Yugoslav Journal of Operations Research
25 (2015), Number 3, 343–360
DOI: 10.2298YJOR140219014H

NEW VARIABLE NEIGHBOURHOOD SEARCH BASED 0-1 MIP HEURISTICS

Saïd HANAFI[1,2,3],Jasmina LAZIĆ[4,5],Nenad MLADENOVIĆ[4,5],
Christophe WILBAUT[1,2,3],Igor CRÉVITS[1,2,3]
1*: Univ Lille Nord de France, F-59000 Lille, France*
2*: UVHC, LAMIH - F-59313 Valenciennes, France*
3*:CNRS, UMR 8201, F-59313 Valenciennes, France*
4*: Brunel University, UK*
5*: Mathematical Institute, Serbian Academy of Sciences and Arts,Serbia*
said.hanafi@univ-valenciennes.fr
christophe.wilbaut@univ-valenciennes.fr
igor.crevits@univ-valenciennes.fr
Jasmina.Lazic@brunel.ac.uk, Nenad.Mladenovic@brunel.ac.uk

Received: February 2014 / Accepted: May 2014

Abstract: In recent years many so-called matheuristics have been proposed for solving Mixed Integer Programming (MIP) problems. Though most of them are very efficient, they do not all theoretically converge to an optimal solution. In this paper we suggest two matheuristics, based on the variable neighbourhood decomposition search (VNDS), and we prove their convergence. Our approach is computationally competitive with the current state-of-the-art heuristics, and on a standard benchmark of 59 0-1 MIP instances, our best heuristic achieves similar solution quality to that of a recently published VNDS heuristic for 0-1 MIPs within a shorter execution time.

Keywords: 0-1 Mixed integer programming, Matheuristics, Variable neighbourhood search, Pseudo-cuts, Convergence.

MSC: 90C11,90C27.

1. INTRODUCTION

The 0-1 Mixed Integer Programming (0-1 MIP) problems can be expressed as follows:

$$\text{(0-1 MIP)} \qquad \max\{c^T x \mid x \in X\}, \tag{1}$$

where $X = \{x \in \mathbb{R}^n \mid Ax \leq b, x_j \geq 0 \text{ for } j = 1, \ldots, n, x_j \in \{0,1\} \text{ for } j = 1, \ldots, p \leq n\}$ is the feasible set, $c^T x$ is the objective function, and $x \in X$ are the feasible solutions. The set of indices of variables $N = \mathcal{B} \cup \mathcal{C}$ is partitioned into two subsets $\mathcal{B} = \{1, 2, \ldots, p\}$ and $\mathcal{C} = \{p+1, p+2, \ldots, n\}$, corresponding to binary and continuous variables, respectively. To simplify the notation we will use notation P to refer to the 0-1 MIP.

Wide range of practical problems in science, engineering and business can be modeled as 0-1 MIP problems (0-1 MIPs). However, a number of special cases of 0-1 MIPs is proven to be NP-hard [6], and for some of them the computational resources required to obtain an optimal solution can grow exponentially with the size of the problem instance. That is why a lot of effort has been made to improve the exact methods for 0-1 MIPs, such as branch-and-bound, branch-and-cut, branch-and-price, dynamic programming, Lagrangian relaxation, and linear programming. This, on the other hand, led to appearance of a number of well-developed and successful designed general-purpose MIP solvers, such as IBM ILOG CPLEX [17], Gurobi [12], XPRESS [2], LINGO [21] or FortMP [5].

Mathematical programming formulation of 0-1 MIP is particularly convenient for some general-purpose solver application. However, many MIP problems still cannot be solved within acceptable time and/or space limits by the best current exact methods. As a consequence, metaheuristics (general frameworks for building problem specific heuristics) have attracted attention as possible alternatives or supplements to the more classical approaches. Matheuristics combine metaheuristics and approaches relying on mathematical programming problem formulations. Often, an exact optimisation method is used as the subroutine of a metaheuristic for solving a smaller subproblem [22], though more generally, any optimisation method available through a call to a general-purpose MIP solver can be used as the subproblem subroutine within a given metaheuristic. At present, many existing general-purpose MIP solvers contain a variety of heuristic solution methods in addition to exact optimisation techniques.

Although the term *matheuristic* (short from math-heuristic, also known as *model-based heuristic*) is only recently in use, over the last decades, a number of methods for solving optimisation problems has emerged that can be considered as matheuristics. In [25], a convergent algorithm for pure 0-1 integer programming, solving a series of small subproblems generated by exploiting information obtained through a series of linear programming relaxations, was proposed. Several enhanced versions of this algorithm have also been proposed, see [14, 26]. Neighbourhood search type metaheuristics, such as Variable Neighbourhood Search (VNS), proposed in [23], Large Neighbourhood Search (LNS) introduced in [24],

or large-scale neighbourhood search [1], are proved to be very efficient when combined with optimisation techniques based on the mathematical programming problem formulations. Recently, Glover and Hanafi [9, 10] developped a more advanced approach for generating the target objective based on exploiting the mutually reinforcing notions of proximity, reaction, and resistance. In [7], an Adaptive Memory Projection (AMP) method for pure and mixed integer programming was proposed, which combines the principle of projection techniques with the adaptive memory processes of tabu search to set some explicit or implicit variables to some particular values. This idea can be used for unifying and extending a number of other procedures: LNS, Local Branching (LB) [4], the Relaxation Induced Neighbourhood Search (RINS) [3], VNS branching [16], or the global tabu search intensification using dynamic programming [27], among others. Following the ideas of LB and RINS, another method for solving 0-1 MIP problems was proposed in [20], based on the principles of Variable Neighbourhood Decomposition Search (VNDS) [15]. This method uses the solution of the linear relaxation of the initial problem to define sub-problems to be solved within the VNDS framework.

In this paper we propose two new heuristics for solving 0-1 MIP, which dynamically improve lower and upper bounds on the optimal value within VNDS. By choosing a particular strategy for updating lower and upper bounds, we define different schemes for generating a series of sub-problems. The proposed heuristics are tested and validated on instances used in [20] and extracted from the standard 0-1 MIP benchmark from MIPLIB 2003 library. The results show that:

- the proposed methods converge to an optimal solution if no limitations regarding the execution time or the number of iterations are imposed;
- our proposed algorithms are comparable with the state-of-the-art heuristics.

