New convergent heuristics for 0–1 mixed integer programming

Christophe Wilbaut, Said Hanafi

▶ **To cite this version:**

Christophe Wilbaut, Said Hanafi. New convergent heuristics for 0–1 mixed integer programming. European Journal of Operational Research, 2008, 195, pp.62 - 74. 10.1016/j.ejor.2008.01.044 . hal-03723751

HAL Id: hal-03723751
https://uphf.hal.science/hal-03723751

Submitted on 15 Jul 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Author's personal copy

Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 195 (2009) 62–74

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Discrete Optimization

New convergent heuristics for 0–1 mixed integer programming

Christophe Wilbaut [*], Said Hanafi

LAMIH-ROI, ISTV2, Université de Valenciennes Le Mont Houy, 59313 Valenciennes Cedex 9, France

Received 28 November 2006; accepted 20 January 2008
Available online 7 February 2008

Abstract

Several hybrid methods have recently been proposed for solving 0–1 mixed integer programming problems. Some of these methods are based on the complete exploration of small neighborhoods. In this paper, we present several convergent algorithms that solve a series of small sub-problems generated by exploiting information obtained from a series of relaxations. These algorithms generate a sequence of upper bounds and a sequence of lower bounds around the optimal value. First, the principle of a linear programming-based algorithm is summarized, and several enhancements of this algorithm are presented. Next, new hybrid heuristics that use linear programming and/or mixed integer programming relaxations are proposed. The mixed integer programming (MIP) relaxation diversifies the search process and introduces new constraints in the problem. This MIP relaxation also helps to reduce the gap between the final upper bound and lower bound. Our algorithms improved 14 best-known solutions from a set of 108 available and correlated instances of the 0–1 multidimensional Knapsack problem. Other encouraging results obtained for 0–1 MIP problems are also presented.
© 2008 Elsevier B.V. All rights reserved.

Keywords: Relaxation; 0–1 Mixed integer programming; Heuristic; Multidimensional Knapsack problem

1. Introduction

The 0–1 mixed integer programming (0–1 MIP) problem works to maximize or minimize a function of many variables subject to inequality/equality constraints and binary choice restrictions on some of the variables. The 0–1 MIP problem allows a wide range of practical problems in business, engineering and science to be formulated (see Nemhauser and Wolsey, 1999). Throughout this paper, we deal with a maximization function, and we assume that the constraints are linear. The 0–1 mixed integer programming problem (P) can be expressed as

$$(P)\left[\begin{array}{ll} \max & \sum_{j=1}^{n'} c_j x_j \\ \text{s.t.} & \sum_{j=1}^{n'} a_{ij}x_j \leqslant b_i \ \forall i \in M = \{1,\ldots,m\} \\ & x_j \in \{0,1\} \ j \in N = \{1,\ldots,n\} \\ & x_j \geqslant 0 \ j = n+1,\ldots,n' \end{array}\right.,$$

where N is the set of binary variables and $n = |N|$. We also assume that P is feasible and that all the data c_j, a_{ij} and b_i data are integers. In this paper, we use the following shortcut notation of the problem P:

$$(P) \quad \max\{c^T x : x \in X\}, \tag{1}$$

where $X = \{x \in \mathrm{IR}^{n'} : Ax \leqslant b, x_j \geqslant 0, j = 1,\ldots,n' \text{ and } x_j \in \{0,1\}, j = 1,\ldots,n\}$. The linear programming relaxation of (P) that results from dropping the integer requirement on x is denoted LP(P) (i.e. $\mathrm{LP}(P) = \{\max c^T x : x \in \overline{X}\}$, where $\overline{X} = \{x \in \mathrm{IR}^{n'} : Ax \leqslant b, x_j \geqslant 0, j = 1,\ldots,n' \text{ and } x_j \in [0,1], j = 1,\ldots,n\}$).

Several special cases of the 0–1 MIP problem including Knapsack, set packing or travelling salesman for instance, are known to be NP-hard (Garey and Johnson, 1979), but even within this class of problems it is considered as one of the most challenging. Exact methods designed to find the optimal value have been successfully applied to small problems, but such methods are not able to find a high quality solution within a reasonable amount of CPU resources. Large-scale problems require good approximations of the optimal value, and both heuristic and relaxation methods

[*] Corresponding author. Tel.: +33 0 327511957.
E-mail addresses: christophe.wilbaut@univ-valenciennes.fr (C. Wilbaut), said.hanafi@univ-valenciennes.fr (S. Hanafi).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2008.01.044

Author's personal copy

have proved useful for providing good upper and lower bounds of the optimal value in large and difficult optimization problems.

In this paper, we propose several exact algorithms for solving 0–1 mixed integer programming problems; these algorithms incorporate both relaxation and heuristic methods. We summarize a general version of convergent heuristic that combines an exact method for small problems and relaxations generated by applying the basic ideas outlined in the following sections. Our solution approach then proceeds as follows:

1. At each iteration solve one or more relaxations of the current problem P to generate one or more constraints.
2. Solve one or more reduced problems induced by the optimal solution(s) of the previous relaxation(s) to obtain one or more feasible solution(s) of the initial problem.
3. If the stopping criterion is satisfied then return the best lower bound and the best upper bound. Otherwise, add the constraint(s) generated in *step 1* to the problem P and return to *step 1*.

Observe that the implementation of the algorithms depends on the relaxations used, the number of constraints added in the problem, the number of reduced problems to solve and the stopping criterion. We stress that the preceding description of the convergent heuristic is simply a taxonomic device for grouping methods that share certain features of several recent hybrid algorithms for solving optimization problems. Shaw (1998) proposed a Large Neighborhood Search method, combining the neighborhood exploration used in the classical local search strategies with the constraint programming methodology. He applied it to vehicle routing problems. Ahuja et al. (2002) conducted a survey of very large-scale neighborhood search techniques for solving optimization problems. Since two important issues in local search methods are how large the neighborhood should be and how this neighborhood should be explored, they decomposed the techniques into three important classes. In the first one, exponentially large neighborhoods are considered and explored heuristically. In the second one, large neighborhoods are explored using network techniques. In the third class, the neighborhoods are defined using restrictions of the original problem, such that they are solvable in polynomial time. Fischetti and Lodi (2003) developed another approach, called local branching (**LB**), in which a small-size neighborhood is explored at each node of a branch-and-bound (or a branch-and-cut) search tree. The neighborhood is generated around the incumbent feasible solution and a cut called the "local branching cut" is added in the problem. In the same context, Danna et al. (2005) proposed a relaxation induced neighborhood search (**RINS**) for solving the MIP. RINS generates a promising neighborhood from the incumbent solution and from the continuous relaxation at some current nodes of the branch-and-cut search tree, and then solves a sub-problem exactly. Hansen et al. (2006) have combined the LB approach with the variable neighborhood search (**VNS**) one to solve the MIP. The VNS metaheuristic systematically changes the neighborhood of the solution, both by descending toward local optima and by escaping from the subspace that contains these optima. In this paper the neighborhoods are defined from the incumbent and the sub problems are exactly solved as in LB. Wilbaut et al. (2006) have proposed a global intensification scheme that combines Tabu Search (**TS**) with Dynamic Programming (**DP**). The forward phase of DP is used to construct a list L by solving a family of small sub problems exactly. TS uses the list L to evaluate the neighborhood of the current solution in the backtracking phase of DP. Recently, Glover (2005) proposed an adaptive memory projection (**AMP**) method for pure and mixed integer programming which combines the principle of projection techniques and the adaptive memory processes of tabu search. The idea of AMP is to generate solutions by iteratively maintaining some subsets of variables fixed at particular values while varying the values of other variables. The selection of subsets of variables depends on which are the most highly determined and consistent variables. Generally new constraints are added to the model to set certain explicit or implicit variables to some specific values. This AMP method encapsulates an important part of the previous methods. In addition to the methods/approaches mentioned above, there is also the simplex algorithm or the branch-and-cut method for which there are also some links. In this paper, we propose new convergent algorithms based on the linear programming relaxation and the mixed integer programming relaxation for solving the 0–1 MIP. Our algorithms are connected to an approach proposed by Soyster et al. (1978).

