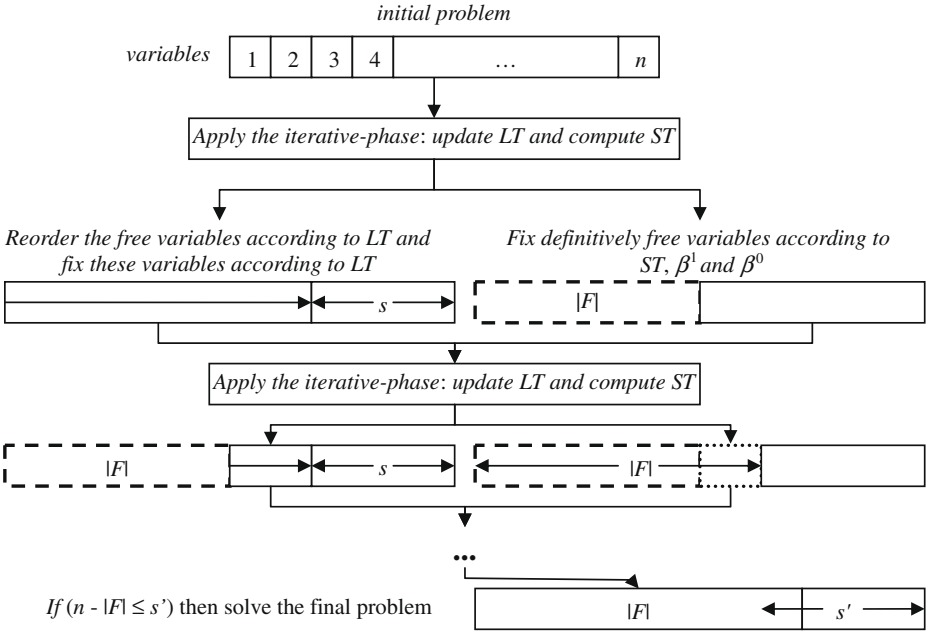

Fig. 5. Construction of the reduced problems and fixation rules.

as $s' \geqslant s$, a parameter to stop the algorithm. In other words, when $n - |F| \leqslant s'$, we solve exactly the problem with the remaining variables and then we stop.

One of the main differences between this algorithm and the one presented in Section 3.1 is that we cannot guarantee that the last upper bound generated in the iterative-phase is a true upper bound of the initial problem because of the permanent fixation of the variables in the problem. Nevertheless, we can record the upper bound associated with the last iteration when applying the iterative-phase for the first time. In addition, we can also generate an upper bound from our final lower bound by solving the following linear program:

$$max\{c^T x : Ax \leqslant b, c^T x > \underline{v}^*, x \in \{0,1\}^n\}.$$

The best upper bound of the two is then chosen.

3.2.2. Preliminary analysis

Parameter *n_iter* in Fig. 4 can clearly influence the behaviour of the algorithm. Indeed if it is too small then it is probable that more variables can be fixed definitively from the information stored in *ST*. On the contrary, if it is too large it would be difficult to fix variables. We evaluated the effect of this parameter on the fixation process on a set of correlated instances of MKP available in the OR-Library (Beasley, 1990). These instances have between 100 and 500 variables and 5 and 30 constraints. Three kinds of correlation have been used according to a parameter denoted by $\alpha = 0.25$, 0.5 and 0.75. Ten instances have been generated for every combination of n, m and α. The set is composed of 270 instances. In this preliminary testing we conducted our experiments on a subset of 27 instances only (i.e. one for every distinct triplet (n, m, α)).