This paper is organized as follows. In Section 2, we provide necessary notation and definitions. In Section 3, we present the new general scheme and describe in detail the new heuristics based on the mixed integer and linear programming relaxations of the problem and the VNDS principle. Next, in Section 4, computational results are presented and discussed. In Section **??**, some final outlines and conclusions are provided.

2. NOTATION

In order to mathematically formulate the process of variable fixation, we introduce the notion of *reduced problem*. Given an arbitrary solution x^0 (not necessarily a feasible solution) and an arbitrary subset of indices corresponding to binary variables $J \subseteq \mathcal{B}$, the problem reduced from the original problem P and associated with x^0 and J can be defined as:

$$P(x^0, J) \quad \left[\begin{array}{lll} \max & c^T x & \\ \text{s.t.} & Ax \leq b & \\ & x_j = x_j^0 & \forall j \in J \\ & x_j \in \{0,1\} & \forall j \in \mathcal{B} \\ & x_j \geq 0 & \forall j \in N \end{array} \right.$$

Obviously, the reduced problem is derived from the original by setting variables with indices in J at values of x^0. We further define the sub-vector associated with the set of indices $J \subseteq \mathcal{B}$ and solution x^0 as $x^0(J) = (x_j^0)_{j \in J}$, the set of indices of binary variables with integer values as $B(x^0) = \{j \in \mathcal{B} \mid x_j^0 \in \{0,1\}\}$. We also use the short form notation $P(x^0)$ for the reduced problem $P(x^0, B(x^0))$. Apparently, $P(x^0) = P$ if $x_j^0 \in]0,1[\,, \forall j \in \mathcal{B}$. The LP-relaxation of problem P is denoted as LP(P), i.e. :

$$\text{LP}(P) \quad \left[\begin{array}{lll} \max & c^{\mathrm{T}} x & \\ \text{s.t.} & Ax \leq b & \\ & x_j \in [0,1] & \forall j \in \mathcal{B} \\ & x_j \geq 0 & \forall j \in N \end{array} \right.$$

Hanafi and Wilbaut [13], and Glover [8] separately proposed the use of the Mixed Integer Relaxation (MIR). It is defined for a given problem P and a subset of indices $J \subseteq \mathcal{B}$ by forcing variables with indices in J to be binary in an optimal solution, i.e. :

$$\text{MIR}(P, J) \quad \left[\begin{array}{lll} \max & c^{\mathrm{T}} x & \\ \text{s.t.} & Ax \leq b & \\ & x_j \in \{0,1\} & \forall j \in J \\ & x_j \in [0,1] & \forall j \in \mathcal{B} - J \\ & x_j \geq 0 & \forall j \in N \end{array} \right.$$

If C is a set of constraints, we will denote with $(P \mid C)$ the problem obtained by adding all constraints in C to the problem P. Let x and y be two arbitrary solutions of the problem P, the distance between x and y is then defined as $\delta(x,y) = \sum_{j \in \mathcal{B}} | x_j - y_j |$. If $J \subseteq \mathcal{B}$, then we define partial distance between x and y, relative to J, as $\delta(J,x,y) = \sum_{j \in J} | x_j - y_j |$ (obviously, $\delta(\mathcal{B},x,y) = \delta(x,y)$). More generally, let $\bar{x}$ be an optimal solution of the LP relaxation LP(P) (not necessarily MIP feasible), and $J \subseteq B(\bar{x})$ an arbitrary subset of indices corresponding to binary variables (by definition of $B(\bar{x})$ all the variables in J have a binary value), the

partial distance $\delta(J, x, \overline{x})$ can be linearised as follows:

$$\delta(J, x, \overline{x}) = \sum_{j \in J} \left[x_j(1 - \overline{x}_j) + \overline{x}_j(1 - x_j) \right].$$

Note that $\delta(J, x, \overline{x})$ is not defined if $J = \emptyset$.

Let X be the feasible set of the problem P. The neighbourhood structures $\{\mathcal{N}_k \mid k = k_{min}, \ldots, k_{max}\}$, $1 \leq k_{min} \leq k_{max} \leq p$ can be defined if the distance $\delta(\mathcal{B}, x, y)$ between any two solutions $x, y \in X$ is known. The set of all solutions in the k^{th} neighbourhood of $x \in X$ is denoted as $\mathcal{N}_k(x)$, where

$$\mathcal{N}_k(x) = \{y \in X \mid \delta(\mathcal{B}, x, y) \ \leq \ k\}.$$

From the definition of $\mathcal{N}_k(x)$ it follows that $\mathcal{N}_k(x) \subset \mathcal{N}_{k+1}(x)$, for any $k \in \{k_{min}, k_{min} + 1, \ldots, k_{max} - 1\}$, since $\delta(\mathcal{B}, x, y) \ \leq \ k$ implies $\delta(\mathcal{B}, x, y) \ \leq \ k + 1$. If we completely explore neighbourhood $\mathcal{N}_{k+1}(x)$, it is not necessary to explore the neighbourhood $\mathcal{N}_k(x)$.

3. NEW ADVANCED VNDS BASED HEURISTICS

It is well known that heuristics and relaxations are useful for providing upper and lower bounds on the optimal value for large and difficult optimisation problems, respectively. In [20], a hybrid approach is proposed for solving 0-1 MIP problems that combines Variable Neighbourhood Decomposition Search (VNDS) heuristic [15] and a generic MIP solver. VNDS is used to define a variable fixation scheme for generating a sequence of smaller subproblems that are normally easier to be solved than the original problem. We here consider this approach, denoted as `VNDS-MIP`, and provide its pseudo-code for maximization problems in Figure 1. This algorithm can easily be adjusted for minimization problems.

Input parameters for the algorithm are an instance P of the 0-1 MIP problem, a parameter d that defines the number of variables to be released in each iteration and an initial feasible solution x^* of P. The algorithm returns the best solution found within the stopping criteria, defined by the variable *proceed*1. Variable *proceed*2 is used to define the stopping condition of the second loop *while* in the algorithm.