The rest of this paper is organized as follows. In Section 2 we summarize the principles of the algorithm proposed by Soyster et al. (1978) and several properties and enhancements that we have recently proposed (Hanafi and Wilbaut, 2006). In the Section 3 we present the new heuristics based on the mixed integer and linear programming relaxations of the problem. The mixed integer programming relaxation is intended to provide a form of diversification and intensification. Section 4 describes the application of the new algorithms to a simple example – a 0–1 multidimensional Knapsack problem. The computational results are presented and discussed in Section 5 in an effort to assess the performance of the proposed algorithms. In the Section 6, we present our conclusions and offer suggestions for future research.

2. Linear programming-based algorithm

At the end of the 1970s, Soyster et al. (1978) proposed an exact algorithm to solve 0–1 integer programs. In this paper, we refer to this algorithm as the linear programming-based algorithm (**LPA**). It solves a series of small-size sub problems obtained from a series of linear programming

Author's personal copy

relaxations optimally. To describe the LPA algorithm, we must first define the notion of reduced problem, which is obtained from the original problem by setting some variables at given values. Formally, given a vector $x^0 \in [0,1]^n$, let $J^0(x^0) = \{j \in N : x_j^0 = 0\}$, $J^1(x^0) = \{j \in N : x_j^0 = 1\}$, $J^*(x^0) = \{j \in N : x_j^0 \in]0,1[\}$ and $J(x^0) = \{j \in N : x_j^0 \in \{0,1\}\}$ (i.e. $J(x^0) = J^0(x^0) \cup J^1(x^0)$). Let P be an optimisation problem and C be a set of constraints; $(P|C)$ denotes the optimisation problem obtained from P by adding the constraint set C to P. The reduced problem associated to x^0 can thus be defined as follows:

$$P(x^0) = (P|x_j = x_j^0 \ \forall j \in J(x^0)). \tag{2}$$

Obviously, $P(x^0) = P$ for any vector $x^0 \in]0,1[^n$. The notion of reduced problem is useful in several different contexts. For example, at each iteration, the pivot move of the simplex method solves a reduced problem with one free variable for solving the LP-relaxation. Another example is the following preprocessing phase that tries to set some variables at their optimal values by the common rule: for any feasible solution x^0 of P and any $j \in N$, if $v(P((e+\ (1-2x_j)e_j)/2)) \leqslant c^T x^0$, where e is the vector with all its components set to 1, e_j is the vector with only component j set to 1, and $v(P)$ is the optimal value of the optimisation problem P, then either x^0 is an optimal solution or $x_j = x_j^0$ in any optimal solution of P. Note that the RINS proposed by Danna et al. (2005) solves reduced problems at some nodes when exploring a branch-and-cut tree. More precisely, if x^* is the current incumbent feasible solution and $\bar{x}$ is the solution of the continuous relaxation at the current node, RINS solves the following reduced problem

$$P((\bar{x} + x^*)/2 | c^T x > c^T x^*),$$

where $c^T x > c^T x^*$ is a constraint on the objective of the solution.

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

Proposition 1. *Given a 0–1 MIP problem P, let $\bar{x}$ be an optimal solution of the LP-relaxation LP(P) and x^0 be an optimal solution of the reduced problem $P(\bar{x})$. Then an optimal solution of P is either the feasible solution x^0 or an optimal solution of the problem*

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

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

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

Proof. It is evident that $v(P) = \max\{v(P^-), v(P^+)\}$, where $P^- = (P|\{f^T x \leqslant |J^1(\bar{x})| - 1\})$ and $P^+ = (P|\{f^T x \geqslant |J^1(\bar{x})|\})$. In addition, from the definition of the vector f, we have $f^T x \leqslant |J^1(\bar{x})|$ for each binary vector x. Consequently, the inequality $\geqslant$ can be replaced by equality in (P^+) which becomes $P^+ = (P|\{f^T x = |J^1(\bar{x})|\})$. The definition of the reduced problem $P(\bar{x})$ therefore implies that $v(P^+) = v(P(\bar{x}))$. The proposition is thus follows from the definitions of x^0 and (3), (4). □

Other proofs of this proposition are given in Soyster et al. (1978) and in Wilbaut (2006). Balas and Jeroslow (1972) call the constraints added to the problem (3) canonical cuts on the unit hypercube. The inequalities in (3) have been also 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 in AMP in diversification and intensification phases.

Algorithm 1 provides a description of the LPA algorithm. At each iteration, the LPA algorithm solves the LP-relaxation of the current problem Q to generate an optimal solution $\bar{x}$ (line 6). From this optimal solution the associated reduced problem $P(\bar{x})$ is solved exactly to generate a feasible solution x^0 for the original problem P. The reduced problem $P(\bar{x})$ is obtained from P by setting the 0–1 variables to their value in the solution $\bar{x}$ if these variables are integers. If the current best feasible solution x^* is not optimal, the current problem Q is enriched by a pseudo-cut to avoid generating the optimal basis of the LP-relaxation more than once (line 14). The process stops if the difference between the upper and the lower bounds is less than 1 (line 15). Assuming that all the data are integers, if the condition $\lfloor c^T\bar{x} - c^T x^* \rfloor < 1$ (where for a real number $\lfloor \alpha \rfloor$ identifies the greatest integer $\leqslant \alpha$) is satisfied then the final solution x^* corresponding to the lower bound is an optimal solution of the problem P.

Algorithm 1 Linear programming-based algorithm (LPA)

```
Input: A 0–1 MIP problem $P$.
Output: An optimal solution $x^*$ of $P$.
Let $x^*$ be a feasible solution of $P$ if one is available;
$Q = P$; $stop$ = False;
while $stop$ = False
  Solve the LP-relaxation of $Q$ to obtain an optimal solution $\bar{x}$;
  if $\bar{x} \in \{0,1\}^n$
    $x^* = \bar{x}$; $stop$ = True;
  end if
  Generate an optimal solution $x^0$ of the reduced problem $P(\bar{x})$;
  Update the best know-solution: if $c^T x^0 > c^T x^*$ then $x^* = x^0$;
  Generate the current cut $\{f^T x \leqslant |J^1(\bar{x})| - 1\}$ according to (3),(4)
  Update the current problem $Q$ by adding the above constraint:
        $Q = (Q|\{f^T x \leqslant |J^1(\bar{x})| - 1\})$.
  Check stopping criteria: if $\lfloor c^T\bar{x} - c^T x^* \rfloor < 1$ then $stop$ = True;
end while
Return the best solution $x^*$ of $P$ if one is generated;
```

The following theorem states the finite convergence of LPA.

Theorem 1. *The LPA algorithm converges to an optimal solution of the input problem or indicates that the problem is infeasible in a finite number of iterations.*

Author's personal copy

A proof of this theorem can be found in Hanafi and Wilbaut (2006).

