**Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing
Moscow, Russia, June 3-5, 2009**

Solving 0-1 Mixed Integer Programs with Variable Neighbourhood Decomposition Search

Jasmina Lazić [*,***] **Saïd Hanafi** [**] **Nenad Mladenović** [*]
Dragan Urošević [***]

[*] *Brunel University, West London UB8 3PH, UK*
[**] *LAMIH - Universite de Valenciennes, ISTV 2 Le Mont Houy, 59313 Valenciennes Cedex 9, France*
[***] *Mathematical Institute, Serbian Academy of Sciences and Arts, Kneza Mihaila 36, 11000 Belgrade, Serbia*

Abstract: In this paper we propose a new heuristic for solving 0-1 mixed integer programs based on the variable neighbourhood decomposition search principle. It combines variable neighbourhood search with general-purpose CPLEX MIP solver. We perform systematic hard variables fixing (or diving) following the variable neighbourhood search rules. Variables to be fixed are chosen according to their distance from the corresponding linear relaxation solution values. If there is an improvement, variable neighbourhood descent branching is performed as the local search in the whole solution space. Numerical experiments have proven that by exploiting boundary effects in this way, solution quality can be considerably improved. With our approach, we have managed to improve the best known published results for 8 out of 29 instances from a well-known class of very difficult MIP problems. Moreover, computational results show that our method outperforms CPLEX MIP solver, as well as three other recent most successful MIP solution methods.

Keywords: Mathematical Programming, 0-1 Mixed Integer Programming, Metaheuristics, Variable Neighbourhood Search, Diving, Soft Variable Fixing.

1. INTRODUCTION

The 0-1 Mixed Integer Programming (0-1 MIP) problem consists of maximizing or minimizing a linear function, subject to equality or inequality constraints and binary choice restrictions on some of the variables. The zero-one mixed integer programming problem (P) can be expressed as:

$$(P) \quad \begin{bmatrix} \min c^{\mathrm{T}}x & \\ \text{s.t.} \quad Ax \geq b & \\ \qquad x_j \in \{0,1\} & \forall j \in \mathcal{B} \neq \emptyset \\ \qquad x_j \geq 0 & \forall j \in \mathcal{C} \end{bmatrix}$$

where the set of indices $N = \{1, 2, \ldots, n\}$ 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.

In this paper we propose a new approach for solving 0-1 MIP which combines variable neighbourhood decomposition search (VNDS) with exact solution method.

We first find an initial feasible solution and an optimal solution of the LP-relaxation of the problem using CPLEX solver. Trivially, if the LP-solution is 0-1 MIP feasible, we stop the search. Otherwise, at each iteration, we rank variables in nondecreasing order of their absolute values of the difference between the LP-solution and the incumbent solution values. Subproblems within VNDS are obtained by successively fixing certain number of variables in the order provided. In that way, the subproblem involves the free variables that are furthest from their linear relaxation values. Then those subproblems are solved exactly or within CPU time limit. The subproblems are changed according to VNS rules.

This paper is organised as follows. In the second section we provide a brief overview of the existing approaches related to our work. In the third section we describe VNDS heuristic. In the fourth section we provide detailed description of our VNDS implementation for solving 0-1 MIP problems. Next, in the fifth section, we analyse performance of VNDS method as compared to other three methods mentioned and the CPLEX solver alone. At last, in the sixth section, we give some final remarks and conclusions.

2. PRELIMINARIES

In this section, we first present mathematical notations and then survey of closely related recent work.

2.1 Notations and Definitions

To formally describe the process of variables fixing, we introduce the notion of reduced problem. Let x^0 be an arbitrary binary solution, and $J \subseteq \mathcal{B}$ an arbitrary subset of binary variables. The reduced problem associated with x^0 and J can be defined as

$$P(x^0, J) \qquad \min\{c^T x \mid x \in X, x_j = x_j^0 \ \forall j \in J\} \qquad (1)$$

where $X = \{x \in \mathbb{R}^n \mid Ax \geq b, x_j \geq 0 \text{ for } j = 1, \dots, n, x_j \in \{0,1\} \text{ for } j = 1, \dots, p\}$. We will also use notation $x^0(J) = (x_j^0)_{j \in J}$, to define the sub-vector associated with the set of indices J and solution x^0. The LP-relaxation of problem P, that results from dropping the integer requirement on x, is denoted as LP(P).