To evaluate the effect of the parameter *n_iter*, we applied the algorithm described in Fig. 4 with several values. We used as the stopping condition the fact that it is impossible to fix any variable after the application of the iterative-phase, or that the remaining problem was sufficiently small to be solved exactly (i.e., when $n - |F| \leqslant s'$). We give in Table 1 the value of the parameter s' we use. Obviously from Table 1, a value of 50 will be large enough to be used in all sizes. Note that when the size of the problem increases the value of s' is then set at 50. In addition when a reduced problem is difficult to solve exactly, it is possible to solve it heuristically for instance by imposing a time limit. We report in Table 2 the average results observed for $n\_iter = 50, 75, 100, 100 - m$. We propose the value $100 - m$ as this takes into account the size of the initial problem (the larger the number of constraints is the more difficult the problem may become). We give in Table 2 the average percentage of variables definitively fixed in the problem when the iterative-phase is terminated for the first time (column "%F1"), and the average percentage of variables definitively fixed in the problem when Algorithm 2 terminates (column "%F"). We also mention the average gap with the best solution ("Avg(Gap)" and the average CPU times (in seconds). We compare our lower bounds with those obtained by Chu and Beasley (1998) or mentioned in Vasquez and Vimont (2005) for the largest instances. Table 2 shows that the value of the parameter *n_iter* has two main effects: the first one is on the fixation process, and the second one is on the execution time. It can be observed that it is more difficult to fix variables when *n_iter* increases. However, the average CPU time is relatively decreased, even though more LP-relaxations are solved. That can be explained by the fact that the total number of times the iterative-phase is launched decreases as *n_iter* increases. It can be noted that the average percentage of variables definitively fixed after the first execution of the iterative-phase is really important for every value of the parameter *n_iter* in Table 2 (see column "%F1"). It was also observed that it is more difficult to fix variables during the remaining passes. Given that the setting $100 - m$ is found to provide reasonably good solutions while consuming less CPU for these instances, this setting will be used in our subsequent testing. However note that in a more general case we can choose the value $n\_iter = 50$ when the number of constraints m increases.

From the above observations, and from the fact that the number of variables we fix at every step can clearly change the behaviour of the algorithm, we propose to introduce a flexible scheme by allowing the fixation of a subset of the potential candidates only.

3.3. Algorithm 3: flexibility in the permanent variables fixation

We propose a way of limiting the number of variables to fix after the iterative-phase. By integrating this scheme within Algorithm 2, we define a kind of "multi-start" version of the algorithm. Our motivation is to avoid fixing too many variables at a given step which has the tendency to restrict the search. Besides, we also introduce diversity into the search by exploring other parts of the search space which were not going to be visited otherwise. This mechanism can also be viewed as a way of limiting the risk associated with the fixation of many variables at their wrong values.

An illustrative example

We report the progress of the algorithm for an instance with $n = 100$ and $m = 5$ when we apply Algorithm 2 with parameter $n\_iter = 95$ (i.e.: $100 - m$). At the end of the first application of the iterative-phase, it is possible to fix 62 variables by applying the rules described in Section 3.2. After the second application of the iterative-phase, five more variables were fixed, and finally four more variables were fixed after the third application of the iterative-phase. The algorithm then terminates by solving exactly the final reduced problem composed by the 29 remaining variables (here $s' = s = 30$). The principle of the third algorithm is to limit the number of variables fixed after the first execution of the iterative-phase, and hence guide the algorithm to explore other directions of the search. For example for this particular instance when we allow only the fixation of 45 (respectively 50) variables among the 62 potential variables, the algorithm stops with 75 (respectively 63) variables fixed. This example shows that this kind of strategy can affect the progress and the results of the method. In the next subsection we explain how we choose to fix at a given step this subset of variables.

3.3.1. Selection of the subset of variables to fix

The choice of the variables we fix among the set of candidates at a given step of the algorithm is based on a bias selection using a pseudo-random algorithm. It introduces the efficiency of the variables according to the classical definition for knapsack problems as

Table 1
Values of s'.

n	m		
	5	10	30
100	35	40	45
250	35	45	50
500	40	45	50

Table 2
Effect of parameter *n_iter* on the fixation method.