Variables are ordered according to their distances from the LP relaxation solution values (see lines 5, 6 in Figure 1). More precisely, we compute distances $\delta_j =\mid x_j^* - \overline{x}_j \mid$ for $j \in \mathcal{B}$, where x_j^* is a variable value of the current incumbent (feasible) solution, and $\overline{x}_j$ a variable value of the LP-relaxation. We then index variables x_j, $j \in \mathcal{B}$, so that $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_p$, $p =\mid \mathcal{B} \mid$. Parameters k_{min}, k_{step} and k_{max} (see line 8 in Figure 1) are determined in the following way. Let q be the number of binary variables which have different values in the LP relaxation solution and in the incumbent solution ($q =\mid \{j \in \mathcal{B} \mid \delta_j \neq 0\} \mid$), and let d be a given parameter

```
VNDS-MIP(P, d, x*)
 1  Choose stopping criteria (set proceed1 = proceed2 = true);
 2  Find an optimal solution x̄ of LP(P);
 3  if (B(x̄) = B) then return x̄;
 4  while (proceed1) do
 5      Set δ_j =| x*_j − x̄_j |, j ∈ B;
 6      Index x_j so that δ_j ≤ δ_{j+1}, j = 1, ..., p − 1, p = |B|;
 7      Set q =| {j ∈ B | δ_j ≠ 0} |;
 8      Set k_min = p − q, k_step = [q/d], k_max = p − k_step, k = k_max;
 9      while (proceed2 and k ≥ 0) do
10          J_k = {1, ..., k}; x' = MIPSOLVE(P(x*, J_k), x*);
11          if (c^T x' > c^T x*) then
12              x* = LocalSearch(P, x');
13          else
14              if (k − k_step < k_min) then k_step = max{[k/2], 1};
15              Set k = k − k_step;
16          endif
17          Update proceed2;
18      endwhile
19      Update proceed1;
20  endwhile
21  return x*.
```

Figure 1: VNDS for MIPs.

(its value is experimentally found) that controls the neighbourhood size. Then, we set $k_{min} = p - q$, $k_{step} = [q/d]$, (where $[\alpha]$ is the largest integer not greater than α) and $k_{max} = p - k_{step}$. We also allow the value of k to be less then k_{min} (see lines 14 and 15 in Figure 1). In other words, we allow the variables having the same integer value as in the LP relaxation solution to be released anyway. In case that $k < k_{min}$, k_{step} is set to (approximately) the half of the number of the remaining fixed variables. Note that the maximum value of parameter k (which is k_{max}) indicates the maximum possible number of fixed variables. This value implies the minimum number of released variables, and therefore the minimum possible neighbourhood size in the VNDS scheme. Also note that in all pseudo-codes, the statement y = MIPSOLVE(P, x) denotes a call to a generic MIP solver for a given 0-1 MIP problem P, starting from a given solution x and returning a new solution y (if P is infeasible, then the value of y remains the same as before the call to the MIP solver).

In this paper, two heuristics are derived by choosing different strategies for updating lower and upper bounds, thus defining different schemes for generating a series of subproblems.

3.1. VNDS for 0-1 MIPs with Pseudo-Cuts (VNDS-PC1)

We here propose an algorithm that exactly solves a sequence of reduced problems, obtained from a sequence of linear programming relaxations. The set of reduced problems for each LP relaxation is generated by fixing a certain number of variables according to the rules of VNDS (mentioned in Figure 1). That way, two sequences of upper and lower bounds continue to be generated until the completion of an optimal solution of the problem is justified. Also, when a reduced problem is solved, a pseudo-cut is added into the problem to guarantee that this subproblem is not revisited. Furthermore, whenever an improvement in the objective function value occurs, a local search procedure is applied in the whole solution space to attempt the further improvement (so-called boundary effect within VNDS). We will refer to this method as VNDS-PC1, since it employs VNDS to solve 0-1 MIPs while incorporating pseudo-cuts to reduce the search space. The corresponding pseudo-code is provided in Figure 2 (here again, we consider the maximization case).

```
VNDS-PC1(P, d, x*)
 1  Choose stopping criteria (set proceed1 = proceed2 = true);
 2  Add objective cut: LB = c^T x*; P = (P | c^T x > LB).
 3  while (proceed1) do
 4     Find an optimal solution x̄ of LP(P); set UB = c^T x̄;
 5     if (B(x̄) = B) break;
 6     Set δ_j =| x*_j − x̄_j |, j ∈ B;
 7     Index x_j so that δ_j ≤ δ_{j+1}, j = 1, ..., p − 1, p = |B|;
 8     Set q =| {j ∈ B | δ_j ≠ 0} |;
 9     Set k_min = p − q, k_step = [q/d], k_max = p − k_step, k = k_max;
10     while (proceed2 and k ≥ 0) do
11        J_k = {1, ..., k}; x' = MIPSOLVE(P(x*, J_k), x*);
12        P = (P | δ(J_k, x', x) ≥ 1);
13        if (c^T x' > c^T x*) then
14           x* = LocalSearch(P, x'); LB = c^T x*;
15           Update objective cut: P = (P | c^T x > LB); break;
16        else
17           if (k − k_step < k_min) then k_step = max{[k/2], 1};
18           Set k = k − k_step;
19        endif
20        Update proceed2;
21     endwhile
22     Update proceed1;
23  endwhile
24  return LB, UB, x*.
```

Figure 2: VNDS for MIPs with pseudo-cuts.

The main differences between the pseudo-code of VNDS-MIP and VNDS-PC1 are the following. The LP relaxations of the current problem P is solved in the first loop in VNDS-PC1 (line 4 in Figure 2). Indeed, the problem is enriched at each iteration by a pseudo-cut (line 12). Then, variables are reordered (lines 5-6) according to the new corresponding LP solution. These relaxations generate a sequence of upper bounds, whereas the objective cut is updated each time the lower bound is improved (line 15). If an improvement occurs after solving the subproblem $P(x^*, J_k)$, where x^* is the current incumbent solution (see line 11 in Figure 2), we perform a local search on the complete solution space, starting from x' (see line 14 in Figure 2). The local search applied at this stage is the variable neighbourhood descent for 0-1 MIPs, as described in [16]. In the VNDS-PC1 algorithm, lines 20 and 22 mean that the stopping conditions of the loops are checked and updated. In particular, according to the following Proposition 1, *procced*1 can be stated as $UB - LB > 0$ (for the maximization case).

