Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 33 (2006) 3034–3045
computers & operations research
www.elsevier.com/locate/cor

Variable neighborhood search and local branching

Pierre Hansen[a,*], Nenad Mladenović[a,b], Dragan Urošević[b]

[a]*GERAD and Ecole des Hautes Etudes Commerciales, 3000 ch. de la Cote-Sainte-Catherine, Montréal, Canada H3T 2A7*
[b]*Mathematical Institute, Serbian Academy of Science, Kneza Mihajla 35, 11000 Belgrade, Yugoslavia*

Available online 24 August 2005

Abstract

In this paper we develop a variable neighborhood search (VNS) heuristic for solving mixed-integer programs (MIPs). It uses CPLEX, the general-purpose MIP solver, as a black-box. Neighborhoods around the incumbent solution are defined by adding constraints to the original problem, as suggested in the recent local branching (LB) method of Fischetti and Lodi (Mathematical Programming Series B 2003;98:23–47). Both LB and VNS use the same tools: CPLEX and the same definition of the neighborhoods around the incumbent. However, our VNS is simpler and more systematic in neighborhood exploration. Consequently, within the same time limit, we were able to improve 14 times the best known solution from the set of 29 hard problem instances used to test LB.

Keywords: Mixed integer programming; Local branching; Variable neighborhood search; CPLEX

1. Introduction

Combinatorial optimization problems usually may be formulated in different ways. Mathematical programming formulations such as mixed integer programs (MIPs) are very popular, since they make possible the use of general-purpose solvers, independently of some problem specific properties. However, in some hard cases, a general-purpose solver might not be an adequate choice. In such cases one tends to use a combinatorial formulation and build a heuristic solution method, thus loosing the advantage of working in a generic and well-explored framework. Therefore, the tradeoff between the use of combinatorial

* Corresponding author. Fax: +1 514 3405665.
E-mail addresses: Pierre.Hansen@gerad.ca (P. Hansen), Nenad.Mladenovic@gerad.ca, nenad@mi.sanu.ac.yu (N. Mladenović), draganu@mi.sanu.ac.yu (D. Urošević).

0305-0548/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.02.033

and mathematical programming formulations is very often a major concern in solving hard real-word optimization problems.

Metaheuristics, that is general frameworks to build heuristics, usually use combinatorial formulations. They are more abstract than e.g. MIPs. Then, problem specific knowledge may be used for various steps of the heuristic. (For discussion of the best-known metaheuristics the reader is referred to the books of surveys edited by Reeves [1] and Glover and Kochenberger [2].) However, it is possible to design heuristics based on metaheuristic rules for solving mathematical programming problems in general, and more particularly, MIPs. For example, a Tabu search (TS) heuristic for solving MIP has been proposed in [3]. The search through the solution space starts from a relaxed solution and then, using pivoting together with TS rules, non-feasibility is decreased step by step.

A new boost in that direction was recently given by Fischetti and Lodi [4]. They suggest a new MIP technique called local branching (LB). It uses CPLEX [5], the general-purpose MIP solver, as a black-box. Neighborhoods around the incumbent solution are defined by adding constraints (or cuts) to the original problem.

Variable neighborhood search (VNS) [6–8] is a recent metaheuristic which exploits systematically the idea of neighborhood change, both in descent to local minima and in escape from the valleys which contain them. In this paper we develop a VNS heuristic for solving MIPs. Our VNS uses the CPLEX solver as a black-box as well. Also, the definition of neighborhoods around the incumbent solution is the same as in LB. The main difference is that we change neighborhoods, (in both a local search and in a diversification or shaking phase) more systematically, following the rules of the general VNS scheme.

This paper is organized as follows. In Section 2 we define MIP and briefly discuss rules of the LB and VNS methods. In Section 3 we present a VNS heuristic for solving MIP. Section 4 consists of computational results. Conclusions are drawn in Section 5.

2. Preliminaries

In this section, after the definition of 0–1 MIPs, we briefly recall the rules of LB and VNS methods.