n_iter	%F1	%F	Avg(Gap)	Avg(CPU)
50	76.23	78.41	0.06	26
75	72.24	75.67	0.07	22
100	65.58	71.19	0.09	19
$100 - m$	70.99	74.76	0.06	18

follows. The efficiency of variable x_j for $j \in \{1,\ldots,C\}$ is defined by $e_j = \frac{c_j}{\sum_{i\in M}\mu_i a_{ij}}$, where μ is a vector of dimension m, and C is the total number of candidates. In our method, each component μ_i corresponds to the shadow price of the i^{th} constraint in the LP-relaxation of the initial problem. From the sequence $\{e_j\}_{j=\{1,\ldots,C\}}$ we compute the following sequences associated with the subset of candidates to fix at value 1 (their number is denoted by C_1) and the subset of candidates to fix at value 0 (their number is denoted by C_0)

- $\{\delta_j\}_{j=1,\ldots,C_1}$ with $0 \leqslant \delta_j = \frac{e_j}{\sum_{i=1}^{C_1} e_i} \leqslant 1$ (resp. $\{\delta'_j\}_{j=1,\ldots,C_0}$ with $0 \leqslant \delta'_j = \frac{1/e_j}{\sum_{i=1}^{C_0} 1/e_i} \leqslant 1$).
- $\{\varphi_j\}_{j=1,\ldots,C_1}$ (resp. $\{\varphi'_j\}_{j=1,\ldots,C_0}$) defined by

$$\begin{cases} \varphi_1 = \delta_1 \ (\text{resp.}\varphi'_1 = \delta'_1), \\ \varphi_j = \varphi_{j-1} + \delta_j, j = 2,\ldots,C_1 \quad (\text{resp.}\varphi'_j = \varphi'_{j-1} + \delta'_j, j = 2,\ldots,C_0). \end{cases}$$

The following process is repeated until the required number of fixed variables is reached:

- Choose randomly two numbers d and r in $[0,1]$.
- If $d < \frac{C_1}{C_0+C_1}$ then determine the corresponding candidate to fix at 1 by $\varphi^{-1}(r)$; otherwise the corresponding candidate to fix at 0 is given by $\varphi'^{-1}(r)$, with $\varphi^{-1}(r) = max\{j : \varphi_j \leqslant r\}$ and $\varphi'^{-1}(r) = max\{j : \varphi'_j \leqslant r\}$.

An illustrative example

The following example is used to show how this works. Suppose that we have 10 candidates to fix at value 1 at a given step with the following values of δ_j and φ_j.

From this table, suppose that $r = 0.33$. In this condition we choose item 3.

Item	1	2	3	4	5	6	7	8	9	10
δ_j	0.02	0.05	0.2	0.07	0.16	0.19	0.08	0.13	0.04	0.06
φ_j	0.02	0.07	0.27	0.34	0.5	0.69	0.77	0.9	0.94	1

When using this method, two executions of the algorithm may obviously lead to different solutions. Thus we start the algorithm again by changing the variables fixed (i.e. by choosing other candidates). To do this, we apply a backtrack phase to find a step for which we fixed less variables than the number of candidates. In practice, according to the results presented in Table 2, this selection mechanism happens to be applied generally after the first pass of the iterative-phase. This could be because other steps of the algorithm do not generate an important number of candidates to fix. Note that Algorithm 2 is a particular case of this version in which all the candidates are fixed.

3.3.2. The size of the selected subset

In this subsection, we define the size of the subset of the variables to fix, by a parameter $Fixed_1$. This corresponds to the percentage of variables we allow to fix among the set of candidates after the first iterative-phase. We report in Table 3 the results obtained when we execute this algorithm with four different values of parameter $Fixed_1$ over the 27 instances described in Section 3.2.2. The algorithm terminates if the size of the initial problem is less than s' or if we do not improve the best solution during the current execution. We record the average gap with the best solution ("Avg(Gap)"), the number of times a best solution is obtained ("#Best") and the average CPU time (in seconds). There is obviously no optimal value of parameter $Fixed_1$ that guarantees the best results, but in these experiments the value of 90% appears to produce the best results on average, though requiring about twice CPU times than those used for the value of 100%. The latter corresponds to the execution of Algorithm 2.

Table 3
Impact of the value of $Fixed_1$.