Proposition 1. *The* VNDS-PC1 *algorithm finishes in a finite number of steps and either returns an optimal solution x^* of the original problem, or proves the infeasibility of the original problem.*

Proof. Recall that $p = |\mathcal{B}|$ denotes the number of binary variables in the input problem (see (1)). The number of outer loop iterations of VNDS-PC1 is at most the number of all possible incumbent integer solutions, which is not greater than 2^p (because the cut $\delta(J_k, x', x) \geq 1$ enforces the change of incumbent integer solution in each iteration, except when $J_k = \emptyset$).

When $J_k = \emptyset$, all possible integer solution vectors have been examined. The number of inner loop iterations is bounded by $d + \log_2 p$, so the total number of iterations is at most $2^p(d + \log_2 p)$.

The pseudo-cut $\delta(J_k, x', x) \geq 1$ does not necessarily change the optimal value of the LP relaxation of P at each iteration. However, we have that $P(x^*, J_k) = (P \mid \delta(J_k, x^*, x) = 0)$ and $v(P) = \max\{v((P \mid \delta(J_k, x^*, x) \geq 1)), v((P \mid \delta(J_k, x^*, x) = 0))\}$ (where $v(P)$ denotes the optimal value of P). So, if the optimal solution of the reduced problem $P(x', J_k)$ is not optimal for P, then the cut $\delta(J_k, x^*, x) \geq 1$ does not discard the optimal value of the original problem P. We have already proved that this algorithm finishes in a finite number of steps, so it follows that either it returns an optimal solution x^* of the original problem (if $LB = UB$), or proves the infeasibility of the original problem (if $LB > UB$). ■

In practice, when used as a heuristic with the time limit as a stopping criterion, VNDS-PC1 has a good performance (see Section 4). One can observe that if pseudo-cuts (line 12 in Figure 2) and objective cuts (lines 2 and 15) are not added, the algorithm from Figure 1 is obtained as a special case of VNDS-PC1 with a fixed LP relaxation reference solution.

In this paragraph we analyze similarities and dissimilarities between our pseudo-cut strategy and the recent ones from the literature. In particular, Lasdon et al. [18] proposed two methods to solve efficiently constrained global

optimization problems. They are based on the framework of adaptive memory programming. One of those methods uses pseudo-cuts within local search procedure to prevent the search from being trapped in local optima. Hyperplanes that are orthogonal to selected rays (originating at a given original point and passing through another point) are constructed. The process is repeated to guide and to diversify the search. This work was extended recently in [11], where the authors propose some strategies to generate and to manage a pool of pseudo-cuts to guide the search, but without providing the computational results. However, their pseudo-cuts provide temporary and possibly invalid restrictions on the space of feasible solutions. On the contrary, in our approaches, the pseudo-cuts are added into the problem guaranteeing that an optimal solution of the original problem will be kept at the end of the process. We need that property to prove convergence of our matheuristics.

3.2. VNDS with pseudo-cuts and another ordering (VNDS-PC2)

In the VNDS variant discussed previously, variables in the incumbent integer solution were ordered according to the distances of their values to the values of the current LP solution. However, it is possible to employ different ordering strategies. For example, consider the following two problems:

$$(LP^{-}_{x^*}) \left[\begin{array}{ll} & \min \delta(x^*, x) \\ \text{s.t.:} & Ax \leq b \\ & c^{\mathrm{T}}x \geq LB + \epsilon \\ & x_j \in [0,1]\,,\ j \in \mathcal{B} \\ & x_j \geq 0,\ j \in N \end{array}\right. \qquad (LP^{+}_{x^*}) \left[\begin{array}{ll} & \max \delta(x^*, x) \\ \text{s.t.:} & Ax \leq b \\ & c^{\mathrm{T}}x \geq LB + \epsilon \\ & x_j \in [0,1]\,,\ j \in \mathcal{B} \\ & x_j \geq 0,\ j \in N \end{array}\right.$$

where x^* is the best known integer feasible solution and LB is the best lower bound found so far (i.e., $LB = c^{\mathrm{T}}x^*$). One can observe that ϵ can be set to a small value (e.g., 0.0001) to impose an improvement of the best lower bound in problems $LP^{-}_{x^*}$, and $LP^{+}_{x^*}$. If $\overline{x}^{-}$ and $\overline{x}^{+}$ are optimal solutions of LP-relaxation problems $LP^{-}_{x^*}$ and $LP^{+}_{x^*}$ respectively, then components of x^* could be ordered in ascending order of values $|\overline{x}^{-}_j - \overline{x}^{+}_j|$, $j \in \mathcal{B}$. Since both solution vectors $\overline{x}^{-}$ and $\overline{x}^{+}$ are real-valued (i.e., from $\mathbb{R}^n$), this ordering technique is expected to be more sensitive than the standard one, i.e., the number of pairs (j, j'), $j, j' \in N$, $j \neq j'$ for which $|\overline{x}^{-}_j - \overline{x}^{+}_j| \neq |\overline{x}^{-}_{j'} - \overline{x}^{+}_{j'}|$ is expected to be greater than the number of pairs (h, h'), $h, h' \in N, h \neq h'$ for which $|x^*_h - \overline{x}_h| \neq |x^*_{h'} - \overline{x}_{h'}|$, where $\overline{x}$ is an optimal solution of the LP relaxation $LP(P)$. Also, according to the definition of $\overline{x}^{-}$ and $\overline{x}^{+}$, it is intuitively more likely that the variables x_j, $j \in N$ for which $\overline{x}^{-}_j = \overline{x}^{+}_j$ will have the same value ($\overline{x}^{-}_j$) in the final solution, than for variables x_j, $j \in N$ for which $x^*_j = \overline{x}_j$ (and $\overline{x}^{-}_j \neq \overline{x}^{+}_j$), to have the final value x^*_j. In practice, if $\overline{x}^{-}_j = \overline{x}^{+}_j$, $j \in N$, then usually $x^*_j = \overline{x}^{-}_j$, which justifies the ordering of components of x^* in described way. However, if we want to keep the number of iterations in one pass of VNDS approximately the same as in the standard ordering (i.e., if we want to use the same value for the parameter d), then the subproblems examined will be larger than if the standard ordering is