Let us consider a generic MIP with some 0–1 variables of the form [4]:

$$\text{(P)} \quad \min c^{\mathrm{T}} x \tag{1}$$

$$Ax \geqslant b, \tag{2}$$

$$x_j \in \{0, 1\} \quad \forall j \in \mathscr{B} \neq \emptyset, \tag{3}$$

$$x_j \geqslant 0, \text{ integer} \quad \forall j \in \mathscr{G}, \tag{4}$$

$$x_j \geqslant 0, \quad \forall j \in \mathscr{C}. \tag{5}$$

The variable index set $N = \{1, 2, \ldots, n\}$ is partitioned into the three subsets $(\mathscr{B}, \mathscr{G}, \mathscr{C})$, corresponding to binary, general integer and continuous variables.

2.1. Local branching

LB is a new MIP technique [4] designed, in principle, as an exact method; however, if the total time ($t_{\max}$) allocated to solve a given instance is reached before an optimal solution is found and its optimality proved, LB behaves as a heuristic, i.e., it stops at time $t_{\max}$ with the best solution known as output

(or, possibly, with no feasible solution). The total time limit ($t_{\max}$) is a first parameter. LB uses a general-purpose MIP solver such as CPLEX. However, as direct application would be too time-consuming for very large instances, the solution space is reduced, within a branching scheme, by introducing new linear constraints. These constraints exploit the values of the binary variables in a feasible solution, specifying that at most k (a second parameter, called *size* of the constraint) of them can be complemented. This defines a neighborhood with 2^k solutions in the binary variables only (observe that such neighborhoods are often used in VNS [7,8] or Tabu search [10,11] heuristics, but not implemented through linear constraints). After adding one or several linear constraints the resulting MIP, which has the same structure as the original MIP but a smaller solution space, is solved by CPLEX, exactly or heuristically, within a node time limit, t_{node}, a third parameter.

Branching: The LB algorithm starts with the original formulation, and CPLEX is called to get a feasible solution $\tilde{x}_1$. Then, a new constraint is added to reduce the solution space X, using a distance function $d(x, \tilde{x}_1)$, to the set $X_1 = X \cap N_k(\tilde{x}_1)$, where

$$N_k(\tilde{x}_1) = \{x \mid d(x, \tilde{x}_1) \leqslant k\}. \tag{6}$$

For the 0–1 MIP (1)–(5) given above, $d(.,.)$ represents the Hamming distance and $N_k(\tilde{x}_1)$ may be expressed by the following so-called *local branching constraint*:

$$\Delta(x, \tilde{x}_1) = \sum_{j \in \bar{S}} (1 - x_j) + \sum_{j \in \mathscr{B} \setminus \bar{S}} x_j \leqslant k, \tag{7}$$

where $\bar{S} = \{j \in \mathscr{B} \mid \tilde{x}_1 = 1\}$. If a better solution $\tilde{x}_2$ is found by CPLEX (within a given time limit t_{node}), a new problem is generated as follows. Relation $\leqslant$ in (7) is replaced by $>$, and a new local branching constraint (7) is added, but centered around the new incumbent $\tilde{x}_2$ ($\Delta(x, \tilde{x}_2) \leqslant k$). Thus, a feasible set X_2 for this subproblem is given as

$$X_2 = X \cap (N_k(\tilde{x}_2) \setminus N_k(\tilde{x}_1)) = (X \setminus X_1) \cap N_k(\tilde{x}_2).$$

CPLEX is called again, etc. This *branching* procedure is iterated as long as there are improvements in the objective function values within t_{node} time: ℓ new constraints are added in iteration ℓ and the feasible set is

$$X_\ell = (X \setminus X_1 \setminus X_2 \ldots \setminus X_\ell) \cap N_k(\tilde{x}_\ell).$$