$Fixed_1$	Avg(Gap)	#Best	CPU
70	0.16	6	38
80	0.11	8	28
90	0.05	10	35
100	0.06	9	18

As it is not practical to set such a parameter of the algorithm to a particular value for every instance of the MKP, we have introduced an adaptive and robust method to avoid the issue of fine tuning. We define this version as a "multi-start" version that dynamically adjusts the value of the parameter $Fixed_1$ according to the state of the search. The final version of the algorithm, which is described in the next subsection, attempts to take into account these ideas.

3.3.3. An adaptive fixation of the subset size

The algorithm described in this section is based on Algorithm 2 as given in Section 3.2. The term "multi-start" is used because the algorithm manages the value of parameter $Fixed_1$ during the search. From Fig. 4, we incorporate an adaptive scheme to select the variables to fix (Section 3.3.1), which makes up our final algorithm as described in Fig. 6.

In Step 4 the algorithm determines a promising range to explore. This range is obtained by decreasing the value of parameter $Fixed_1$ while the best lower bound is improved. Then the algorithm explores this range in Step 5 with the last two values of $Fixed_1$ using a dichotomous method which is defined below. Note that in this version the stopping condition for every application of Algorithm 2 when $Fixed_1$ is different to 100, corresponds to the fact that the best "global" solution has not been improved, or the size of the remaining problem is less than s'. Step 5 in Fig. 6 ensures the termination of the algorithm. Finally note that a list of several reduced problems induced from solutions of the LP-relaxations of the problem is also kept. This list is used to check whether a reduced problem happens to be already generated during the search.

Dichotomous scheme

The dichotomous method used in Step 5 in Fig. 6 can be summarized as follows. The idea of this commonly used numerical method is to squeeze the range $[I_0, I_1]$ until its length is fairly small. Initially $I_1 = Fixed_1$, $Fixed_1 = 0.8Fixed_1$ (see Step 4 of Fig. 6).

(i) Set $Fixed_1 = (I_0 + I_1)/2$.
(ii) If the best solution was not improved, set $I_1 = Fixed_1$ else set $I_0 = Fixed_1$.
(iii) Repeat (i) and (ii) until $I_1 - I_0 \leqslant \epsilon$. In our experimentations, we use $\epsilon = 3$.

Effect of the pseudo-randomness of the selection

In this last version of the algorithm, we use a random-based algorithm (see Section 3.3.1). We give in Table 4 an illustration of the impact of this algorithm when choosing the variables to fix on the instance 5.250_10. We have executed the algorithm five times for several values of $Fixed_1$ (between 70 and 90), and also five times for Algorithm 3 (adaptive setting of $Fixed_1$). We present in Table 4 the average results observed in terms of the value of the best solution visited ("Avg(cx)"), the number of times that an optimal solution is obtained ("#Best") and the average CPU time. The results presented in Table 4 show the fact that the pseudo-random algorithm clearly involves changes during the process since the average value of the best solution visited changes for every value

Step 0: Initialize the long term memory LT and the best lower bound $\underline{v}^*$=0, and set $Fixed_1 = 100$.

Step 1: Initialize the short term memory ST for every free variable of the current problem.

Step 2: Apply the iterative-phase during $iter$ iterations, and update ST.

Step 3: Determine some variables to fix in the problem according to ST. Generate and solve a reduced problem constructed from LT. Update $\underline{v}^*$ if necessary. If there is no variable to fix then apply a backtrack phase to fix other possible candidates in a previous step. If the backtrack phase succeeds then go to Step 2, otherwise go to Step 4.

Step 4: If $\underline{v}^*$ has been improved in the current pass then set $I_1 = Fixed_1$, $Fixed_1 = 0.8Fixed_1$ and $I_0 = Fixed_1$, and go to Step 1; Otherwise go to Step 5.

Step 5: Use a dichotomous method to explore the range $[I_0; I_1]$ by applying steps 2 and 3 with different values of $Fixed_1$.

Fig. 6. The adaptive fixation algorithm.

Table 4
Illustration of the pseudo-random algorithm to fix the variables.