used, since the value of q will be smaller (see line 8 in Figure 3). The pseudo-code of this variant of VNDS-PC, denoted as VNDS-PC2, is provided in Figure 3.

```
VNDS-PC2(P, d, x*)
 1  Choose stopping criteria (set proceed1=proceed2=true);
 2  Add objective cut: LB = c^T x*; P = (P | c^T x > LB);
 3  while (proceed1) do
 4      Find an optimal solution x̄ of LP(P); set UB = c^T x̄ ;
 5      if (B(x̄) = B) break;
 6      Find optimal solutions x̄^- of LP^-_{x*} and x̄^+ of LP^+_{x*};
 7      δ_j =| x̄^-_j − x̄^+_j |, j = 1..p ; index x_j so that δ_j ≤ δ_{j+1}, j = 1..p − 1 ;
 8      Set q =| {j ∈ B | δ_j ≠ 0} |, k_step = [q/d], k = p − k_step;
 9      while (proceed2 and k ≥ 0) do
10          J_k = {1, ..., k}; x' = MIPSOLVE(P(x*, J_k), x*);
11          if (c^T x' > c^T x*) then
12              Update objective cut: LB = c^T x'; P = (P | c^T x > LB);
13              x* = LocalSearch(P, x'); LB = c^T x*; break;
14          else
15              if (k − k_step > p − q) then k_step = max{[k/2], 1};
16              Set k = k − k_step;
17          endif
18          Update proceed2;
19      endwhile
20      x' = MIPSOLVE(P(x̄, B(x̄)), x*); LB = max{LB, c^T x'};
21      Add pseudo-cut to P : P = (P | δ(B(x̄), x, x̄) ≥ 1);
22      x' = MIPSOLVE(P(x̄^-, B(x̄^-)), x*); LB = max{LB, c^T x'};
23      Add pseudo-cut to P : P = (P | δ(B(x̄^-), x, x̄^-) ≥ 1);
24      x' = MIPSOLVE(P(x̄^+, B(x̄^+)), x*); LB = max{LB, c^T x'};
25      Add pseudo-cut to P : P = (P | δ(B(x̄^+), x, x̄^+) ≥ 1);
26      Update proceed1;
27  endwhile
28  return LB, UB, x*.
```

Figure 3: VNDS for MIPs with pseudo-cuts and another ordering strategy.

Proposition 2. *The* VNDS-PC2 *algorithm finishes in a finite number of steps and either returns an optimal solution x^* of the original problem (if $LB = UB$), or proves the infeasibility of the original problem (if $LB > UB$).*

Proof. It is easy to prove that the convergence of the VNDS-PC2 algorithm is guaranteed by pseudo-cuts added in lines 21, 23, and 25 of Figure 3 if no limitations regarding the execution time or the number of iterations are imposed. The proof is similar to the one in the case of linear programming based algorithm [26]. ■

4. COMPUTATIONAL RESULTS

4.1. Platform for experiments

Hardware and Software: All values presented are obtained by using a Pentium 4 computer with 3.4GHz processor and 4GB RAM and general purpose MIP solver CPLEX 11.2 [17]. We use C++ programming language to code our algorithms and compile them with g++ and the option -O2.

Methods compared: We compare the two convergent variants of the VNDS, namely `VNDS-PC1` and `VNDS-PC2` with the VNDS proposed in [20], denoted as `VNDS-MIP`. The three algorithms are executed for all the instances in the considered test bed. The results reported in [20] showed that the `VNDS-MIP` algorithm was able to obtain better results than `CPLEX` in average. Thus, we do not consider `CPLEX` in our comparison. Some other recent approaches as Local Branching (LB) [4], or LB embedded within VNS [16] can be also considered for a comparison. However, `VNDS-MIP` was already compared with those approaches in [20] and the authors showed that `VNDS-MIP` obtained in average better results.

Test bed: We consider 59 0-1 MIP problems extracted from [20] and the MIPLIB 2003 Library. These minimization problems were considered in recent papers for general 0-1 MIP approaches (in [4, 20] for instance). We only consider instances with binary and continuous variables, according to the definition of problem P at the beginning of the paper. The characteristics of this test bed are given in Table 1: the number of constraints is given in column "num. constr.", the total number of variables is given in column "num. var.". Column "num. bin." indicates the number of binary variables, whereas column "Obj." provides the optimal or the best known published objective value, according to the site of the MIPLIB 2003 and some recent papers [19, 20, 4] when no objective values are reported on the website of the MIPLIB 2003.

CPLEX parameters: As mentioned earlier, the CPLEX MIP solver is used in each method compared. We follow the choices made in [20]: We set the `CPX_PARAM_MIP_EMPHASIS` to `FEASIBILITY` for the first feasible solution, and then change to the default `BALANCED` option after the first feasible solution is found. After the first feasible solution is found, we set the local heuristics frequency (parameter `CPX_PARAM_HEUR_FREQ`) to 100.

VNDS Parameters: As a local search procedure in our algorithms, we use the variable neighbourhood descent for 0-1 MIPs (`VND-MIP`) from [16]. We decide to use the same values for parameters of VNDS for all our algorithms. We set the value of parameter d to 10. We set the maximum neighbourhood size within `VND-MIP` to 5, and the execution time limit for `VND-MIP` to 900s. The time limit for

http://miplib.zib.de/miplib2003/index.php

calls to CPLEX MIP solver for subproblems within the four algorithms is set to t_{sub} = 1200s.

Termination: All methods were run for 5h (t_{max} = 18,000s). We chose this stopping condition to favor the comparison with the results obtained by VNDS-MIP in [20].