Given two solutions x and y of the problem P and a subset of indices $J \subseteq \mathcal{B}$, we define the distance between x and y as $\delta(J, x, y) = \sum_{j \in J} \mid x_j - y_j \mid$. More generally, if $\bar{x}$ is an optimal solution of the LP relaxation LP(P) (not necessarily MIP feasible), we also define the distance between x and $\overline{x}$ as $\delta(J, x, \overline{x}) = \sum_{j \in J} \mid x_j - \overline{x}_j \mid$.

Let X be the solution space of the problem P considered. The neighbourhood structures $\{\mathcal{N}_k \mid k = 1, \dots, k_{max}\}$ can be defined, knowing the distance $\delta(\mathcal{B}, x, y)$ between any two solutions $x, y \in X$. The set of all solutions in the kth neighbourhood of $x \in X$ is denoted as $\mathcal{N}_k(x)$ where

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

Finally, if Q is a set of constraints, we will denote with $(P \mid Q)$ the problem obtained by adding all constraints in Q to the original problem P.

2.2 Local Branching

Local Branching (LB) was introduced by Fischetti and Lodi in 2003 Fischetti and Lodi (2003), as a branching strategy for MIP problems which can also be effectively used as a heuristic improving the incumbent solution. The usage of local branching as an improvement heuristic relies on the observation that the neighbourhood of a feasible 0-1 MIP solution (in terms of the rectangular distance) often contains better solutions. More precisely, let x' be the incumbent solution, x^* the best known solution found and $\varepsilon \geq 0$ a small nonnegative real number. The local branching heuristic solves the subproblem $P(k, x') = (P \mid \{\delta(\mathcal{B}, x, x') \leq k, \ c^{\mathrm{T}}x \leq c^{\mathrm{T}}x^* - \varepsilon\})$.

Local branching combines local search with a generic MIP solver, which is used as a black box for exactly solving problems $P(k, x')$ in an iterative process.

2.3 Variable Neighbourhood Search Branching

Variable neighbourhood search (VNS) is a metaheuristic which is based on the systematic change of neighbourhoods, towards the local optima, as well as out of the regions that contain them Mladenović and Hansen (1997); Hansen and Mladenović (2001). The pseudo-code of the step that performs systematic change is given in figure 2.3. There x, x' and k denote the incumbent solution, new solution and the neighbourhood index respectively.

VNS Branching (VNSB) is a heuristic for solving MIP problems, using the general-purpose MIP solver CPLEX as a black-box Hansen et al. (2006). VNSB adds constraints to the original problem, as in LB method. However, in VNSB neighbourhoods are changed in a systematic manner, according to the rules of the general VNS algorithm Glover et al. (2003). The main idea of both LB and VNSB for solving MIP is in fact change of the neighbourhood during the search. Therefore, LB could be seen as a specialized variant of VNSB (or vice versa).

```
Procedure NeighbourhoodChange($x, x', k$)
1  if $c^T x' < c^T x$ then
2      $x \leftarrow x'$, $k \leftarrow 1$; // Make a move.
3  else $k \leftarrow k + 1$; // Next neighbourhood.
```

Fig. 1. Neighbourhood change pseudo code.

Variable neighbourhood descent (VND) is a variant of VNS in which neighbourhoods are changed deterministically. Both VNSB and present paper use the implementation of VND with only time limit as a stopping criterium. The pseudo code for the VND procedure is given in figure 2.3, where the statement $y =$ LocalSearch(P, x) means that some local search technique is applied to the problem P, starting from x as the initial solution, with solution y as a result.

```
Procedure VND($P, x, t_{vnd}$)
1  repeat
2      $k = 1$; $t_{start} = cpuTime(\,)$;
3      repeat
4          $x' =$ LocalSearch($P(k, x), x$);
5          NeighbourhoodChange($x, x', k$);
6          $t_{end} = cpuTime(\,)$, $t = t_{end} - t_{start}$;
7      until $c^T x' = c^T x$ or $t > t_{vnd}$;
8  until $t > t_{vnd}$;
9  return $x'$.
```

Fig. 2. VND pseudo code.