Even if the LPA is convergent this algorithm is hardly usable in practice as an exact algorithm. Table 1 illustrates the convergence of the LPA on two instances of the 0–1 multidimensional Knapsack problem (see Section 4). This table provides the value of the linear programming relaxation $v(LP)$ and the value of the feasible solution generated c^Tx^0 for some iterations (*iter*). The first instance has 30 binary variables and 10 constraints. Its optimal value is equal to 376. The process is very fast and an optimal solution is obtained within 13 iterations (see Table 1). The second instance has $n = 100$ binary variables and $m = 5$ constraints; however, the process is distinctly slower. The results obtained for this instance clearly show that the convergence cannot be easily reached in practice since it required 292 iterations (see Table 1) and about 400 seconds on our machine whereas the branch-and-bound algorithm required only about 10 seconds to obtain an optimal solution. We then decided to use the LPA algorithm as a heuristic with a fixed number of iterations. Note that the algorithm can also be stopped when the gap between the upper bound and the lower bound is less than some threshold.

Most of the CPU time consumed in running the LPA is due to the exact solving of the reduced problems. Previously, Hanafi and Wilbaut (2006) proposed dominance properties to decrease the total number of exact resolutions, thus accelerating the search. Let x^1 and x^2 be two solutions in $[0,1]^n$, solution x^1 dominates solution x^2 if $J^1(x^1) \subseteq J^1(x^2)$ and $J^0(x^1) \subseteq J^0(x^2)$. If solution x^1 dominates solution x^2 then $v(P(x^1)) \geqslant v(P(x^2))$. This implies that only the reduced problems relative to the undominated optimal solutions of the LP-relaxation will be solved exactly. From this definition, we proposed a 2-phase LPA algorithm. In the first phase, the algorithm solves a series of LP-relaxations by adding at each iteration the pseudo-cut generated according to (3),(4). In the second phase, it solves only the undominated reduced problems. Implementing these properties enabled to reduce the computational effort when a maximum number of iterations was set. The results of our numerical experiments on a set of 270 instances on the 0–1 multidimensional Knapsack problem show that the average reduction in the number of problems solved exactly was 30% (see Hanafi and Wilbaut, 2006 or Wilbaut, 2006 for more details).

Table 1
Convergence illustration

GK9			OR-100-5.4		
iter	v(LP)	c^Tx^0	*iter*	v(LP)	c^Tx^0
1	380.3	336	1	23724.1	22554
2	379.7	368	2	23722.7	22983
3	379.6	360	3	23720.3	23056
4	379.5	368	4	23715.5	22606
5	378.7	368	40	23687.4	23447
8	377.9	368	41	23687.3	**23534**
9	377.6	372	42	23686.8	23402
10	377.2	368	98	23652.3	23497
11	376.9	372	99	23651.6	23486
12	376.5	364	100	23651.1	23497
13	376.4	**376**	292	23534.6	23497

As mentioned above, the convergence of the LPA is generally very slow in practice, particularly when the gap between the linear programming relaxation of the initial problem and its optimal value is large. To reduce the gap between the lower and the upper bound when the LPA is used as a heuristic (with a maximum number of iterations), we propose several new iterative heuristics based on mixed integer programming relaxations.

3. New heuristics based on mixed integer programming relaxations

In this section, we describe our new heuristics, designed to reduce the gap between the lower and upper bounds by using mixed integer programming relaxations to improve the quality of the upper bounds. Since the heuristics also introduce intensification and diversification into the search, the lower bounds are also expected to improve. Theoretically, these heuristics converge more quickly than the LPA algorithm.

3.1. Iterative mixed integer programming relaxation-based algorithm

We propose the use of the mixed integer programming (**MIP**) relaxation of the problem P relative to a subset J of N expressed as

$$\text{MIP}(P,J)\begin{cases} \max & c^Tx \\ \text{s.t.} & x \in \bar{X} \\ & x_j \in \{0,1\}\ j \in J \end{cases}.$$

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. The following proposition shows that the MIP-relaxation generally provides stronger bounds than the LP-relaxation.

Proposition 2. *For any subsets J, J' of N, such that $J' \subseteq J \subseteq N$, yields*:

$$v(P) \leqslant v(\text{MIP}(P,J)) \leqslant v(\text{MIP}(P,J')) \leqslant v(\text{LP}(P)). \qquad (5)$$

Proof. As $J' \subseteq J$, the problem $\text{MIP}(P,J')$ is a relaxation of the problem $\text{MIP}(P,J')$, therefore $v(\text{MIP}(P,J)) \leqslant v(\text{MIP}(P,J'))$. The other inequalities are deduced directly from the observation that $v(\text{MIP}(P,N)) = v(P)$ and $v(\text{MIP}(P,\varnothing)) = v(LP(P))$.

There are several ways to integrate the MIP relaxation into the LPA. First of all, as shown in Algorithm 2, the mixed integer programming relaxation could simply

Author's personal copy

replace the linear programming relaxation. We call this algorithm the mixed integer programming relaxation based-algorithm (**MIPA**). In the first iteration of the algorithm, the mixed integer programming relaxation is defined from an optimal solution of the LP-relaxation by constraining the fractional variables of the solution of this relaxation to be integers in the next iteration. Then the fractional variables in an optimal solution of the current MIP-relaxation are constrained to be integers in the next iteration (line 7; please note that if the optimal solution $\bar{x}$ of the current relaxation is integer then the process stops). This algorithm also generates a feasible solution by solving the reduced problem $P(\bar{x})$ at each iteration.

Algorithm 2 Mixed integer programming-based algorithm (MIPA)

```
**Input:** A 0–1 MIP problem $P$.
**Output:** An optimal solution $x^*$ of $P$.
Let $x^*$ be a feasible solution of $P$ if one is available;
$v^* = c^Tx^*$; $Q = P$;
Let $\bar{x}$ be an optimal solution of LP($P$); $\bar{v} = v(\mathrm{LP}(P)) = c^T\bar{x}$;
**while** $\lfloor \bar{v} - v^* \rfloor \geqslant 1$
    Let $\bar{x}$ be an optimal solution of MIP$(Q, J^*(\bar{x}))$;
    **if**$(c^T\bar{x} < \bar{v})$ **then** $\bar{v} = c^T\bar{x}$;
    **if**$(\bar{x} \in \{0,1\}^p)$ **then** $x^* = \bar{x}$; $v^* = c^T\bar{x}$;
    Let $x$ be an optimal solution of $P(\bar{x})$;
    **if**$(c^Tx > v^*)$ **then** $x^* = x$; $v^* = c^Tx$;
    $Q = (Q|\{f^Tx \leqslant |J^1(\bar{x})| - 1\})$;
**end while**
```

Theorem 2. *The MIPA algorithm converges to an optimal solution of the input problem or indicates that the problem is infeasible in a finite number of iterations bounded by* $3^n - 2^n$.

Proof. First, let x^* be an optimal solution of the original problem P. Obviously, $v(P) = v(P(x^*))$. We consider *partial vectors* x that may not have all components x_j determined. The MIPA generates partial solutions from the optimal solutions of the MIP-relaxations of the current problem Q. Since at each iteration the MIPA generates a new partial solution, the number of iterations of the MIPA is bounded by the maximum number of partial solutions. A partial solution of order k is a vector for which exactly $n-k$ variables are assigned to values 0 or 1; others variables remain free. Since there are $\binom{n}{k}2^{n-k}$ partial solutions of order k, the total number of partial solutions is $\sum_{k=0}^{n}\binom{n}{k}2^{n-k} = (1+2)^n = 3^n$. In addition, as soon as a partial solution of order 0 is reached the MIPA stops. Thus the number of iterations is bounded by $3^n - 2^n$. This completes the proof. □

Note that between two consecutive iterations the subsets of fractional variables are completely separated when applying the MIPA. Therefore, at iteration k, the following property is satisfied

$$J^*(\bar{x}^k) \cap J^*(\bar{x}^{k+1}) = \varnothing, \qquad (6)$$

where $\bar{x}^k$ is an optimal solution of the MIP-relaxation of the current problem at iteration k.