4.2. Comparison on the MIPLIB instances

We provide a synthesis of the results for the VNDS-MIP method proposed in [20] and the two convergent algorithms proposed in this paper for the 0-1 MIP instances in Tables 2-3. In this table, we report in column "*obj*" the objective value returned by a given algorithm, and in column "*cpu**", we report the running time needed to reach this value (in seconds). The last three rows in Table 3 give additional information: the number of times the given algorithm provides the best solution among the three algorithms (row "*number of wins*"); the number of times the given algorithm obtains a strictly better solution than the other two algorithms (row "number of strict wins"); and the average running time needed to converge to the best solution visited by an algorithm. A bold value in Tables 2-3 means that the corresponding algorithm visits the best solution among the three algorithms.

Tables 2 and 3 suggest that VNDS-MIP and VNDS-PC1 clearly dominate VNDS-PC2. If we compare directly the new convergent algorithms with VNDS-MIP, we can observe the following:

- VNDS-PC1 obtains a (strictly) better objective value than VNDS-MIP for 12 instances. This method is also able to reach the same value as VNDS-MIP for 32 other instances. So, VNDS-PC1 can provide solutions at least as good as VNDS-MIP in 74.6% of the studied cases. In addition, VNDS-PC1 provides 30 solutions with better objective value than the solutions provided by VNDS-PC2.

- Even if it is globally dominated, VNDS-PC2 is also able to obtain better objective value than VNDS-MIP for 5 instances (cap6000, harp2, protfold, rail2586c, rail4284c). VNDS-PC2 reaches the same value as VNDS-MIP for 23 other instances, leading to 47.5% of the instances with a solution value at least as good as VNDS-MIP.

These results show that, if we consider only the objective value of the final feasible solution provided by a given algorithm, the convergent VNDS-PC1 algorithm is able to rival the original VNDS-MIP. In general, the other new algorithm (VNDS-PC2) has difficulties to escape from local optima during the search, and it has more difficulties to rival with VNDS-MIP. The values reported in the row "*avg. cpu**" cannot be interpreted directly since they were obtained over all the instances. This row simply shows that, in average, the VNDS-PC1 and the VNDS-PC2 algorithms converge faster to their best solution than the VNDS-MIP algorithm.

Instance	num. const.	num. var	num. bin.	Obj
10teams	230	2,025	1800	924
a1c1s1	3,312	3,648	192	11,503.44
a2c1s1	3,312	3,648	192	10,889.14
aflow30a	479	842	421	1,158
aflow40b	1,442	2,728	1,364	1,168
air04	823	8,904	8,904	56,137
air05	426	7,195	7,195	26,374
b1c1s1	3,904	3,872	288	24,544.25
b2c1s1	3,904	3,872	288	25,740.15
biella1	14,021	7,328	6,11	3,065,005.78
cap6000	2,176	6	6	-2,451,377
dano3mip	3,202	13,873	552	687.733333
danoint	664	521	56	65.6667
disctom	399	10	10	-5
ds	656	67,732	67,732	93,52
fast0507	507	63,009	63,009	174
fiber	363	1,298	1,254	405,935.18
fixnet6	478	878	378	3,983
glass4	396	322	302	1,200,012,600
harp2	112	2,993	2,993	-73,899,798.84
liu	2,178	1,156	1,089	1,138
markshare1	6	62	50	1
markshare2	7	74	60	1
mas74	13	151	150	11,801.1857
mas76	12	151	150	40,005.0541
misc07	212	260	259	2,81
mkc	3,411	5,325	5,323	-563.846
mod011	4,48	10,958	96	-54,558,535
modglob	291	422	98	20,740,508.1
momentum1	42,68	5,174	2,349	109,143
net12	14,021	14,115	1,603	214
nsrand-ipx	735	6,621	6,62	51,2
nw04	36	87,482	87,482	16,862
opt1217	64	769	768	-16
p2756	755	2,756	2,756	3,124
pk1	45	86	55	11
pp08aCUTS	246	240	64	7,35
pp08a	136	240	64	7,35
protfold	2,112	1,835	1,835	-31
qiu	1,192	840	48	-132.873137
rail2536c	2,539	15,293	15,284	689
rail2586c	2,589	13,226	13,215	947
rail4284c	4,287	21,714	21,705	1,071
rail4872c	4,875	24,656	24,645	1,534
rail507	509	63,019	63,009	174
rd-rplusc-21	125,899	622	457	165,395.28
set1ch	492	712	240	54,537.75
seymour	4,944	1,372	1,372	423
sp97ar	1,761	14,101	14,101	6.61E+08
sp97ic	1,033	12,497	12,497	429,562,635.68
sp98ar	1,435	15,085	15,085	529,814,784.70
sp98ic	825	10,894	10,894	449,144,758.40
stp3d	159,488	204,88	204,88	493.72
swath	884	6,805	6,724	467.407
t1717	551	73,885	73,885	170,195
tr12-30	750	1,08	360	130,596
UMTS	4,465	2,947	2,802	30,090,469
van	27,331	12,481	192	4.57
vpm2	234	378	168	13.75

Table 1: Characteristics for 0-1 MIP instances.

However, this solution is not necessarily the best one among the 3 algorithms (in particular for VNDS-PC2). Then, we provide in Table 4 additional information to compare the behaviour of the three algorithms. In this table, we report in column "*cpu**" the average running time (in seconds) needed to obtain the final solution by a given algorithm, and in column "*# wins*" the number of times that a given algorithm obtains this solution faster (or in the same second) than the others. As we wish to provide more accurate information about the performance of the algorithms, we only consider subsets of instances where the algorithms obtains the same solution. Then, in row "VNDS-MIP *vs* VNDS-PC1 *vs* VNDS-PC2", we only consider the 23 instances for which the three algorithms reach the same solution. We do the same comparison with the 3 possible combinations of two algorithms in the next rows (VNDS-MIP vs VNDS-PC1, VNDS-MIP vs VNDS-PC2 and VNDS-PC1 vs VNDS-PC2).

The results reported in Table 4 are clearly encouraging. Indeed, the values reported in rows "VNDS-MIP vs VNDS-PC1 vs VNDS-PC2" and in row "VNDS-MIP vs VNDS-PC1" suggest that, in average, the VNDS-PC1 algorithm is able to converge faster to its best solution than VNDS-MIP. The average difference in terms of time is not very important. However, the number of times that VNDS-PC1 converges to its final solution faster than VNDS-MIP for this subset of instances is more important (13 - 7 in the first row, 22 - 15 in the second one). Table 4 also confirms that VNDS-PC2 is, in average, dominated by VNDS-MIP and VNDS-PC1.