$Fixed_1$	Avg(cx)	#Best	Avg(CPU)
70	109074.2	0	5.4
80	109026.6	0	4.6
90	109087.8	1	7.2
Algorithm 3	109109	5	59.2

of $Fixed_1$. However, we can note that Algorithm 3 obtains an optimal solution for every execution. That means that, based on these experiments, the behaviour of this algorithm seems to be independent of the random factors used in practice and it is rather stable. As this was the case for all the instances used during the preliminary experiments, we decided to apply Algorithm 3 only once for each instance instead.

4. Computational results

The algorithms presented in this paper are coded in *C++*. The results have been obtained on a Pentium IV 3.4 GHz. CPLEX9.0 of Ilog is used to solve exactly the reduced problems and the linear programming relaxations during the iterative-phase.

We present in Table 5 the final results obtained over the 270 instances of the OR-Library presented in Section 3.2.2. We report for Algorithms 1–3 the average gap between our lower bounds and the LP-value (rows "Avg(LP)"). We also give the average gap with our final upper bound (rows "Avg(UB)"), and the average CPU times in seconds (rows "CPU"). Note that each value in Table 5 is an average over 10 instances. The last column reports the overall average results for our three variants. Table 5 shows that with regard to the CPU time, Algorithm 3 appears to be 3–4 times slower than the others. This is mainly due to the use of several values of the parameter $Fixed_1$. The contribution of Algorithm 3 may, at the first glance, be considered relatively small in terms of solution quality given its relatively larger CPU time. However, this improvement is found to be remarkably interesting when analysing the results more precisely as will be shown in the next subsection.

4.1. Existence of upper bounds

Note that Algorithm 1 obtains better results from the upper bound point of view. This is due to the fact that we can recover the last upper bound generated by this algorithm as a final upper bound. For all the algorithms we are able to improve the gap between the upper bound and the lower bound from 0.5 to 0.3 approximately for Algorithm 1 and about 0.4 for both Algorithms 2 and 3. This added information about the duality gap is useful in practice in general and in heuristic search in particular.

4.2. Comparison vs other methods

We compare our lower bounds with those obtained with other efficient algorithms in Table 6. To avoid overloading the presentation, we only report the results obtained with Algorithm 3. In Table 6 we give the average values over 10 instances (for each (n, m, α) values) mentioned by Chu and Beasley (1998) who used a genetic algorithm (column "C&B") and those obtained by Osorio et al. (2002) who exploit nested cut inequalities and surrogate constraints (column "O&G&H"). Note that Hanafi and Glover (2007) have shown recently how this method can be improved to yield

Table 5
Average results over the 270 instances of the OR-Library.

	n	100	250	500	100	250	500	100	250	500	Overall average
	m	5			10			30			
Algorithm 1	Avg(LP)	0.59	0.14	0.05	0.95	0.28	0.12	1.70	0.65	0.33	0.54
	Avg(UB)	0.13	0.07	0.03	0.40	0.22	0.10	1.01	0.58	0.32	0.32
	CPU	3	15	46	12	76	115	85	145	162	73
Algorithm 2	Avg(LP)	0.58	0.14	0.05	0.95	0.29	0.11	1.68	0.65	0.33	0.53
	Avg(UB)	0.29	0.10	0.04	0.65	0.24	0.10	1.35	0.61	0.32	0.41
	CPU	3	18	60	17	104	162	107	185	209	96
Algorithm 3	Avg(LP)	0.58	0.14	0.05	0.95	0.28	0.11	1.68	0.65	0.32	0.53
	Avg(UB)	0.29	0.08	0.04	0.65	0.23	0.10	1.31	0.59	0.31	0.4
	CPU	11	30	117	37	216	273	363	774	884	301

Table 6
Comparison with other efficient algorithms.