2.4 Relaxation Induced Neighbourhood Search

The relaxation induced neighbourhoods search (RINS for short), proposed by Danna et al. in 2005, solves reduced problems at some nodes of a branch-and-bound tree when performing tree search. It is based on the observation that often an optimal solution of a 0-1 MIP and an optimal solution of its LP relaxation have some variables that have the same values. Therefore, it seems justifiable to fix the values of those variables and then solve the remaining subproblem, in order to obtain a 0-1 MIP feasible solution with a good objective value. Based on this idea, at a node of branch-and-bound tree, RINS heuristic performs the following procedure: (i) fix the values of variables which are the same in the current continuous relaxation and incumbent integral solution, (ii) set the objective cutoff to the value of the current incumbent solution and (iii) solve MIP subproblem on the remaining variables.

3. VARIABLE NEIGHBOURHOOD DECOMPOSITION SEARCH

Variable neighbourhood decomposition search (VNDS) is a two level VNS scheme based upon the decomposition of the problem Hansen et al. (2001). Its steps are presented in figure 3, where the statement $y =$ LS(P, x, t) means that a local search technique is applied to the problem P, starting from x as the initial solution, with a given running time limit t and with solution y as a result.

```
Procedure VNDS($P, x, k_{min}, k_{max}, t_{max}, t_{sub}, t_{vnd}$)
 1  repeat
 2     $k = k_{min}$; $t_{start} = cpuTime()$;
 3     repeat
 4        Choose randomly $J_k \subseteq \mathcal{B}$ such that $| J_k |= k$;
 5        $x'(J_k) = x(J_k)$; $\overline{J}_k = \mathcal{B} \setminus J_k$;
 6        $x'(\overline{J}_k) = \mathtt{LS}(P(x, J_k), x(\overline{J}_k), t_{sub})$;
 7        if $c^T x' < c^T x$
 8           then $x'' = \mathtt{VND}(P, x', t_{vnd})$;
 9        else $x'' = x'$;
10        NeighbourhoodChange($x, x'', k$);
11        $t_{end} = cpuTime()$; $t = t_{end} - t_{start}$;
12     until $(k = k_{max})$ or $(t > t_{max})$;
13  until $t > t_{max}$;
14  return $x''$;
```

Fig. 3. VNDS pseudo code.

Input parameters for the VNDS algorithm are 0-1 MIP problem P, initial solution x, maximal number of neighbourhoods to be explored k_{max}, maximal running time allowed t_{max}, time allowed for the inner local search procedure t_{sub} and time allowed for the VND procedure t_{vnd}.

At each iteration, VNDS chooses randomly a subset of indices $J_k \subseteq \mathcal{B}$ with cardinality k. Then a local search is applied to the subproblem $P(x, J_k)$ where variables with indices from J_k are fixed to their values of the current incumbent solution x. The local search starts from the current solution $x(\overline{J}_k)$, where $\overline{J}_k = \mathcal{B} \setminus J_k$, to obtain the final solution x' and it operates only on free variables indiced by $\overline{J}_k$. If an improvement is reached, then VND is launched, starting from the solution x' to obtain the solution x''. The VND heuristic works in the whole space where all variables are allowed to be changed. Finally, experiments have shown that solution quality can be considerably improved by making use of boundary effects, so the local search (VND) is once more applied to x'' (in the whole solution space), in case that x' is better than the starting solution x. More precisely, when improved solution is obtained in the subproblem, it is likely that even better improvement can be obtained when that solution is embedded in the whole solution space. In that case, points on the boundary between the subproblem solution space and the whole search space will be most affected. This phenomenon is known as *boundary effect.*

In recent years, similar decomposition strategies for solving MIP problems have also been proposed in few other approaches (see for instance Glover (2005) or Wilbaut and Hanafi (2008)).

4. VNDS FOR 0-1 MIP WITH GENERAL-PURPOSE MIP SOLVER AS A BLACK BOX

In order to employ the idea mentioned above, we first solve LP-relaxation of the original problem P to obtain an optimal solution $\overline{x}$ and we find an initial feasible solution x. At each iteration of VNDS procedure, we compute the distances $\delta_j =| x_j - \overline{x}_j |$ from the current incumbent solution values $(x_j)_{j\in\mathcal{B}}$ to their corresponding LP-relaxation solution values $(\overline{x}_j)_{j\in\mathcal{B}}$ and index the variables $x_j, j \in \mathcal{B}$ so that $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_p$. Then we solve the subproblem $P(x, \{1, \ldots, k\})$ obtained from the original problem P, where the first k variables are fixed to their values in the current incumbent solution x. If an improvement occurs, the local search VND is performed over the whole search space and the process is repeated. Otherwise, the number of fixed variables in the current supbroblem is decreased. Note that by fixing only the variables whose distance values are equal to zero, i.e. setting $k = \max\{j \in \{1, \ldots, p\} \mid \delta_j = 0\}$, RINS scheme is obtained.