According to the previous observations and to the results presented in this section, we can say that the average performance of VNDS-PC1 suggest that the use of pseudo-cuts can improve the VNDS scheme initially proposed in [20] to solve hard 0-1 MIP problems.

Instance	VNDS-MIP obj	cpu*	VNDS-PC1 obj	cpu*	VNDS-PC2 obj	cpu*
10teams	**924**	8	**924**	8	1,056	3
a1c1s1	**11,503.44**	7,008	11,538.26	6,414	11,879.40	1,144
a2c1s1	**10,889.14**	16,694	11,008.87	6,007	11,559.85	184
aflow30a	**1,158**	35	**1,158**	65	**1,158**	150
aflow40b	**1,168**	10,887	**1,168**	5,102	**1,168**	1446
air04	**56,137**	112	**56,137**	46	**56,137**	3,2723
air05	**26,374**	34	**26,374**	59	**26,374**	86
b1c1s1	24,960.78	2,976	**24,948.16**	3,604	26,046.30	316
b2c1s1	26,194.59	10,896	**26,092.57**	3,771	26,438.95	434,00
biella1	**3,065,005.78**	7,978	3,065,052.45	10,125	3,070,001.58	17,809
cap6000	-2,451,372	12	**-2,451,377**	1,134	**-2,451,377**	687
dano3mip	694.42	3,417	**694.29**	2,254	740.13	4,944
danoint	**65.6667**	225	**65.6667**	35	**65.6667**	17
disctom	**-5,000**	179	**-5,000**	179	**-5,000**	137
ds	429.6	15,043	**425.7**	5,211	515.9	2,962
fast0507	**174**	3,529	**174**	453	**174**	3,796
fiber	**405,935.18**	3	**405,935.18**	< 1	**405,935.18**	< 1
fixnet6	**3,983**	2	**3,983**	3	**3,983**	1,56
glass4	**1,200,012,600**	5,020	**1,200,012,600**	2,505	1,700,010,547	1,202
harp2	-73,899,700.00	1,753	**-73,899,798.84**	1,706	**-73,899,798.84**	1,540
liu	**1,152**	15,500	**1,152**	13,174	**1,152**	11,669
markshare1	**3**	6,497	**3**	2,236	4	5,946
markshare2	**5**	1,605	7	6,593	6	8,323
mas74	**11,801.1857**	223	**11,801.1857**	49	**11,801.1857**	57
mas76	**40,005.0541**	27	**40,005.0541**	< 1	**40,005.0541**	35
misc07	**2,810**	202	**2,810**	1	**2,810**	2
mkc	**-563.53**	16,702	-560.81	1,691	-542.89	17
mod011	**-54,558,535**	222	**-54,558,535**	155	**-54,558,535**	4,351
modglob	**20,740,508.1**	1	**20,740,508.1**	1	**20,740,508.1**	38
momentum1	**109,158**	3,357	**109,158**	1,505	147,644.49	15,018
net12	**214**	3,882	**214**	4,013	255	3,121

Table 2: Results for 0-1 MIP instances (beginning).

Instance	VNDS-MIP obj	VNDS-MIP cpu*	VNDS-PC1 obj	VNDS-PC1 cpu*	VNDS-PC2 obj	VNDS-PC2 cpu*
nsrand-ipx	**51,360**	17,083	51,520	2,052	54,240	4
nw04	**16,862**	441	**16,862**	478	**16,862**	752
opt1217	**-16**	< 1	**-16**	< 1	**-16**	< 1
p2756	3,128	5	**3,124**	6	3,130	4
pk1	**11**	76	**11**	68	**11**	77
pp08aCUTS	**7,350**	10	**7,350**	3	**7,350**	147
pp08a	**7,350**	5	**7,350**	2	**7,350**	6
protfold	-23	14,512	-21	1,377	**-28**	12,251
qiu	**-132.873137**	13	**-132.873137**	46	**-132.873137**	3
rail2536c	**689.00**	364	**689.00**	689	691.00	6,324
rail2586c	966.00	13,838	**961.00**	17,657	964.00	13,107
rail4284c	1,083.00	17,148	**1,073.00**	17,843	1,078.00	17,194
rail4872c	1,558.00	17,107	**1,556.00**	12,834	1,560.00	16,824
rail507	**174.00**	2,818	**174.00**	1,941	175.00	960
rd-rplusc-21	**165,516.32**	2,957	165,842.58	3,379	184,260.88	229
set1ch	54,539.80	147	**54,537.75**	435	54,587.5	16
seymour	**423**	11,72	425	6,457	424	130
sp97ar	**662,204,945**	15,803	663,111,072	8,229	665,576,653	8,075
sp97ic	**428,561,466.56**	17,037	430,233,510.56	3,791	435,995,576.48	3,77
sp98ar	530,074,293.76	5,693	**530,047,577.44**	5,248	531,637,561.76	5,42
sp98ic	**449,144,758.40**	5,733	449,468,491.84	4,019	455,457,274.24	4,34
stp3d	**521.76**	17,624	**521.76**	16,822	**521.76**	14,981
swath	**477.56**	16,031	483.40	338	530.15	< 1
t1717	**222,859**	5,410	238,055	5,022	243,994	12,646
tr12-30	**130,596**	3,738	**130,596**	4,045	130,673	12,424
UMTS	**30,125,519.00**	1,163	30,128,228.00	3,384	30,456,926.00	1,039
van	**5.88**	13,538	**5.88**	10,820	**5.88**	17,938
vpm2	**13.75**	1	**13.75**	< 1	14	< 1
number of wins	46		43		27	
number of strict wins	14		9		2	
avg. cpu* (global)	5,064		3,476		4,501	

Table 3: Results for 0-1 MIP instances (end).

	number of instances	VNDS-MIP cpu*	VNDS-MIP # wins	VNDS-PC1 cpu*	VNDS-PC1 # wins	VNDS-PC2 cpu*	VNDS-PC2 # wins
VNDS-MIP vs VNDS-PC1 vs VNDS-PC2	23	2,067	7	2,068	13	3,850	7
VNDS-MIP vs VNDS-PC1	32	2,147	15	2,016	22	/	
VNDS-MIP vs VNDS-PC2	23	2,067	15	/		3,850	9
VNDS-PC1 vs VNDS-PC2	25	/		2,016	17	3,576	10

Table 4: Comparison between VNDS-MIP, VNDS-PC1 and VNDS-PC2.