The MIP-relaxation can thus be used as a technique to diversify the search from (6). This kind of diversification does not exist in the LPA, which is why we propose to combine the two approaches to improve both the search quality and the final upper bound.

In practice the MIPA is used as a heuristic with a limited number of iterations. Moreover, if the size of 0–1 variables in the current MIP-relaxation MIP$(Q, J^*(\bar{x}))$ (i.e. $|J^*(\bar{x})|$) is large, then a relaxation of MIP$(Q, J^*(\bar{x}))$ is considered instead, for example MIP(Q, J) with $J \subset J^*(\bar{x})$.

3.2. Iterative relaxations-based heuristics

The conjoint use of the two relaxations (i.e. LP and MIP relaxations) ensures the decrease of the upper bounds sequence. In addition, adding more constraints allows the upper bounds to be refined. In general, it appears that heuristics based on the two relaxations also generate a better lower bound in the majority of the cases, as our computational results (provided in Section 5) confirm.

An instance of this kind of heuristics that we call iterative relaxation-based heuristic (**IRH**) is described in Fig. 1. In this and in the following heuristics, $\bar{x}$ (resp. $\bar{x}^k$) denotes an optimal solution of the LP-relaxation (resp. at iteration k), and $\tilde{x}$ (resp. $\tilde{x}^k$) denotes an optimal solution of the MIP-relaxation (resp. at iteration k). In the Fig. 1 the algorithm solves two relaxations and two reduced problems by iteration. The algorithm then obtains two feasible solutions and generates two pseudo constraints. At iteration k, the MIP-relaxation is obtained from $\bar{x}^k$, which insures that the upper bound generated is at least as good as the bound generated by the LP-relaxation (Proposition 2). At the next iteration, the constraints generated according to (3),(4) with the vectors $\bar{f}$ and $\tilde{f}$ are added in the problem. Note that in another version, only one reduced problem is solved at each iteration. This problem is generated from the solution x expressed as

$$x_j = \begin{cases} \bar{x}_j^k & \text{if } \tilde{x}_j^k \in]0,1[, \\ \tilde{x}_j^k & \text{if } \bar{x}_j^k \in]0,1[, \\ \bar{x}_j^k \text{ or } \tilde{x}_j^k & \text{otherwise.} \end{cases} \qquad (7)$$

The number of free variables in solution x can be controlled using a parameter. We also propose another group of heuristics, in which the two relaxations can be independent and the MIP-relaxation can be defined without the LP-relaxation. The underlying principles of these heuristics, which are called iterative independent relaxation-based heuristics (**IIRH**), are presented in Fig. 2. An initial phase is used to define the first MIP-relaxation as in the MIPA. After this initial phase, the LPA and the MIPA are applied simultaneously (one after the other from an algorithmic point of view). The best lower and upper bounds generated during the process are then memorized. As in the IRH, Eq. (7) can also be applied. One difference

Author's personal copy

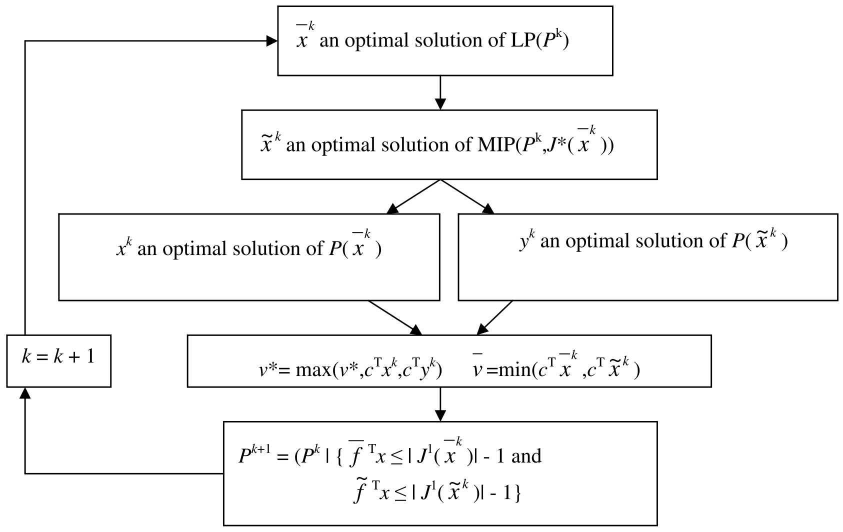
$\bar{x}^k$ an optimal solution of LP(P^k)
$\tilde{x}^k$ an optimal solution of MIP(P^k,$J^*(\bar{x}^k)$)
x^k an optimal solution of $P(\bar{x}^k)$
y^k an optimal solution of $P(\tilde{x}^k)$
$k = k + 1$
$v^*= \max(v^*,c^Tx^k,c^Ty^k)$ $\bar{v}=\min(c^T\bar{x}^k,c^T\tilde{x}^k)$
$P^{k+1} = (P^k \mid \{ \bar{f}^Tx \le |J^1(\bar{x}^k)| - 1$ and $\tilde{f}^Tx \le |J^1(\tilde{x}^k)| - 1\}$

Fig. 1. An instance of the IRH.

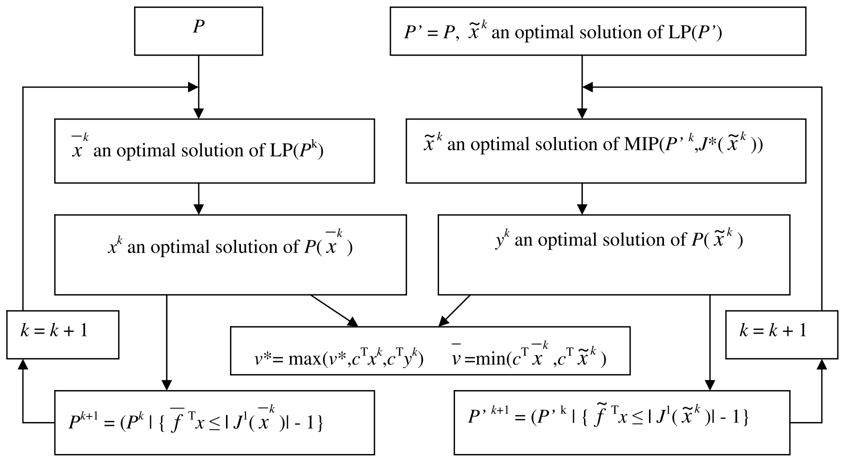
P
$P' = P$, $\tilde{x}^k$ an optimal solution of LP(P')
$\bar{x}^k$ an optimal solution of LP(P^k)
$\tilde{x}^k$ an optimal solution of MIP(P'^k,$J^*(\tilde{x}^k)$)
x^k an optimal solution of $P(\bar{x}^k)$
y^k an optimal solution of $P(\tilde{x}^k)$
$k = k + 1$
$v^*= \max(v^*,c^Tx^k,c^Ty^k)$ $\bar{v}=\min(c^T\bar{x}^k,c^T\tilde{x}^k)$
$k = k + 1$
$P^{k+1} = (P^k \mid \{ \bar{f}^Tx \le |J^1(\bar{x}^k)| - 1\}$
$P'^{k+1} = (P'^k \mid \{ \tilde{f}^Tx \le |J^1(\tilde{x}^k)| - 1\}$

Fig. 2. An instance of the IIRH.

between the IRH and the IIRH is that the intersection of the subsets of fractional variables of the solutions $\bar{x}^k$ and $\tilde{x}^k$ at iteration k is not necessarily empty for the IIRH. This means that the size of the reduced problem obtained with (7) could be smaller.