The pseudo code for VNDS for the 0-1 MIP, called VNDS-MIP, is given in figure 4. Input parameters for the VNDS-MIP algorithm are instance P of $0 - 1$ MIP problem, parameter d, that defines the value of variable k_{step}, i.e. defines the number of variables to be unfixed in each iteration of the algorithm, maximal running time allowed t_{max}, time allowed for solving subproblems t_{sub} and time allowed for the VND procedure t_{vnd}. Throughout the pseudo code calls to general MIP solver are denoted as x' = MIPSOLVE(P, t, x), meaning that the solver is called for the problem instance P, with the solving time limit t and starting solution supplied x, where x' designates the best solution found.

```
VNDS-MIP($P, d, t_{max}, t_{sub}, t_{vnd}, t_{mip}, rhs_{max}$)
 1  Find an optimal solution $\overline{x}$ of LP($P$).
 2  Find the first feasible 0-1 MIP solution $x$ of $P$.
 3  Set $t_{start} = cpuTime()$, $t = 0$.
 4  while $(t < t_{max})$
 5     Compute $\delta_j =| x_j - \overline{x}_j |$ for $j \in \mathcal{B}$, and index
       variables $x_j, j \in \mathcal{B}$ so that $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_p$.
 6     Set $n_d =| \{j \in \mathcal{B} \mid \delta_j \neq 0\} |$, $k_{step} = [n_d/d]$,
       $k = p - k_{step}$;
 7     while $(t < t_{max})$ and $(k > 0)$
 8        $x' =$ MIPSOLVE($P(x, \{1, 2, \ldots, k\})$, $t_{sub}$, $x$);
 9        if $(c^T x' < c^T x)$ then
10           $x =$ VND-MIP($P, t_{vnd}, t_{mip}, rhs_{max}, x'$);
             break;
11        else
12           if $(k - k_{step} > p - n_d)$ then
                $k_{step} = \max\{[k/2], 1\}$;
13           Set $k = k - k_{step}$;
14           Set $t_{end} = cpuTime()$, $t = t_{end} - t_{start}$;
15        endif
16     endwhile
17  endwhile
18  return $x$.
```

Fig. 4. VNDS for MIPs.

Input parameters for the VND-MIP algorithm are instance P of $0 - 1$ MIP problem, total running time allowed t_{vnd}, time allowed for the MIP solver t_{mip}, maximal size of neighbourhood to be explored rhs_{max}, and starting solution x'. The output is new solution obtained.

At each iteration of VND-MIP algorithm, the pseudo-cut $\delta(\mathcal{B}, x', x) \leq rhs$, with the current value of rhs is added to the current problem, and CPLEX solver run to get the next solution x''. Then the following steps depend on the status of the CLPEX solver. In case that optimal or feasible solution is found, it becomes new incumbent and search continues from its first neighbourhood ($rhs = 1$). Furthermore, if optimality is proven, we do not consider the current neighbourhood in further solution space exploration, so the current pseudo-cut is reversed into the complementary one. In case of nonfeasibility, neighbourhood size is increased by one. Finally, if running time limit is reached or the maximal size of neighbourhood is exceeded, VND algorithm is terminated. The VND pseudo code is given in figure 5.

```
VND-MIP($P, t_{vnd}, t_{mip}, rhs_{max}, x'$)
 1  $rhs = 1$; $t_{start} = cpuTime(\,)$; $t = 0$;
 2  while ($t < t_{vnd}$ and $rhs \leq rhs_{max}$) do
 3      $TimeLimit = \min(t_{mip},\, t_{vnd} - t)$;
 4      add the pseudo-cut $\delta(\mathcal{B}, x', x) \leq rhs$;
 5      $x'' = $ MIPSOLVE($P$, $TimeLimit$, $x'$);
 6      switch solutionStatus do
 7        case "optSolFound":
 8            reverse last pseudo-cut into
                 $\delta(\mathcal{B}, x', x) \geq rhs + 1$;
 9            $x' = x''$; $rhs = 1$;
10        case "feasibleSolFound":
11            reverse last pseudo-cut into
                 $\delta(\mathcal{B}, x', x) \geq 1$;
12            $x' = x''$; $rhs = 1$;
13        case "provenInfeasible":
14            remove last pseudo-cut;
15            $rhs = rhs + 1$;
16        case "noFeasibleSolFound":
17            Go to 20;
18      end
19      $t_{end} = cpuTime(\,)$; $t = t_{end} - t_{start}$;
20  end
21  return $x''$.
```

Fig. 5. VND for MIPs.