n	m	α	C&B	O&G&H	V&V	W&H	Algorithm 3	
							Avg	Nopt
100	5	0.25	**24197.2**	**24197**	n/k	n/k	**24197.2**	10
100	5	0.5	**43252.9**	**43253**	n/k	n/k	**43252.9**	10
100	5	0.75	**60471.0**	**60471**	n/k	n/k	**60471.0**	10
100	10	0.25	**22601.9**	**22602**	n/k	n/k	**22601.9**	10
100	10	0.5	42659.1	**42661**	n/k	n/k	42655.5	9
100	10	0.75	**59555.6**	**59556**	n/k	n/k	**59555.6**	10
100	30	0.25	21654.2	21656	n/k	n/k	**21660.4**	10
100	30	0.5	41431.3	41437	n/k	n/k	**41438.3**	8
100	30	0.75	59199.1	**59202**	n/k	n/k	**59201.8**	10
250	5	0.25	60409.7	**60413**	n/k	n/k	60409.9	8
250	5	0.5	109284.6	**109293**	n/k	n/k	109288.9	7
250	5	0.75	151555.9	**151560**	n/k	n/k	**151560.3**	10
250	10	0.25	58993.9	**59019**	n/k	n/k	59015.2	n/o
250	10	0.5	108706.4	108607	n/k	n/k	**108724.4**	n/o
250	10	0.75	151330.4	**151363**	n/k	n/k	151334.4	n/o
250	30	0.25	56875.9	**56959**	n/k	n/k	56894.4	n/o
250	30	0.5	106673.7	**106686**	n/k	n/k	106684.0	n/o
250	30	0.75	150443.5	150467	n/k	n/k	**150470.7**	n/o
500	5	0.25	120615.5	120610	120629.2	**120630.3**	120623.3	5
500	5	0.5	219503.1	219504	**219512.7**	**219512.7**	219508.5	5
500	5	0.75	302354.9	302361	**302363.4**	**302363.4**	302360.7	6
500	10	0.25	118565.5	118584	**118628.6**	118626.2	118610.5	n/o
500	10	0.5	217274.6	217297	217327.1	**217329.9**	217313.0	n/o
500	10	0.75	302556.0	302562	302602.7	**302604.6**	302586.6	n/o
500	30	0.25	115473.5	115520	**115623.7**	115607	115540.4	n/o
500	30	0.5	216156.9	216180	**216274.7**	216258.6	216201.5	n/o
500	30	0.75	302353.4	302373	**302446.5**	302433	302389.5	n/o
		Average time	20 min	3 h	16 h	1 h 15min	5 min	
		Processor	SGI R4000 100 MHz	P3 450 MHz	P4 2 GHz	P4 3.4 GHz	P4 3.4 GHz	
	n/k:	Not known						
	n/o:	Not obtained						

better results. We also report the values for the largest instances ($n = 500$) mentioned in Vasquez and Vimont (2005) (column "V&V") and in Wilbaut and Hanafi (2009) (column "W&H"). Lower bounds reported in these papers surpass many other lower bounds referenced in the literature. However the CPU times for these approaches are found to be rather high. For each line we put in bold font the best average value(s). For each method we also give the average CPU time and the processor used. Finally column "Nopt" reports the number of optimal solutions found by Algorithm 3 for the instances optimally solved by Cplex with a CPU time of 1 h. Table 6 confirms that our approach is competitive for solving this set of 270 instances of the MKP. The compromise between the solution quality and the required CPU time seems to be interesting. To complete the analysis, we give in Table 7 a synthesis of the results over the 90 largest instances when $n = 500$. We report the average deviations from the LP-relaxation upper bound (i.e. (LP value − solution value)/LP value). We compare the results of our three algorithms with those of the previous methods, and also with the results obtained very recently by Fleszar and Hindi (2008) (row "F&H"). They proposed fast heuristics for solving the MKP. Some of these heuristics are based on ideas previously used in Volgenant and Zwiers (2007) or Vasquez and Vimont (2005). Table 7 also reports the average CPU time and the processor used. Table 7 shows that our three algorithms surpass those of Chu and Beasley (1998), Osorio et al. (2002), and Fleszar and Hindi (2008) for the largest instances. As mentioned previously, the results obtained by Vasquez and Vimont (2005) and Wilbaut and Hanafi (2009) are superior for these particular instances.

Table 7
Comparison with other efficient algorithms for $n = 500$.