Note that parallel processing can be applied with this algorithm, one process designed for the LP-relaxation and another for the MIP-relaxation.

Heuristics IRH and IIRH have a complexity greater than the LPA because more or larger reduced problems are solved exactly during the process. However, despite this increased complexity, the quality of the bounds generated is expect to improve. The numerical results presented in Section 5 show the positive impact of these heuristics. The next section provides an example of the execution of our heuristics for a medium-sized instance of the 0–1 multidimensional Knapsack problem.

4. Illustration

In our experiments, a simplex method was used to solve the LP-relaxations and a branch-and-bound method was applied to solve the MIP-relaxations and the reduced problems exactly with the commercial software CPLEX of ILOG. This section demonstrates the process of our heuristics on an instance of the 0–1 multidimensional Knapsack problem (**MKP**). The MKP works to find a subset of items that maximizes a linear objective function while satisfying the capacity constraints. The MKP can be formulated as follows:

$$(\text{MKP}) \quad \left[\begin{array}{ll} \max & \sum_{j\in N} c_j x_j, \\ \text{s.t.} & \sum_{j\in N} a_{ij}x_j \leqslant b_i \ \forall i \in M = \{1,\dots,m\}, \\ & x_j \in \{0,1\} \ j \in N = \{1,\dots,n\}. \end{array}\right. \quad (8)$$

Author's personal copy

Table 2
Illustration for the instance GK9

iter	LPA		MIPA		IIRH		IRH	
	$\bar{v}$	$\underline{v}$	$\bar{v}$	$\underline{v}$	$\bar{v}$	$\underline{v}$	$\bar{v}$	$\underline{v}$
1	380.3	336	380.3	336	380	368	379.5	360
2	379.7	368	379.8	336	379.6	368	378.5	368
3	379.6	360	378.2	372	377.9	368	377.6	360
4	379.5	368	378.0	364	378.1	364	376.7	368
5	378.7	368	378.7	368	377	368	376.6	368
6	378.3	352	377.2	376	377.7	364	376.4	376
7	378.2	368	376.2	356	376.4	368		
8	377.9	368			376.6	368		
9	377.6	372			376.5	376		
10	377.2	368						
11	376.9	372						
12	376.5	364						
13	376.4	376						

The MKP involves n items defined by binary variables x_j, with profits $c_j > 0$ and m resources having capacities $b_i > 0$. Each item j consumes an amount $a_{ij} \geqslant 0$ from each resource i. A well-stated MKP assumes that $\max_{j \in N} a_{ij} \leqslant b_i \leqslant \sum_{j \in N} a_{ij}, \forall i \in M$.

The MKP is known to be NP-hard, but not strongly NP-hard. A very large number of papers can be found for solving the MKP (the interested reader can consult Fréville, 2004, Fréville and Hanafi, 2005 or Kellerer et al., 2004 for a comprehensive annotated bibliography). To our knowledge, the best results for commonly used benchmark instances (Beasley, 1990) are produced with the method proposed by Vasquez and Vimont (2005). Puchinger and Raidl, 2005 have recently applied a new variant of VNS for the MKP called Relaxation Guided variable neighborhood search. They show that this approach can improve the behaviour of VNS alone.

Our heuristics were applied to one MKP instance, called GK9, which has 30 variables and 10 constraints (see Table 1). All the algorithms presented in this paper were coded in 'C' language. The following notations are used for designing our algorithms.

- LPA: the iterative linear programming-based algorithm.
- MIPA: the MIP-relaxation-based algorithm.
- IRH: the iterative relaxations-based heuristic, where the constraints relative to the two relaxations (LP-relaxation and MIP-relaxation) are added to the problem at each iteration.
- IIRH: the iterative independent relaxations-based heuristic, where the constraints are added in all the problems.

To avoid overloading the presentation, the results for the major versions of the heuristics are provided in Table 2. The values of the upper bound ($\bar{v}$) and the lower bound ($\underline{v}$) obtained at each iteration (*iter*) are given for all the algorithms. As shown in Table 2, the MIPA gives an optimal solution for this instance in 6 iterations, but the solution is only proved optimal at iteration 7, when the upper bound is rounded down to 376. Here, the number of iterations necessary to reach an optimal solution is about 50% less than the number of iterations needed with the LPA. The IIRH heuristic, which combines two relaxations, needed 9 iterations to obtain an optimal solution to the problem and to prove the optimality of this solution. The most efficient version for this instance seems to be the IRH heuristic, which obtained an optimal solution and proved its optimality in 6 iterations, mainly due to the improved upper bound. For this instance, the execution times of all the algorithms are similar because of the size of the instance.

5. Computational results

The iterative relaxation-based heuristics proposed in this paper were tested on two sets of MKP instances. The first set was a collection of 270 correlated and thus hard instances, generated using the procedure proposed by Fréville and Plateau (1994). These instances are available on the OR-Library (Beasley, 1990). The 270 instances were generated with different values of m (5, 10, 30), n (100, 250, 500), and different tightness ratios ($\alpha = 0.25, 0.5, 0.75$). Ten instances were generated for each $n - m - \alpha$ combination. The second set contained 18 instances generated by Glover and Kochenberger (1996), with n equal to values from 100 to 2500 and m equal to values from 15 to 100. These problems are known to be very hard to solve using branch-and-bound methods.

As mentioned previously, all the algorithms presented in this paper were implemented in the 'C' language and compiled with "gcc" and the option -O2. The tests were carried out using a 3.4 GHz Pentium IV. All times are expressed in CPU seconds. The results presented in this section summarize the best values obtained by all the heuristics and are compared to the results obtained with CPLEX 9.0 for a 2-hour execution time as well as to the best-known solutions (Vasquez and Hao, 2001 and Vasquez and Vimont, 2005). The preliminary results showed that the MIPA was less efficient than the IRH and IIRH algorithms. For this reason, only the results of these two most efficient algorithms are presented below.

First, the results obtained by the algorithms LPA, IRH and IIRH on the OR-Library instances with $n = 100$ and $n = 250$ were compared. These algorithms LPA, IRH and IIRH globally obtained the same values as CPLEX for all these instances.