Intensification and diversification: If the solution is not improved within the t_{node} time limit, the neighborhood X_ℓ is further reduced by replacing k in (7) with $k/2$ ($\Delta(x, \tilde{x}_\ell) \leqslant k/2$). This step is called *intensification* step. If CPLEX reports proven infeasibility, or in t_{node} time a feasible solution is not found, then the so-called *diversification* step takes place. Here, the right-hand side value of (7) is increased by $k/2$ ($\Delta(x, \tilde{x}_\ell) \leqslant k + k/2$), a new constraint $\Delta(x, \tilde{x}_\ell) > 1$ added and all other branching constraints deleted. CPLEX is called just to get a new feasible solution for the branching step that follows again. For the diversification, another (fourth) parameter $dv_{\max}$ is used to indicate the maximum number of such steps allowed. If $dv_{\max} = \infty$, LB behaves as a pure heuristic. If $dv_{\max} < \infty$ and if $t_{\max} = \infty$, LB switches to an exact solution method: the remaining time is used for running CPLEX.

Initialization. Select the set of neighborhood structures $\mathcal{N}_k$, for $k = 1, \ldots, k_{max}$, that will be used in the shaking phase, and the set of neighborhood structures N_ℓ for $\ell = 1, \ldots, \ell_{max}$ that will be used in the local search; find an initial solution $\tilde{x}$; choose a stopping condition;

Repeat the following sequence until the stopping condition is met:

(1) Set $k \leftarrow 1$;

(2) *Repeat* the following steps until $k = k_{max}$:

(a) *Shaking*. Generate $\tilde{x}'$ at random from the k^{th} neighborhood $\mathcal{N}_k(\tilde{x})$ of $\tilde{x}$;

(b) *Local search by VND*.

(b1) Set $\ell \leftarrow 1$;

(b2) Repeat the following steps until $\ell = \ell_{max}$;

· *Exploration of neighborhood*. Find the best neighbor $\tilde{x}''$ of $\tilde{x}'$ in $N_\ell(\tilde{x}')$;

· *Move or not*. If $f(\tilde{x}'') < f(\tilde{x}')$ set $\tilde{x}' \leftarrow \tilde{x}''$ and $\ell \leftarrow 1$; otherwise set $\ell \leftarrow \ell + 1$;

(c) *Move or not*. If the local optimum $\tilde{x}''$ is better than the incumbent, move there ($\tilde{x} \leftarrow \tilde{x}''$), and continue the search with $\mathcal{N}_1$ ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$;

Fig. 1. Steps of the general VNS.

2.2. Variable neighborhood search

As mentioned above, variable neighborhood search exploits systematically the idea of neighborhood change, both in descent to local minima and in escape from the valleys which contain them. In order to apply VNS, a neighborhood structure must be defined on the solution space X. To define neighborhoods, we need a distance function $d(\tilde{x}, \tilde{x}')$ between any two solutions $\tilde{x}$ and $\tilde{x}'$ from X. VNS exploits systematically the following observations: (i) a local minimum with respect to one neighborhood structure is not necessarily one for another; (ii) a global minimum is a local minimum with respect to all possible neighborhood structures; (iii) for many problems local minima with respect to one or several neighborhoods are relatively close to each other.

Variable neighborhood descent (VND), a deterministic variant of VNS, is based on observation (i) above, i.e., a local optimum within the neighborhood $N_1(\tilde{x})$ is not necessarily one within neighborhood $N_2(\tilde{x})$. It may thus be advantageous to combine descent heuristics.