Hence, in our algorithm we combine two approaches: hard variable fixing in the main scheme and soft variable fixing in the local search.

5. COMPUTATIONAL RESULTS

In this section we will present computational results for our algorithm. All results reported in this section are obtained on Pentium 6 computer with 2.4GHz processor and 4GB RAM, using General purpose MIP solver CPLEX 10.1 ILOG (2006). Algorithms were implemented in C++ and compiled within Microsoft Visual Studio 2005.

Methods compared. The VNDS is compared with the four other recent MIP solution methods: Variable Neighbourhood Search Branching (VNSB) Hansen et al. (2006), Local Branching (LB) Fischetti and Lodi (2003), Relaxation Induced Neighbourhood Search (RINS) Danna et al. (2005) and CPLEX MIP solver (with all default options but without RINS heuristic). The VNSB and the LB use CPLEX MIP solver as a black box. The RINS heuristic is directly incorporated within a CPLEX branch and cut tree search algorithm.

Test bed. The 29 test instances that we consider here for comparison purposes are the same as those previously used for testing performances of LB and VNSB (see Fischetti and Lodi (2003), Hansen et al. (2006)) and most of the instances used for testing RINS (see Danna et al. (2005)).

Termination. All methods were run for 5 hours ($t_{max} = 18,000$ seconds), the same as in the papers about Local Branching (Fischetti and Lodi (2003)) and VNSB (Hansen et al. (2006)). An exception is the `NSR8K` which is the largest instance in test bed. Due to the long time needed to get the first feasible solution (more than 13,000 seconds), we decided to allow 15 hours for solving this problem ($t_{max} = 54,000$).

VNDS Implementation. In order to evaluate performance of the algorithm and its sensitivity to parameters values, we tried out different parameters settings. As the result of our preliminary testing, we got two variants of VNDS for MIP that differ only in the set of parameters used for the inner VND subroutine. Moreover, we found an automatic rule how to switch between these two variants. The details are given below.

VNDS with the first VND version (VNDS1). In the first version we do not restrict the size of neighbourhoods to explore, neither the time for the MIP solver (apart from the overall time limit for the whole VND procedure). In this version the number of parameters is minimized (following the main idea of VNS that there should be as few parameters as possible). Namely, we set $t_{mip} = \infty$ and $rhs_{max} = \infty$, leaving the input problem P, the total time allowed t_{vnd} and initial solution x' as the only input parameters. This way, there are four input parameters for the whole VNDS algorithm (apart from input problem P): d, t_{max}, t_{sub} and t_{vnd}. We set $d = 10$ in all cases[1], total running time allowed t_{max} as stated above, $t_{sub} = 1200s$ and $t_{vnd} = 900s$ for all models except `NSR8K`, for which we put $t_{sub} = 3600s$ and $t_{vnd} = 2100s$.

VNDS with the second VND version (VNDS2). In the second version of VND procedure, we aim to reduce the search space and in that way speed up the solution process. Therefore, we limit the maximal size of neighbourhood that can be explored, as well as the time allowed for the MIP solver. Values for parameters d and t_{max} are the same as in the first variant, and the other settings are as following: $rhs_{max} = 5$ for all instances, $t_{sub} = t_{vnd} = 1200s$ for all instances except `NSR8K`, $t_{sub} = t_{vnd} = 3600s$ for `NSR8K`, and $t_{mip} = t_{vnd}/d$ (i.e. $t_{mip} = 360s$ for `NSR8K` and $t_{mip} = 120s$ for all other instances). So, the number of parameters for this second variant of VNDS is again limited to four (not including input problem P): d, t_{max}, t_{vnd} and rhs_{max}.

[1] d is the number of groups in which variables (that differ in the incumbent integral and linear relaxation solution) are divided in order to define the increase k_{step} of neighbourhood size within VNDS (see figure 4).