Tables 3–5 present the results obtained by the algorithms LPA, IRH and IIRH on the 90 most difficult instances in the OR-Library. These instances have 500 variables and 5, 10 or 30 constraints, and are known to be very difficult to solve exactly, as mentioned by Fréville (2004) and shown by Osario et al. (2002). Osario et al. (2002) exploit nested cut inequalities and surrogate constraints to solve 0–1 multidimensional Knapsack problems efficiently. Hanafi and Glover (2007) have shown how this method can be improved to give better results. To illustrate

Author's personal copy

Table 3
Results for the OR-500-5 instances

Problem	v^*	LPA			IIRH			IRH		
		$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$T^*(s)$	$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$T^*(s)$	$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$T^*(s)$
1	120148	19	83	119	0	59	1303	0	55	2906
2	117879	16	69	54	0	47	56	0	43	1145
3	121131	0	59	73	0	54	79	0	51	107
4	120804	10	71	33	5	61	471	0	49	3671
5	122319	0	77	4	0	71	4	0	67	8
6	122024	13	86	147	0	64	1303	0	61	1363
7	119127	0	68	19	0	62	130	0	62	117
8	120568	0	55	17	0	48	19	0	45	39
9	121575	0	67	11	0	64	14	**−11**	51	3767
10	120717	0	61	146	0	55	651	0	50	113
11	218428	2	53	253	0	47	5382	0	45	5730
12	221202	11	58	1	0	43	29	0	24	44
13	217542	8	63	47	6	52	1141	6	51	2337
14	223560	0	69	10	0	62	121	0	55	26
15	218966	4	74	1	0	62	294	0	50	582
16	220530	3	67	66	3	58	676	3	52	387
17	219989	7	69	7	0	51	308	0	45	315
18	218215	20	68	81	0	39	9	0	32	22
19	216976	0	62	1	0	46	4	0	44	12
20	219719	8	78	27	0	64	206	0	55	257
21	295828	0	45	1	0	34	2	0	28	1
22	308086	0	51	66	0	38	555	0	26	428
23	299796	0	54	9	0	47	2	0	43	2
24	306480	4	62	3	0	38	1103	0	33	318
25	300342	0	48	1	0	39	12	0	34	4
26	302571	9	70	54	0	50	393	0	40	189
27	301339	10	51	3	0	31	525	0	19	343
28	306454	0	41	14	0	34	69	0	16	26
29	302828	0	47	42	0	42	481	0	32	222
30	299910	0	47	78	0	34	1329	0	35	926

the difficulty of these instances, note that CPLEX was not able to solve one of the 30 largest instances (with $m = 30$) exactly in 2 hours. Our algorithms were launched with 120, 120 and 60 iterations for the instances when $m = 5$, 10 and 30, respectively. For each problem, Tables 3–5 report the best-known solution mentioned by Vasquez and Vimont (2005) (column v^*), the difference between this solution and our final best lower bound (column $v^* - \underline{v}$), the difference between our final upper and lower bounds (column $\bar{v} - \underline{v}$) and the time needed to obtain the lower bound in seconds (column $T^*(s)$).

As shown in Table 3, the IRH generates 27 best-known solutions and improves on one solution, while the LPA and the IIRH generate, respectively, 15 and 27 best-known solutions. Despite the identical number of best-known solutions, IIRH seems to be a little less effective than IRH. The difference between the upper and the lower bounds is not very large for these instances, although the convergence cannot be applied since the limit on the number of iterations is stronger. The execution times for the IIRH and the IRH are, on average, about 2600 and 3000 seconds, respectively, with about 190 seconds for the LPA. These results confirm that the LPA allows good lower bounds to be generated within a very reasonable time. The execution times observed for the IRH and the IIRH are reasonable when compared with the execution time mentioned by Vasquez and Vimont (2005), who cite an average execution time for each instance of nearly 50 hours using a Pentium IV 2 GHz (the RAM is not specified). While it is true that our machine has better characteristics, the difference in CPU time is too large to be ignored. CPLEX was able to obtain the best solutions of those reported in Table 3 for the 30 instances and was also able to prove the optimality of 29 of these solutions in an average of about 3100 seconds. However, as shown in the paragraphs that follow, the efficiency of CPLEX tended to decrease as m increased.

Table 4 provides the results obtained for the problems in which $m = 10$. As shown in this table, the IRH obtained 14 best-known solutions and improved upon 7; The IIRH obtained 12 best-known solutions and improved 3 solutions, and the LPA obtained only 6 best-known solutions. The difference between the upper and lower bounds is larger for these problems. The total execution time was about 730, 4500 and 4800 seconds for the LPA, the IIRH and the IRH, respectively, whereas Vasquez and Vimont (2005) reported 70 hours. Globally Table 4 confirms that the IRH and the IIRH are more effective than the LPA, with the IRH having the most efficient process. CPLEX was not able to prove the optimality of any instance among the 30 in 2 hours. Moreover, CPLEX obtained a worse (resp. the same) solution for 19 (resp. 8) instances than IRH.

Author's personal copy

Table 4
Results for the OR-500–10 instances

Problem	v^*	LPA			IIRH			IRH		
		$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$T^*(s)$	$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$T^*(s)$	$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$T^*(s)$
1	117811	2	186	449	2	176	1133	2	172	3086
2	119232	15	197	119	15	188	1685	15	185	2068
3	119215	4	173	569	4	163	643	0	156	5028
4	118813	0	218	105	0	209	65	**−12**	191	3400
5	116509	0	170	70	**−5**	155	829	**−5**	155	241
6	119504	35	214	64	0	168	1988	14	178	3130
7	119827	39	216	378	14	179	3719	33	199	2810
8	118329	20	207	373	6	183	363	**−4**	170	6146
9	117815	39	200	65	34	183	1582	34	182	2200
10	119231	65	251	318	0	178	993	**−20**	156	4282
11	217377	0	154	784	0	138	1708	0	133	9
12	219077	11	167	693	11	149	377	0	137	1727
13	217806	9	168	330	**−41**	97	33	**−41**	98	428
14	216868	17	172	196	0	132	1307	0	136	1032
15	213859	13	149	1114	7	125	3151	6	122	1707
16	215086	0	153	238	0	138	958	0	133	1317
17	217940	14	159	529	9	131	2258	0	122	4606
18	219984	0	167	96	0	144	22	**−6**	141	5729
19	214375	24	185	203	0	145	663	0	142	329
20	220899	47	206	62	27	166	1370	13	149	3174
21	304387	24	164	161	17	135	2508	0	112	5118
22	302379	43	193	60	0	126	1048	0	120	1462
23	302416	0	143	201	0	113	446	0	116	85
24	300757	10	179	160	**−27**	125	535	**−27**	120	2004
25	304374	17	194	546	0	151	4081	0	150	1561
26	301836	40	136	315	40	113	316	0	67	5314
27	304952	3	167	7	0	135	45	0	134	326
28	296478	22	157	22	6	112	2409	6	115	785
29	301359	28	187	42	2	151	1872	2	133	1015
30	307089	17	154	72	6	117	545	0	108	839

Table 5 provides the results for the largest instances ($m = 30$). A column "Cplex" was added to present the quality of the solution obtained by CPLEX ($v^* - v(Cplex)$). The results in Table 5 are less conclusive in terms of the lower bound since the IRH obtained 8 best-known solutions and improved 3 solutions only. The IIRH obtained 5 best-known solutions and improves one, and the LPA obtained only 2 best-known solutions. For these instances, the algorithms were run for 60 iterations. The lower bound value could perhaps be improved by increasing the total number of iterations, but this would increase the execution time. However, we chose to maintain a reasonable execution time: about 3600, 4600 and 5200 seconds, respectively, for the LPA, the IIRH and the IRH. This time is much less than the 100 hours mentioned by Vasquez and Vimont (2005). The difference between the upper and lower bounds is quite large, and thus convergence does not seem to be applied to these problems, even if the number of iterations is increased considerably. For this set of instances, the IRH and the IIRH clearly dominate the LPA, and the final solutions are very close to the best-known solutions. CPLEX was not able to prove the optimality of any of its solutions for these largest instances. In addition, CPLEX tended to terminate because the tree expansion exceeded the memory capacity. Thus, the IRH clearly dominates CPLEX since it obtains a better (resp. the same) solution for 22 (resp. 4) instances.