Basic VNS combines deterministic and stochastic search. A series of neighborhood structures, which define neighborhoods around any point $\tilde{x} \in X$ of the solution space, are first selected. Then, the local search is used and leads to a local optimum $\tilde{x}$. A point $\tilde{x}'$ is selected at random within the first neighborhood $\mathcal{N}_1(\tilde{x})$ of $\tilde{x}$ and a descent from $\tilde{x}'$ is done with the local search routine. This leads to a new local minimum $\tilde{x}''$. At this point, three outcomes are possible: (i) $\tilde{x}'' = \tilde{x}$, i.e., one is again at the bottom of the same valley; in this case the procedure is iterated using the next neighborhood $\mathcal{N}_k(\tilde{x})$, $k \geqslant 2$; (ii) $\tilde{x}'' \neq \tilde{x}$ but $f(\tilde{x}'') \geqslant f(\tilde{x})$, i.e., another local optimum has been found, which is not better than the previous best solution (or incumbent); in this case too the procedure is iterated using the next neighborhood; (iii) $\tilde{x}'' \neq \tilde{x}$ and $f(\tilde{x}'') < f(\tilde{x})$ i.e., another local optimum, better than the incumbent has been found; in this case the search is re-centered around $\tilde{x}''$ and begins again with the first neighborhood.Should the last neighborhood be reached without a solution better than the incumbent being found, the search begins again at the first neighborhood $\mathcal{N}_1(\tilde{x})$ until a stopping condition, (e.g., a maximum time or maximum number of iterations or maximum number of iterations since the last improvement), is satisfied.

General VNS (*GVNS*): If instead of simple local search in basic VNS one uses VND, one obtains the general VNS scheme that is presented in Fig. 1.

Note that GVNS uses three parameters: $t_{\max}$ if the maximum time allowed is chosen for the stopping condition; $k_{\max}$ the number of neighborhood structures used in the outer loop; $\ell_{\max}$ the number of neighborhoods used in the inner or VND loop. The last two parameters are fixed for a given heuristic, while the first one may be changed according to the problem considered.

3. VNS for MIP

We next develop a VNS for MIP. A main fact that makes our method possible is the same as in LB: a commercial MIP solver (CPLEX) is used for getting an initial solution, and in Shaking (step 2a) as well as for the neighborhood exploration step (step 2b2).

A solution space X is supplied with the Hamming distance $d(.,.)$. Neighborhood structures $N_k, k=1,\ldots$ induced by this metric are defined as in (6). The same neighborhoods are used for both inner and outer loops of GVNS from Fig. 1, i.e., N_k and $\mathcal{N}_k$ are equivalent. As in LB, we use total time $t_{\max}$ as a global stopping rule and the node time t_{node} as a local stopping rule for one call of the mixed integer solver (CPLEX).

We do not fix k, the size of the neighborhood around the incumbent. Instead, in the outer loop, we change it from $k_{\min}$ to $k_{\max}$ with a given step size k_{step}. In VND the neighborhood size ℓ increases from 1 to $\ell_{\max}$. However, we make our heuristic more user-friendly by reducing those three parameters to only one (k_{step}) in the following way: (i) we allow increase of the neighborhood size without limit, i.e., the $k_{\max}$ parameter is not specified. This size is reduced to $k_{\min}$ either when an improved or no feasible solution (in t_{node} time) are found by CPLEX; (ii) we set $k_{\min} = k_{\text{step}}$; (iii) the $\ell_{\max}$ parameter is not specified either. It gets an initial value of 1 when a better solution than the current one is found.

Thus, our VNS uses three parameters: $t_{\max}$ (or `total_time_limit`), t_{node} (or `node_time_limit`) and k_{step}. Note that, by specifying that $k_{\max}$ is equal to the number of binary variables in the instance considered and taking $t_{\max} = \infty$, the proposed GVNS heuristic becomes an exact method. A more detailed description of our heuristic, called VnsBra function is given in Fig. 2.

Beside `total_time_limit`, `node_time_limit` and `k_step` whose meaning have already been described, other variables used in the description of `VnsBra` function from Fig. 2 are:

- UB—input variable for CPLEX solver which represents the current upper bound;
- *first*—logical input variable for CPLEX solver which is `true` if the first solution lower than UB is asked for in the output; if *first* = `false`, CPLEX returns the best solution found so far;
- TL—maximum time allowed for running CPLEX;
- *rhs*—right-hand side of constraint (6); it defines the size of the neighborhood within the inner or VND loop;
- *cont*—logical variable which indicates if the inner loop continues (`true`) or not (`false`);
- *x_opt* and *f_opt*—incumbent solution and corresponding objective function value;
- *x_cur*, *f_cur*, *k_cur*—current solution, objective function value and neighborhood from where VND local search starts (i.e., the distance between the new initial solution and the incumbent is equal to *k_cur*), respectively;
- *x_next* and *f_next*—solution and corresponding objective function value obtained by CPLEX in inner loop.

```
function VnsBra(total_time_limit, node_time_limit, k_step, x_opt);
begin
1    TL := total_time_limit; UB := ∞; first := true;
2    stat := MIPSOLVE(TL, UB, first, x_opt, f_opt);
3    x_cur:=x_opt; f_cur:=f_opt;
4    while (elapsedtime < total_time_limit) do
5       cont := true; rhs := 1; first := false;
6       while (cont or elapsedtime < total_time_limit) do
7          TL = min(node_time_limit, total_time_limit-elapsedtime);
8          add local br. constr. Δ(x, x_cur) ≤ rhs; UB := f_cur;
9          stat := MIPSOLVE(TL, UB, first, x_next, f_next);
10         switch stat do
11            case "opt_sol_found":
12               reverse last local br. const. into Δ(x, x_cur) ≥ rhs + 1;
13               x_cur := x_next; f_cur := f_next; rhs := 1;
14            case "feasible_sol_found":
15               reverse last local br. constr. into Δ(x, x_cur) ≥ 1;
16               x_cur := x_next; f_cur := f_next; rhs := 1;
17            case "proven_infeasible":
18               remove last local br. constr.; rhs := rhs+1;
19            case "no_feasible_sol_found":
20               cont := false
           end
       end
21      if f_cur < f_opt then
22         x_opt := x_cur; f_opt := f_cur; k_cur := k_step;
       else
23         k_cur := k_cur+k_step;
       end
24      remove all added constraints; cont := true;
25      while cont and (elapsedtime < total_time_limit) do
26         add constraints k_cur ≤ Δ(x, x_opt) < k_cur + k_step;
27         TL := total_time_limit-elapsedtime; UB := ∞; first := true;
28         stat := MIPSOLVE(TL, UB, first, x_cur, f_cur);
29         remove last two added constraints; cont =false;
30         if stat = "proven_infeasible" or "no_feasible" then
31            cont :=true; k_cur := k_cur+k_step
           end
       end
    end
end
```

Fig. 2. The overall VNS function VnsBra.

The initialization step is performed in lines 1 and 2. In line 3 the initial solution becomes both the incumbent (*x_opt*) and starting solution for the VND local search (*x_cur*).

The outer loop starts from line 4. In line 5, initial parameters for the inner loop are defined: the first neighborhood is 1 ($rhs := 1$); a best improvement strategy will be applied (i.e., *first*:=`false`). In the computational results section we also tested first improvement VNS. This version is obtained when the command *first* := `false` is removed from line 5.

From lines 7 to 20, statements of VND local search are given. At line 9 a local branching constraint is introduced. The CPLEX solver gives four possible outputs. Only if a feasible solution is not found within

the given time limit, is VND terminated. In the other three cases, the inner VND loop continues: if non feasibility is proven, then the neighborhood is increased by 1 ($rhs := rhs + 1$); if an optimal or feasible solution is found, which means that its value is better that *f_cur*, it becomes the new current solution, and the search starts again from the first neighborhood ($rhs := 1$).

The *move or not* step of GVNS is presented at lines 21–23. If the new local minimum *x_cur* is better than the incumbent, it becomes the new incumbent and the neighborhood for the next shaking step reverts to the smallest size (i.e., `k_min = k_step`). Otherwise, the neighborhood is increased by `k_step`.

In the *Shaking step*, given at lines 24–31, the first feasible solution (i.e., *first* := *true*) is searched for in the disk around the incumbent solution, with radii *k_cur* and *k_cur* + `k_step`. That solution *x_cur* becomes the initial one for the VND local search. If there is no feasible solution in the current disk, then *k_cur* is increased by `k_step`. Note that our `VnsBra` function switches to an exact algorithm if the condition from line 25 ($elapsedtime <$ `total_time_limit`) is replaced by (*k_cur* < `k_max`), with an additional call of CPLEX.