Problems classification. We say that MIP model P is *computationally demanding with respect to time limit T*, if the time needed for default CPLEX MIP optimiser to solve it is greater than $2T/3$, where T is a maximum time allowed for a call to CPLEX MIP optimiser. We say that MIP model P is *computationally non-demanding with respect to time limit T*, if it is not computationally demanding with respect to T. Since the time limit for all problems in our test bed is already given, we will refer to computationally demanding problems with respect to 5 hours (or 15 hours for `NSR8K` instance) as *demanding problems*. Similarly, we will refer to computationally non-demanding problems with respect to 5 hours as *non-demanding problems*.

As the step of our final method, we choose to apply VNDS1 to non-demanding problems and VNDS2 to demanding problems. In figure 6 we give average performance of those two variants over the problems in test bed. As predicted, it is clear that in the early stage of solution process, heuristic VNDS2 improves faster. However, later, due to the longer time allowed for solving subproblems, VNDS1 improves its

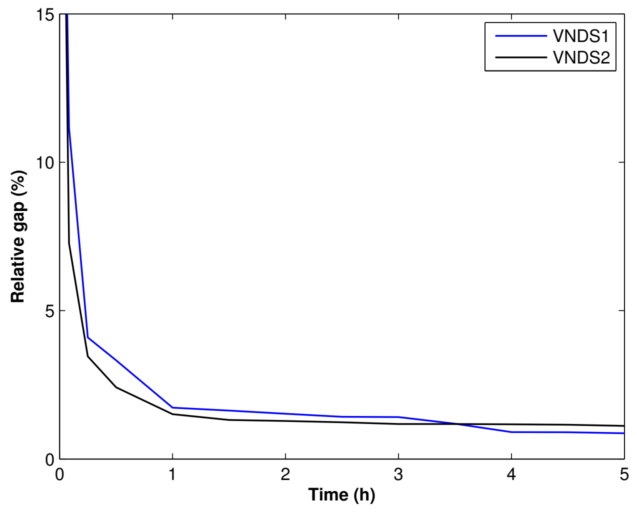

Fig. 6. Test bed gap average .

Comparison. In table 1 we present objective function values for the methods tested.[2] Here we report values obtained with one of the two parameters settings selected according to our automatic rule explained earlier. For each instance, the best of the five values obtained in our experiments is bolded, and values that are better than the currently best known are marked with a star. It appears:

(i) With our VNDS based heuristics we improved the best published result for as many as 8 test instances (out of 29). VNSB got the new best known result in 3 cases, and local branching and RINS got it for one instance; CPLEX alone did not improve any of the best known objective values.
(ii) With our VNDS based heuristic we were able to reach the best result among all the five methods in 16 out of 29 cases, whereas RINS heuristic got the best result in 12 cases, VNS branching in 10 cases, CPLEX alone in 6 and local branching in 2 cases.

[2] Instances in table 1 are given in the same order as in table 2 and table 3, but their full names are omitted due to the lack of space.

Table 1. Objective function values.

In.	VNDS	VNSB	LB	CPLEX	RINS
1	-561.9	**-563.9**	-560.4	**-563.9**	**-563.9**
2	**467.4**	**467.4**	477.6	509.6	524.2
3	**65.7**	**65.7**	**65.7**	**65.7**	**65.7**
4	**3.0***	**3.0***	12.0	5.0	7.0
5	**8.0***	12.0	14.0	15.0	17.0
6	**7580813.1**	7580889.4	7581918.4	7581076.3	7581007.5
7	425.0	**423.0**	424.0	424.0	424.0
8	20752809.0	21157723.0	**20449043.0***	164818990.4	83340960.0
9	**174.0**	**174.0**	176.0	**174.0**	**174.0**
10	**689.0**	691.0	691.0	**689.0**	**689.0**
11	957.0	960.0	956.0	959.0	**954.0**
12	1075.0	1085.0	1075.0	1075.0	**1074.0**
13	1552.0	1561.0	**1546.0**	1551.0	1548.0
14	1550009237.6	**1400013000.0***	1600013800.0	1575013900.0	1460007793.6
15	**4.6***	4.8	5.1	5.4	5.1
16	**3065005.8***	3142409.1	3078768.5	3065729.1	3071693.3
17	**30090469.0***	30127927.0	30128739.0	30133691.0	30122984.0
18	**214.0**	255.0	255.0	255.0	**214.0**
19	12930.0	**12890.0**	12899.0	**12890.0**	12899.0
20	**51200.0***	51520.0	51360.0	51360.0	51360.0
21	**11503.4***	11515.6	11554.7	11505.4	**11503.4***
22	10958.4	10997.6	10891.8	**10889.1**	**10889.1**
23	24646.8	25044.9	24762.7	24903.5	**24544.3**
24	25997.8	25891.7	25857.2	25869.4	**25740.2**
25	**130596.0**	130985.0	130688.0	**130596.0**	**130596.0**
26	**662156718.1***	662221963.5	662824570.6	670484585.9	662892981.1
27	431596203.8	**427684487.7***	428035177.0	437946706.6	430623977.0
28	530232565.1	**529938532.2**	530056232.3	536738808.5	530806545.3
29	**449144758.4**	**449144758.4**	449226843.5	454532032.5	449468491.8