Algorithm	Average Deviation (%)	Average time	Processor
C&B	0.178	34.67 min	SGI R4000 100 MHz
F&H	0.173	1 min	PM 2 GHz
O&G&H	0.169	3 h	P3 450 MHz
W&H	0.144	1 h 15 min	P4 3.4 GHz
V&V	0.141	16 h	P4 2 GHz
Algorithm 1	0.166	1.8 min	P4 3.4 GHz
Algorithm 2	0.163	2.4 min	
Algorithm 3	0.159	7 min	

4.3. Some experiments with larger instances

We evaluate the "adaptation" of our algorithm for larger instances. We apply our heuristics on a set of 18 instances proposed by Glover and Kochenberger (1996) with $n \in [100, 2500]$ and $m \in [15, 100]$. Table 8 reports the results obtained by our 3 algorithms. We also give the lower bounds reported by Vasquez and Hao (2001b) (column "V&H"). Table 8 shows that the computational effort associated with our algorithms does not increase excessively when the number of variables and/or constraints increases. That is possible with a preliminary adjustment of some of the parameters of the algorithms namely $n\_iter = 50$ in the iterative-phase, $s' = 50$ as a stopping condition, and use of a heuristic way to solve the reduced problems. The results also show that the number of constraints in the instance seems to be decisive from the CPU time point of view. The difference between our lower bounds and those reported in column "V&H" is not really important and this justifies the robustness of our approach. It is not easy to evaluate the exactness of the fixation during the process. Some complementary experiments showed that wrong fixation can happen principally at stages 1 or 2 of the algorithm especially when the number of fixed variables is large. Despite such a possible drawback, the results presented in this section show that Algorithm 3 is generally able to obtain high quality solutions. In particular, Table 6 shows that our algorithm visits 128 optimal solutions

Table 8
Results for larger instances.

Pb	*n*	*m*	V&H	Algorithm 1		Algorithm 2		Algorithm 3	
				Value	CPU	Value	CPU	Value	CPU
GK018	100	25	4528	4526	95	4528	80	4528	121
GK019	100	25	3869	3867	102	3869	59	3869	87
GK020	100	25	5180	5180	52	5180	87	5180	135
GK021	100	25	3200	3200	66	3200	115	3200	170
GK022	100	25	2523	2523	96	2523	92	2523	147
GK023	200	15	9235	9234	27	9234	54	9234	60
GK024	500	25	9070	9068	127	9067	78	9068	106
MK_gk01	100	15	3766	3766	4	3766	4	3766	11
MK_gk02	100	25	3958	3956	32	3957	38	3958	78
MK_gk03	150	25	5656	5651	89	5655	119	5655	235
MK_gk04	150	50	5767	5765	155	5764	257	5765	207
MK_gk05	200	25	7560	7558	140	7559	155	7560	1227
MK_gk06	200	50	7677	7673	199	7670	204	7674	1040
MK_gk07	500	25	19220	19216	244	19216	265	19214	398
MK_gk08	500	50	18806	18799	338	18798	467	18798	1045
MK_gk09	1500	25	58087	58085	350	58086	360	58086	592
MK_gk10	1500	50	57295	57285	556	57284	633	57289	1254
MK_gk11	2500	100	95237	95208	755	95218	3645	95223	5863

among the 150 available. That means that the mechanisms we set up to try to correct the wrong variables fixation are relatively efficient.

5. Conclusions

In this paper we proposed new iterative heuristics with variable fixation to solve the 0-1 multidimensional knapsack problem. The motivation is to reduce the problem until it becomes sufficiently small to be solved with an exact method in a reasonable CPU time. Our algorithms are based on an iterative scheme that uses information from a series of LP-relaxations. This information is used to fix heuristically a subset of variables during the search, some of which are permanently fixed whereas others are just temporarily fixed. Flexibility is also introduced through backtracking to avoid early convergence. We propose a version of the algorithm in which the fixation process is more adaptive to obtain a more robust method. The results obtained over the well-known 270 correlated instances of the 0-1 multidimensional knapsack problems available on the Internet are found to be competitive with existing approaches. One interesting feature of our approaches, besides being adaptive, they provide a reasonable compromise between the solution quality and the CPU times. In addition, our methods appear to rival other heuristics in this field by generating top quality solutions.