The main idea of both LB and VNS for solving MIP is in fact change of the neighborhood during the search. Therefore, LB could be seen as a specialized variant of VNS (or vice versa): (i) in LB the local search step is performed in a fixed size neighborhood $k > 1$ (a parameter), instead of $k = 1$; (ii) as a consequence of (i), *backward* instead of *forward* VNS (see [9]) is used in the inner loop (i.e., instead of increasing neighborhood by 1 in VND or intensification step, its current size, initially set at k, is reduced by half); (iii) the shaking step of VNS and diversification step of LB differ only in the area from where a random feasible solution is chosen: in LB the area is a disk with radii 1 and $k + dv\,[k/2]$ where dv is the current number of diversifications (see [4] for details), while in our VNS disk is defined by radii k_{cur} and $k_{\text{cur}} + k_{\text{step}}$.

4. Computational results

All programs are written in C++ and experiments conducted on a Pentium 4 computer with 1800 MHz processor and 256 RAM memory. The same data sets as in testing LB (available at http://www.or.deis.unibo.it) are used here: 7 MIPLIB-3.0 instances (set A in Table 1); 1 network design instance (set B); 2 crew scheduling instances (set C); 5 railway crew scheduling instances (set D); 1 nesting instance (set E); 2 telecommunication network design instances (set F); 2 rolling stock and line planning instances (set G); 5 lot-sizing instances (set H); and 4 railway line planning instances (set I).

In developing the final version of some heuristic, usually several possible set of parameters and their values are experimentally examined. Beside exploring the parameter set $\{t_{\max}, t_{\text{node}}, k_{\text{step}}\}$, we also tried with $\{t_{\max}, \ell_{\max}, k_{\max}\}$. There appears to be no set of parameters that is better for all 29 instances. We observe that there is no significant difference in average between two possible sets of parameters. However, for some instances much better results are obtained with one or another parameter set. In order to make comparison with LB easier (LB uses $\{t_{\max}, t_{\text{node}}, k, dv_{\max}\}$), we choose a set $\{t_{\max}, t_{\text{node}}, k_{\text{step}}\}$ with the same values of $t_{\max}$ and t_{node}.

All methods use the same stoping rule $t_{\max}$ equal to 5 h, expect for the largest problem NSR8K, which is run 10 h. Values for *node time limit* are the same as suggested in [4]. Based on data available at http://www.spec.org/cpu2000/results/cfp2000.html, we concluded that our computer is about 30% faster than the Digital Alpha Ultimate workstation 533 MHz that was used in testing LB. In all experiments we set $k_{\text{step}} = 5$.

Table 1
Best known values for the hard MIP instances. Boldfaced are new best known values