Table 2. Relative gap values (in %).

Instance	VNDS	VNSB	LB	CPLEX	RINS
mkc	0.337	**0.001**	0.607	**0.001**	**0.001**
swath	**0.000**	**0.000**	2.174	9.017	12.149
danoint	**0.000**	**0.000**	0.005	**0.000**	**0.000**
markshare1	**0.000**	**0.000**	300.000	66.667	133.333
markshare2	**0.000**	50.000	75.000	87.500	112.500
arki001	**0.000**	0.001	0.015	0.003	0.003
seymour	0.473	**0.000**	0.236	0.236	0.236
NSR8K	1.485	3.466	**0.000**	705.999	307.554
rail507	**0.000**	**0.000**	1.149	**0.000**	**0.000**
rail2536c	**0.000**	0.290	0.290	**0.000**	**0.000**
rail2586c	1.056	1.373	0.950	1.267	**0.739**
rail4284c	0.373	1.307	0.373	0.373	**0.280**
rail4872c	1.173	1.760	**0.782**	1.108	0.913
glass4	10.714	**0.000**	14.286	12.500	4.285
van	**0.000**	5.790	11.285	17.041	11.251
biella1	**0.000**	2.525	0.449	0.024	0.218
UMTS	**0.000**	0.124	0.127	0.144	0.108
net12	**0.000**	19.159	19.159	19.159	**0.000**
roll3000	0.310	**0.000**	0.070	**0.000**	0.070
nsrand_ipx	**0.000**	0.625	0.313	0.313	0.313
a1c1s1	**0.000**	0.106	0.445	0.017	**0.000**
a2c1s1	0.636	0.996	0.024	**0.000**	**0.000**
b1c1s1	0.418	2.040	0.890	1.464	**0.000**
b2c1s1	1.001	0.589	0.455	0.502	**0.000**
tr12-30	**0.000**	0.298	0.070	**0.000**	**0.000**
sp97ar	**0.000**	0.010	0.101	1.258	0.111
sp97ic	0.915	**0.000**	0.082	2.399	0.687
sp98ar	0.079	0.023	0.046	1.307	0.187
sp98ic	**0.000**	**0.000**	0.018	1.199	0.072
average gap:	**0.654**	3.120	14.807	32.052	20.173

In table 2 values of relative gap in % are provided. The gap is computed as $[(f - f_{best})/|f_{best}|] \times 100$, where f_{best} is the better value of the following two: the best known published value, and the best among the five results we have obtained in our experiments. The table shows that our algorithm outperforms in average all other methods. Even if we do not take into account very hard specific instances `markshare` 1 and 2 and `NSR8K`, the gap values for VNDS, VNSB, LB, CPLEX and RINS are 0.673, 1.424, 2.092, 2.667 and 1.216, respectively, proving that VNDS is still the best choice.

Finally, for all methods we provide computational time spent before the solution process is finished (see table 3). In computing the average time performance, instance `NSR8K` was not taken into account, since the time allowed for solving this model was 15 hours, as opposed to 5 hours for all other models. The results show that LB has the best time performance, whereas VNDS is the second best method regarding the computational time.

Table 3. Running times (in seconds).