We believe that the dynamic fixation scheme with the use of information induced from the search is challenging but a worthwhile research avenue that deserves to be explored for other combinatorial optimization problems. The authors are currently investigating a class of location problem namely the *p*-median, see Salhi and Drezner (2007) for references.

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 anonymous referees for their valuable suggestions and constructive comments that improved this paper.

References

Babayev, D.A., Mardanov, S.S., 1994. Reducing the number of variables in integer and linear programming problems. Computational Optimization and Applications 3, 99–109.
Beasley, J.E., 1990. OR-Library: distributing test problems by electronic mail. Journal of the Operational Research Society 41, 1069–1072.
Chu, P., Beasley, J.E., 1998. A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics 4, 63–86.
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.
Fleszar, K., Hindi, K.S., 2008. Fast, effective heuristics for the 0-1 multi-dimensional knapsack problem. Computers and Operations Research. doi:10.1016/j.cor.2008.03.003.
Fréville, A., Hanafi, S., 2005. The multidimensional 0-1 knapsack problem – bounds and computational aspects. Annals of Operations Research 139, 195–227.
Gilmore, P.C., Gomory, R.E., 1966. The theory and computation of knapsack functions. Operations Research 14, 1045–1075.
Glover, F., 2005. Adaptive memory projection methods for integer programming. In: Rego, C., Alidaee, B. (Eds.), Metaheuristic Optimization Via Memory and Evolution. Kluwer Academic Publishers, pp. 425–440.
Glover, F., Kochenberger, G., 1996. Critical event tabu search for multidimensional knapsack problems. In: Osman, I., Kelly, J. (Eds.), Meta Heuristics: Theory and Applications. Kluwer Academic Publishers, pp. 407–427.
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.
Hanafi, S., Glover, F., 2007. Exploiting nested inequalities and surrogate constraints. European Journal of Operational Research 179, 50–63.
Kellerer, H., Pferschy, U., Pisinger, D., 2004. Knapsack Problems. Springer.
Lorie, J.H., Savage, L.J., 1955. Three problems in capital rationing. The Journal of Business 28, 229–239.
Meier, H., Christofides, N., Salkin, G., 2001. Capital budgeting under uncertaintyan integrated approach using contingent claims analysis and integer programming. Operations Research 49, 196–206.
Nemhauser, G.L., Wolsey, L.A., 1999. Integer and Combinatorial Optimization. Wiley-Interscience, New York.
Osorio, M.A., Glover, F., Hammer, P., 2002. Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. Annals of Operations Research 117, 71–93.
Salhi, S., Drezner, Z., 2007. Location analysis: theory and applications. IMA – Journal of Management Mathematics 17, 305–425.
Savelsberg, M.W.P., 1994. Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal of Computing 6, 445–454.
Shih, W., 1979. A branch and bound method for the multiconstraint zero-one knapsack problem. Journal of the Operational Research Society 30, 369–378.
Vasquez, M., Hao, J.K., 2001a. A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. Computational Optimization and Applications 20, 137–157.
Vasquez, M., Hao, J.K., 2001b. 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.

Volgenant, A., Zwiers, I.Y., 2007. Partial enumeration in heuristics for some combinatorial optimization problems. Journal of the Operational Research Society 58, 73–79.
Weingartner, H.M., 1966. Capital budgeting of interrelated projects: survey and synthesis. Management Science 12, 485–516.
Wilbaut, C., 2006. Heuristiques hybrides pour la résolution de problèmes en nombres entiers mixtes. Ph.D. Thesis, Université de Valenciennes et du Hainaut Cambrésis.
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 (3), 579–598.
Wilbaut, C., Hanafi, S., Salhi, S., 2008. A survey of effective heuristics and their application to a variety of knapsack problems. IMA – Journal of Management Mathematics. doi:10.1093/imaman/dpn004.
Zhu, N., Broughan, K., 1997. A note on reducing the number of variables in integer programming problems. Computational Optimization and Applications 8, 263–272.