5. CONCLUSION

Most of the discrete and continuous optimisation problems are hard to solve. The idea of combining exact solution methods by using mathematical programming formulation and metaheuristics has attracted a lot of attention recently. As a consequence, a new class of methods, called *matheuristics* (or *model-based* heuristics) has been introduced. In this paper we propose two matheuristic methods for solving 0-1 Mixed Integer Programs (MIP). We combine solutions of exact MIP solver and a Variable Neighborhood Decomposition Search (VNDS) metaheuristic. VNDS consists of systematic fixing of certain number of solution attributes and solving the remaining smaller size problems by using the basic variable neighborhood search. We also propose new variants of VNDS that integrate relaxations, pseudo-cuts, and objective cuts. The proposed VNDS's variants update both lower and upper bounds on the optimal value during the search. Our basic result is, in fact, our proof of their convergence. Based on computational analysis performed on benchmark instances from the literature, we may conclude that theoretical convergence analysis could help in designing efficient matheuristics. Moreover, the VNDS based matheuristic has great potential for solving MIP. Future work may contain more extensive computational analysis with our `VNDS-PC1` heuristic, which should include not only 0-1 MIPs. In addition, the theoretical challenge could be the design of new convergent matheuristics for mixed integer programming.

REFERENCE

[1] Ahuja, R.K., Ergun, Ö, Orlin, J.B, and Punnen, A.P., "A survey of very large-scale neighborhood search techniques", *Discrete Applied Mathematics*, 123(1-3) (2002) 75–102.

[2] Ashford, R., "Mixed integer programming: A historical perspective with Xpress-MP", *Annals of Operations Research*, 149(1) (2007) 5–17.

[3] Danna, E., Rothberg, E., and Le Pape, C., "Exploring relaxation induced neighborhoods to improve mip solutions", *Mathematical Programming*, 102(1) (2005) 71–90.

[4] Fischetti, M. and Lodi, A., "Local branching", *Mathematical Programming*, 98(2) (2003) 23–47.

[5] FortMP Manual Release 3. Numerical algorithms group limited, 1999.

[6] Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman, San Francisco, 1979.

[7] Glover, F., Adaptive memory projection methods for integer programming. In *Metaheuristic Optimization Via Memory and Evolution*, C. Rego and B. Alidaee, Eds. Kluwer Academic Publishers, 425–440, 2005.

[8] Glover, F., "Parametric Tabu Search for Mixed Integer Programs", *Computers and Operations Research*, 33 (2006) 2449–2494.

[9] Glover, F. and Hanafi, S., "Metaheuristic search with inequalities and target objectives for mixed binary optimization part I: Exploiting proximity", *International Journal of Applied Metaheuristic Computing*, 1 (2010) 1–15.

[10] Glover, F. and Hanafi, S., "Metaheuristic search with inequalities and target objectives for mixed binary optimization part II: Exploiting reaction and resistance", *International Journal of Applied Metaheuristic Computing*, 2 (2010) 1–17.

[11] Lasdon, L., Glover, F., Plummer, J., Duarte, A., Marti, R., Laguna, M., and Rego, C., "Pseudo-cut strategies for global optimization", *International Journal of Applied Metaheuristic Computing*, 2 (2011) 1–12.

[12] Gurobi Optimization Inc. Gurobi optimizer reference manual, version 2.0, 2009.

[13] Hanafi, S. and Wilbaut, C., "Heuristiques convergentes basées sur des relaxations", Presented at ROADEF 2006, Lille (february), 2006.

[14] Hanafi, S. and Wilbaut, C. "Improved convergent heuristics for the 0-1 multidimensional knapsack problem", *Annals of Operations Research*, 183(1) (2011) 125–142.
[15] Hansen, P., Mladenović, N., and Perez-Britos, D., "Variable Neighborhood Decomposition Search", *Journal of Heuristics*, 7(4) (2001) 335–350.
[16] Hansen, P., Mladenović, N., and Urošević, D., "Variable neighborhood search and local branching", *Computers & Operations Research*, 33(10) (2006) 3034–3045.
[17] ILOG. Cplex 11.2. user's manual, 2008.
[18] Lasdon, L., Duarte, A., Glover, F., Laguna, M., and Marti, R., "Adaptive memory programming for constrained global optimization", *Computers & OperationsResearch*, 37 (2010) 1500–1509.
[19] Laundy, R., Perregaard, M., Tavares, G., Tipi, H., and Vazacopoulos, A., "Solving hard mixed integer programming problems with xpress-mp: A miplib 2003 case study". Technical Report RRR 2-2007, Rutcor Research Report, January 2007.
[20] Lazić, J., Hanafi, S., Mladenović, N., and Urošević, D., "Variable neighbourhood decomposition search for 0–1 mixed integer programs", *Computers and Operations Research*, 37(6) (2010) 1055–1067.
[21] Lingo Systems Inc. Optimization modeling with lingo, 6th edition, 2006.
[22] Maniezzo, V., Stutzle, T., and Voss, S., (eds.) *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, 2009.
[23] Mladenović, N. and Hansen, P., "Variable neighborhood search", *Computers & Operations Research*, 24(11) (1997) 1097–1100.
[24] Shaw, P., "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems", *Lecture Notes in Computer Science*,Springer – Verlag, Berlin, Geidelberg, pages 417–431, 1998.
[25] Soyster, A.L., Lev, B., and Slivka, W., "Zero-One Programming with Many Variables and Few Constraints", *European Journal of Operational Research*, 2(3) (1978) 195–201.
[26] Wilbaut, C. and Hanafi, S., "New convergent heuristics for 0–1 mixed integer programming", *European Journal of Operational Research*, 195 (2009) 62–74.
[27] Wilbaut, C., Hanafi, S., Fréville, A., and Balev, S., "Tabu search: global intensification using dynamic programming", *Control and Cybernetics*, 35(3) (2006) 579–598.