Instance	Set	Problem paramet.			Best known values		
name		m	n	$\lvert\mathscr{B}\rvert$	from [4]	available	new
mkc	A	3411	5325	5323	−559.51	−563.85	−563.85
swath	A	884	6805	6724	471.03	471.03	**467.41**
danoit	A	664	521	56	65.67	65.67	65.67
markshare1	A	7	74	60	7.00	7.00	7.00
markshare2	A	6	62	50	14.00	14.00	14.00
arki001	A	1048	1388	415	7,581,034.85	7,580,889.44	7,580,889.44
seymour	A	4944	1372	1372	424.00	423.00	423.00
net12	B	14,021	14,115	1603	255.00	255.00	**214.00**
biella1	C	1203	7328	6110	3,070,810.15	3,070,810.15	3,070,810.15
NSR8K	C	6284	38,356	32,040	21,520,487.01	21,520,487.01	**20,780,430.00**
rail507	D	509	63,019	63,009	175.00	174.00	174.00
rail2536c	D	2539	15,293	15,284	691.00	690.00	690.00
rail2586c	D	2589	13,226	13,215	957.00	947.00	947.00
rail4284c	D	4287	21,714	21,705	1078.00	1065.00	1065.00
rail4872c	D	4875	24,656	24,645	1556.00	1534.00	1534.00
glass4	E	396	322	302	1,587,515,737.50	1,587,515,737.50	**1,400,013,666.50**
UMTS	F	4465	2947	2802	30,160,547.00	30,139,634.00	30,139,634.00
van	F	27,331	12,481	192	5.09	5.09	**4.84**
roll3000	G	2295	1166	246	13,065.00	13,065.00	**12,890.00**
nsrand_ipx	G	735	6621	6620	51,520.00	51,520.00	51,520.00
A1C1S1	H	3312	3648	192	11,834.02	11,834.02	**11,551.19**
A2C1S1	H	3312	3648	192	11,251.10	11,251.10	**10,889.14**
B1C1S1	H	3904	3872	288	25,869.15	25,869.15	**24,566.52**
ABC1S1	H	3904	3872	288	26,297.63	26,297.63	**26,073.78**
tr12-30	H	750	1080	360	130,596.00	130,596.00	130,596.00
sp97ar	I	1761	14,101	14,101	667,735,390.40	667,735,390.40	**666,368,944.96**
sp97ic	I	1033	12,497	12,497	436,984,606.56	436,984,606.56	**429,892,049.60**
sp98ar	I	1435	15,085	15,085	531,942,554.88	531,942,554.88	**530,916,867.40**
sp98ic	I	825	10,894	10,894	449,915,159.36	449,915,159.36	**449,226,843.52**

Table 1 contains the following. In the first column, names of instances are given and in the second the set they belong to. The next three columns give problem characteristics, while columns 6–8 report the best known objective function values: column 6 contains the best known values obtained by three methods and reported in [4] (i.e., `cpx-O`, `cpx-F` and LB). However, those values are not always the best ones known from the literature. For instance, better values could be found for instances `mkc` (−563.85 instead of −559.81), `seymour` (423 instead of 424), `rail507` (174 instead of 175), `rail2586` (1534 instead of 1556), etc. That is why we also give in column 7 the best known solutions available to us. Finally, we report new best known values that include results from this paper. Boldfaced are improved values obtained by our VNS. It can be seen that 14 times (out of 29), the best solution has been improved.

In Table 2 results obtained by first and best improvement VNS (FI and BI for short) are compared with LB. Results for all three methods are for a single run, with parameter settings described above. Columns