The results presented in Table 6 demonstrate that our new heuristics are more efficient than the LPA alone. This table gives the results obtained by the three algorithms (LPA, IRH and IIRH) for the 90 largest instances (with $n = 500$) given the same computational time-2000, 3500 and 5000 seconds for $m = 5$, 10 and 30, respectively. Each line is an average over 10 instances. The table gives the average value of the solutions obtained by each of the three algorithms. When only one obtained the best (resp. the worst) average value, this value is indicated in a bold (resp. italic) font. Table 6 clearly shows that the IRH improves the results obtained by the LPA. This heuristic obtains the best results for 7 classes of the 9 classes of instances. The results obtained by the IIRH are close those obtained by the LPA.

Since the results of the IIRH and the IRH were interesting for the largest OR-Library instances, we tried to improve the solutions generated by these two algorithms by increasing the number of iterations (the total number of iterations was set to 200, 200 and 100 for $m = 5$, 10 and 30, respectively). Then the IRH and the IIRH generated the last two best-known values for instances with $m = 5$ (instances 5.500–13 and 5.500–16 presented in Table

Author's personal copy

Table 5
Results for the OR-500-30 instances

Problem	v^*	LPA			IIRH			IRH			Cplex
		$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$T^*(s)$	$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$T^*(s)$	$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$T^*(s)$	
1	116056	109	651	4366	109	598	3162	108	544	747	47
2	114810	88	624	1026	88	574	515	30	472	4136	78
3	116712	51	662	123	22	581	2845	**−29**	468	3499	30
4	115329	86	682	3427	**−25**	508	2660	**−16**	473	548	50
5	116525	51	593	2183	51	551	1034	9	473	7765	66
6	115741	0	612	2277	0	560	1492	0	505	652	7
7	114181	74	566	1659	72	521	1262	70	485	5176	105
8	114348	132	613	3303	54	503	3469	4	418	3313	32
9	115419	0	467	1119	0	422	392	0	369	320	0
10	117116	93	624	2769	0	475	3445	0	429	2990	12
11	218104	36	514	2795	36	471	378	36	421	250	32
12	214648	98	508	1359	32	401	234	3	334	3880	3
13	215978	75	482	1810	36	415	2085	56	386	3431	75
14	217910	83	506	13	48	427	790	25	372	3771	35
15	215689	76	465	1047	25	384	305	49	369	3617	49
16	215890	53	478	1232	23	423	3133	**−29**	335	627	61
17	215907	107	549	846	28	415	2378	0	346	3631	24
18	216542	64	514	3772	0	403	1824	32	396	3271	103
19	217340	27	503	392	27	448	1047	27	405	3544	51
20	214739	59	516	3264	38	447	2468	49	427	4346	0
21	301675	48	399	121	32	345	361	0	290	2586	32
22	300055	41	423	1919	0	343	2232	0	313	1544	25
23	305087	7	399	1731	7	355	1819	7	336	1208	32
24	302032	27	433	4302	24	371	916	24	349	587	28
25	304462	49	467	724	37	408	3096	37	373	4449	42
26	297012	53	433	1384	24	364	2055	43	354	1264	24
27	303364	60	444	2162	42	382	2944	35	339	1505	35
28	307007	63	443	280	8	344	2462	8	328	1320	46
29	303199	56	443	2585	21	357	777	0	293	1928	37
30	300572	36	465	3865	36	427	789	0	343	1204	40

Table 6
Results obtained with the same computational time

m	α	LPA	IIRH	IRH
5	0.25	120628.7	120628.7	**120629.8**
	0.5	**219512.1**	*219510.9*	219511
	0.75	302363.4	*302363.2*	302363.4
10	0.25	*118618.8*	118621.1	**118621.7**
	0.5	217326.5	*217317.4*	**217329.3**
	0.75	*302597*	302598.3	**302600.6**
30	0.25	*115576.2*	115585.4	**115596.3**
	0.5	*216227.8*	**216244.4**	216238.6
	0.75	302420.6	*302419.8*	**302426.8**

3). Two other best-known solutions were obtained with the IRH for $m = 10$ for instances 10.500–6 and 10.500–7. For instances with $m = 30$, the IRH was able to generate three best-known solutions for problems 30.500–13, 30.500–15 and 30.500–27, while IIRH was able to do so for problem 30.500–13. As expected, the execution time of our algorithms increased commensurately. However, the highest value was about 21,500 seconds (about 6 hours) for a problem with $n = 500$ and $m = 30$. The results for the 90 largest instances of the OR-Libraries are summarized in Table 7,

Table 7
Summary of the results for the OR-500 instances

m	#Improved	#Equalled	#Lower
5	1	29	0
10	8	16	6
30	3	12	15

with the number of improved values (#Improved), equal values (#Equal) and lower values (#Lower) given for each set of 30 instances.

Table 8 presents the results for the instances generated by Glover and Kochenberger (1996). These results for the problems GK18 to Mk_gk05 were obtained after 60 iterations; the results for the last six problems (Mk_gk06–Mk_gk11) were produced after 100 iterations. These last six problems are known to be very difficult to solve using branch-and-bound approaches, thus our results are very encouraging since we obtained 12 of the best-known solutions and we improved on two solutions for the problems Mk_gk06 and Mk_gk09. The execution time for the heuristics IIRH and IRH is reasonable since it was less than 2 hours for the problems GK18 to Mk_gk05. For the largest problems, it was less than 11 hours for IRH and less than 7 hours for IIRH. Compared to the LPA, our heuristics improved the quality of the upper bound, in addition to

Author's personal copy

Table 8
Results for the instances of Glover and Kochenberger

Problem	n	m	v^*	LPA				IIRH				IRH			
				$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$iter^*$	$T^*(s)$	$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$iter^*$	$T^*(s)$	$v^* - \underline{v}$	$\bar{v} - \underline{v}$	$iter^*$	$T^*(s)$
GK018	100	25	4528	0	16	58	290	0	12	6	78	0	9	11	680
GK019	100	25	3869	0	15	36	65	0	11	6	71	0	8	2	25
GK020	100	25	5180	0	16	45	239	0	12	25	474	0	9	9	365
GK021	100	25	3200	0	17	19	27	0	13	8	58	0	10	8	245
GK022	100	25	2523	0	18	30	60	0	14	14	90	0	10	7	117
GK023	200	15	9235	2	11	36	5	0	8	40	44	0	7	40	184
GK024	500	25	9070	3	12	3	0	1	10	55	1168	0	8	45	2509
MK_gk01	100	15	3766	0	7	9	0	0	5	9	5	0	4	2	1
MK_gk02	100	25	3958	0	15	40	45	0	11	15	50	0	8	10	100
MK_gk03	150	25	5656	1	14	52	457	0	10	35	1924	0	8	3	32
MK_gk04	150	50	5767	0	25	5	222	0	18	3	282	0	15	3	472
MK_gk05	200	25	7560	0	16	49	458	0	13	25	1261	0	12	8	636
MK_gk06	200	50	7677	2	27	19	1727	**−1**	18	98	23993	**−1**	12	36	10042
MK_gk07	500	25	19220	3	15	51	287	1	10	76	8374	1	10	74	15769
MK_gk08	500	50	18806	3	27	44	3998	1	23	4	975	1	21	2	11182
MK_gk09	1500	25	58087	5	18	45	396	**−4**	7	86	15064	**−2**	8	60	17732
MK_gk10	1500	50	57295	6	29	22	1999	3	22	27	6598	3	20	5	6355
MK_gk11	2500	100	95237	9	49	25	2260	8	43	37	9463	8	43	5	1921

generating 9 solutions better than those generated by the LPA and the same solution for 9 other problems. Overall, these results show that the algorithms described in this paper are more effective for instances with a small number of constraints.