Instance	VNDS	VNSB	LB	CPLEX	RINS
mkc	9003	11440	**585**	18000	18000
swath	901	**25**	249	1283	558
danoint	3360	112	**23**	18001	18001
markshare1	12592	8989	**463**	10019	18001
markshare2	13572	14600	7178	**3108**	7294
arki001	4595	6142	10678	339	**27**
seymour	9151	15995	**260**	18001	18001
NSR8K	53651	53610	**37664**	54001	54002
rail507	2150	17015	**463**	662	525
rail2536c	13284	6543	3817	**190**	192
rail2586c	12822	115716	**923**	18049	18001
rail4284c	17875	**7406**	16729	18189	18001
rail4872c	8349	**4108**	10431	18001	18001
glass4	3198	10296	**1535**	3732	4258
van	11535	**5244**	15349.	18001	18959
biella1	**4452**	18057	9029	18001	18001
UMTS	6837	**2332**	10973	18001	18001
net12	**130**	3305	3359	18001	18001
roll3000	2585	**594**	10176	180001	14193
nsrand_ipx	10595	**6677**	16856	13009	11286
a1c1s1	**1438**	6263	15340	18008	18001
a2c1s1	2357	**690**	2102	18007	18002
b1c1s1	**5347**	9722	9016	18000	18001
b2c1s1	**133**	16757	1807	18003	18001
tr12-30	7617	18209	**2918**	7310	4341
sp97ar	16933	**5614**	7067	11842	8498
sp97ic	2014	7844	2478	1245	**735**
sp98ar	7173	6337	1647	1419	**1052**
sp98ic	2724	4993	2231	1278	**1031**
average time	6883	8103	**5846**	11632	11606

6. CONCLUSION

In this paper we propose a new approach for solving binary Mixed integer programming (MIP) problems. Our method combines both hard and soft variable fixing: hard fixing is based on the Variable neighborhood decomposition search (VNDS) framework, whereas soft fixing introduces pseudo-cuts as in local branching Fischetti and Lodi (2003) according to the rules of the variable neighbourhood descent (VND) scheme Hansen et al. (2006). In this way we obtain a two–level VNS scheme known as VNDS heuristic. Moreover, we found a new way to classify instances within a given test bed. We say that a particular instance is either computationally demanding or non-demanding depending on CPU time needed for default CPLEX optimiser to solve it. Our selection of the particular set of parameters is based on this classification.

The VNDS proposed proves to perform well when compared with the state-of-the-art 0-1 MIP solution methods. More precisely, for our quality measures we consider several criteria: average percentage gap, number of times the method managed to get the best objective function value among the five tested approaches and the number of times the method managed to beat the best known published objective. Results in table 2 show that VNDS is indeed the best method regarding the percentage gap, since its gap value is only 0.654, as opposed to gap values 3.120, 14.807, 32.052, and 20.173 of VNSB, LB, CPLEX and RINS methods, respectively. Secondly, tables 2 and 1 show that VNDS heuristic reached the best result among all the five methods in 16 out of 29 cases, whereas RINS heuristic got the best result in 12 cases, VNSB in 10 cases, CPLEX alone in 6 and LB in 2 cases. Thirdly, from table 1 we can see that VNDS managed to improve the best published result in 8 cases (out of 29), VNSB improved the best known result in 3 cases, and local branching and RINS got the new best result for one instance each. Finally, table 3 shows that VNDS is also competitive regarding the computational time, since it is the second best method after the LB heuristic. Possible explanation why VNDS exhibits better performance in average is that most of the instances are large and therefore the decomposition is useful part of the solution process.

REFERENCES

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

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

Glover, F. (2005). Adaptive memory projection methods for integer programming. *Metaheuristic Optimization Via Memory and Evolution: Tabu Search and Scatter Search. Kluwer Academic Publishers.*

Glover, F., Kochenberger, G., and NetLibrary, I. (2003). *Handbook of Metaheuristics.* Kluwer.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3), 449–467.

Hansen, P., Mladenović, N., and Perez-Britos, D. (2001). Variable Neighborhood Decomposition Search. *Journal of Heuristics*, 7(4), 335–350.

Hansen, P., Mladenović, N., and Urošević, D. (2006). Variable neighborhood search and local branching. *Computers and Operations Research*, 33(10), 3034–3045.

ILOG (2006). Cplex 10.1. user's manual.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research*, 24(11), 1097–1100.

Wilbaut, C. and Hanafi, S. (2008). New convergent heuristics for 0–1 mixed integer programming. *European Journal of Operational Research.*