To conclude this section, we provide some of the results obtained for a set of instances extracted from the mixed integer programming library in Tables 9 and 10. We tested the algorithms on 13 instances of the 0–1 MIP problems used by Pedroso (2004). For every instance, Tables 9 and 10 present the number of binary variables in row "n'", the number of continuous variable in row "$n - n'$", and the number of constraints in row "m". The optimal value is given in row "v^*", and the value of the LP-relaxation in row "LP". In the subsequent rows, the results obtained with the LPA, the IIRH and the IRH, are provided. Values marked with an "*" correspond to an execution with more iterations (1000 rather than 100) (note that a minimization function was used in these problems).

Tables 9 and 10 show that the LPA, the IIRH and the IRH all obtained an optimal solution for the 13 tested instances. The average CPU times were about 60, 151 and 242 seconds, respectively, for the LPA, the IIRH and the IRH. The final solution was proved optimal in three (resp. 2) cases for the IIRH and the IRH (resp. the LPA). The difference between the behaviour of the three algorithms on these instances is smaller. In general, IRH obtains better lower bounds, but requires more CPU time.

Table 9
Results for some 0–1 MIP instances from the MIP-Lib

	Pb. name	egout	enigma	lseu	mod008	modglob	p0033	pk1
	n'	55	100	89	319	98	33	55
	$n - n'$	86	0	0	0	324	0	31
	m	98	21	28	6	291	16	45
	v^*	568,10	0	1120	307	20740508	3089	11
	LP	149.59	0	834.68	290.93	20430947.6	2520.6	0
LPA	$\bar{v} - \underline{v}$	287.79	0	237.81	9.54	293611	495.87	11
	$\bar{v} - v^*$	0	0*	0*	0	0	0	0
	$iter^*$	64	715*	168*	2	14	78	34
	$T^*(s)$	1	44*	3*	0	0	0	19
IIRH	$\bar{v} - \underline{v}$	124.02	0	237.81	9.53	154372	495.87	11
	$\bar{v} - v^*$	0	0*	0*	0	0	0	0
	$iter^*$	1	249*	168*	1	1	75	34
	$T^*(s)$	0	12*	15	0	18	1	261
IRH	$\bar{v} - \underline{v}$	2.31	0	95.91	7.94	93826	33.4	11
	$\bar{v} - v^*$	0	0*	0	0	0	0	0
	$iter^*$	1	247*	25	1	28	33	5
	$T^*(s)$	0	259*	1	0	273	1	49

Author's personal copy

Table 10
Results for some 0–1 MIP instances from the MIP-Lib (end)

	Pb. name	pp08a	pp08aCUTS	rgn	stein27	stein45	vpm1
	n'	64	64	100	27	45	168
	$n - n'$	176	176	80	0	0	210
	m	136	246	24	118	331	234
	v^*	7350	7350	82.19	18	30	20
	LP	2748.35	5480.6	48.79	13	22	15.4
LPA	$\bar{v} - \underline{v}$	3076.96	558.06	21.23	5	8	4.4
	$\bar{v} - v^*$	0	0	0	0	0	0
	$iter^*$	4	0	0	5	10	0
	$T^*(s)$	5	1	0	1	62	0
IIRH	$\bar{v} - \underline{v}$	461.55	337.83	21.22	0	0	4.1
	$\bar{v} - v^*$	0	0	0	0	0	0
	$iter^*$	4	1	1	0	0	1
	$T^*(s)$	7	3	0	0	0	0
IRH	$\bar{v} - \underline{v}$	77.92	26.5	0.41	0	0	2.6
	$\bar{v} - v^*$	0	0	0	0	0	0
	$iter^*$	4	1	1	5	10	1
	$T^*(s)$	9	3	0	6	180	0

6. Conclusions

In this paper, we proposed new hybrid heuristics for solving 0–1 mixed integer programs. Some of the concepts applied are connected to an exact algorithm proposed at the end of the 1970s, for which we recently proposed some extensions. The process consists of generating two sequences of upper and lower bounds by solving relaxations and sub-problems until the visit of an optimal solution to the problem can be proved. This process is used as a heuristic with a maximum number of iterations. We proposed to integrate mixed integer programming (MIP) relaxations to refine the upper bounds and to diversify the search. In most cases, the MIP-relaxation also improves the quality of the lower bounds. The results obtained on two sets of difficult instances available for the 0–1 multidimensional Knapsack problem show the efficiency of our heuristics, since we were able to improve 14 best-known solutions. In addition, we obtained encouraging results on a subset of 0–1 MIP problems.

A prospect to this work could be the implementation of the heuristic IRH using a parallel algorithm, which would decrease the execution time, thus improving the results of these heuristic. We also hope to develop other hybrid heuristics, integrating the adaptive memory processes of tabu search to explore the neighborhoods induced by the relaxations. We expect to improve the search by integrating more memory than in the algorithms presented in this paper.

Acknowledgements

This work was partially supported by the *Nord/Pas-de-Calais* Region and the European funding program FEDER. This support is gratefully acknowledged. 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 (1), 61–69.

Beasley, J.E., 1990. OR-library: Distributing test problems by electronic mail. Journal of the Operational Research Society 41 (11), 1069–1072.

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

Fischetti, M., Lodi, A., 2003. Local branching. Mathematical Programming 98 (B), 23–47.

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

Fréville, A., Plateau, G., 1994. An efficient preprocessing procedure for the multidimensional Knapsack problem. Discrete Applied Mathematics 49, 189–212.

Fréville, A., Hanafi, S., 2005. The multidimensional 0–1 Knapsack problem – bounds and computational aspects. Annals of Operations Research 139 (1), 195–227.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman.

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.

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., Wilbaut, C., 2006. Improved convergent heuristic for the 0–1 multidimensional Knapsack problem, Technical Report, University of Valenciennes, France.

Hanafi, S., Glover, F., 2007. Exploiting nested inequalities and surrogate constraints. European Journal of Operational Research 179 (1), 50–63.

Author's personal copy

Hansen, P., Mladenovic, N., Urosevic, D., 2006. Variable neighbourhood search and local branching. Computers and Operations Research 33 (10), 3034–3045.

Kellerer, H., Pferschy, U., Pisinger, D., 2004. Knapsack Problems. Springer.

Nemhauser, G.L., Wolsey, L.A., 1999. Integer and Combinatorial Optimization. Willey-Interscience, New York.

Osario, M.A., Glover, F., Hammer, P., 2002. Cutting and surrogate constraint analysis for improved multidimensional Knapsack solutions. Annals of Operations Research 117, 71–93.

Pedroso, J.P., 2004. Tabu search for mixed integer programming, Technical Report, DCC-2004-02, University of Porto, Portugal.

Puchinger, J., Raidl, G.R., 2005. Relaxation guided variable neighborhood search. In: Proceedings of the XVIII Mini EURO Conference on VNS, Tenerife, Spain.

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. In: 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., 2006. Heuristiques hybrides pour la résolution de problèmes en nombres entiers mixtes, PhD Thesis, Université de Valenciennes et du Hainaut Cambrésis.

Wilbaut, C., Hanafi, S., Fréville, A., Balev, S., 2006. Tabu search: Global intensification using dynamic programming. Control and Cybernetic 35 (3), 579–598.