Table 2
Comparison on 29 test instances

Instance	Objective value			% GAP			Time (s)		
name	FI	BI	LB	FI	BI	LB	FI	BI	LB
mkc	−558.91	−551.71	−559.51	0.11	1.39	0.00	18,003	14,756	18,000
swath	467.41	467.41	471.03	−0.77	−0.77	0.00	456	7268	4840
danoint	65.67	65.67	65.67	0.00	0.00	0.00	52	89	100
markshare1	7.00	7.00	9.00	0.00	0.00	28.57	3920	16,246	7920
markshare2	14.00	14.00	25.00	0.00	0.00	78.57	3791	7712	10,800
arki001	7,581,138.36	7,580,889.44	7,581,034.85	0.00	0.00	0.00	4428	15,043	3600
seymour	426.00	426.00	424.00	0.47	0.47	0.00	13,372	4508	10,800
net12	214.00	214.00	296.00	−16.08	−16.08	16.08	2207	2210	5040
biella1	3,107,875.16	3,075,887.47	3,070,810.15	1.21	0.17	0.00	4813	225	16,200
NSR8K	20,942,744.00	20,780,430.00	21,520,487.01	−2.68	−3.76	0.00	36,000	36,000	36,000
rail507	176.00	176.00	175.00	0.57	0.57	0.00	12,267	13,987	4320
rail2536c	690.00	693.00	691.00	−0.14	0.29	0.00	290	208	720
rail2586c	955.00	955.00	957.00	−0.21	−0.21	0.00	4849	4870	13,200
rail4284c	1088.00	1088.00	1078.00	0.93	0.93	0.00	1535	1538	16,200
rail4872c	1547.00	1547.00	1556.00	−0.58	−0.58	0.00	15,565	15,563	16,200
glass4	1,460,013,800.00	1,400,013,666.50	1,587,515,737.50	−8.03	−11.81	0.00	9326	8783	16,200
UMTS	30,433,914.00	30,434,962.00	30,198,549.29	0.91	0.91	0.13	2056	1766	16,200
van	4.84	4.84	5.09	−4.91	−4.91	0.00	8389	12,902	7920
roll3000	12,921.00	12,890.00	13,065.00	−1.10	−1.34	0.00	11,269	11,627	6480
nsrand_ipx	52,000.00	52,000.00	51,520.00	0.93	0.93	0.00	16,254	14,276	7920
A1C1S1	11,554.43	11,551.19	11,834.02	−2.36	−2.39	0.00	17,658	18,041	16,200
A2C1S1	10,986.98	10,889.14	11,251.10	−2.35	−3.22	0.00	7278	8418	9360
B1C1S1	24,888.16	24,566.52	25,869.15	−3.79	−5.04	0.00	8193	14,209	12,000
B2C1S1	26,895.72	26,073.78	27,622.24	2.27	−0.85	5.04	8587	16,263	8640
tr12-30	130,942.00	130,979.00	131,029.58	0.26	0.29	0.33	16,674	15,395	5040
sp97ar	666,368,944.96	667,358,765.44	667,735,390.40	−0.20	−0.06	0.00	14,519	18,355	10,080
sp97ic	431,219,871.68	429,892,049.60	436,984,606.56	−1.32	−1.62	0.00	7581	8615	18,000
sp98ar	531,265,245.44	530,916,867.40	531,942,554.88	−0.13	−0.19	0.00	17,941	12,993	9360
sp98ic	450,442,580.16	449,226,843.52	449,915,159.36	0.12	−0.15	0.00	8551	14,794	10,800
Average				−1.27	−1.37	4.44	9376	11,016	10,719

2–4 report values obtained by the three methods: FI, BI and LB. In columns 5–7, the % gap is calculated as

$$\frac{f_{\text{method}} - f_{\text{best}}}{f_{\text{best}}} \times 100,$$

where f_{best} is the best known value from the column 6 (i.e., from the column "from [4]") of Table 1. Thus, a negative gap shows improvement over the three methods from [4]. The last three columns report CPU time (in s) when the best solution was found by each method.

Table 3
Comparison of LB and VNS on the same computer (1.8 MHz Pentium 4) and with the same CPLEX 8.1 version

Instance	Local branching (LB)			Best improvement VNS (BI)		
	Value	% Dev	Time	Value	% Dev	Time
mkc	−560.33	−0.15	7434	−551.71	1.39	14,756
swath	478.03	1.49	5135	467.41	−0.77	7268
danoint	65.67	0.00	14	65.67	0.00	89
markshare1	8.00	14.29	875	7.00	0.00	16,246
markshare2	18.00	28.57	3029	14.00	0.00	7712
arki001	7,580,928.08	0.00	2663	7,580,889.44	0.00	15,043
seymour	425.00	0.24	5620	426.00	0.47	4508
net12	296.00	16.08	873	214.00	−16.08	2210
biella1	3,079,184.35	0.27	17,561	3,055,887.47	0.17	225
NSR8K	21,520,487.00	0.00	89,900	20,780,430.00	−3.76	36,000
rail507	175.00	0.00	5780	176.00	0.57	13,987
rail2536c	691.00	0.00	2721	693.00	0.29	208
rail2586c	956.00	−0.10	16,849	955.00	−0.21	4870
rail4284c	1078.00	0.00	7315	1088.00	0.93	1538
rail4872c	1555.00	−0.06	16,539	1547.00	−0.58	15,563
glass4	1,600,013,433.33	0.79	18,000	1,400,013,666.50	−11.81	8783
UMTS	30,678,670.00	1.72	17,436	30,434,962.00	0.91	1766
van	5.09	0.00	8519	4.84	−4.91	12,902
roll3000	12,934.00	−1.00	16,898	12,890.00	−1.34	11,627
nsrand_ ipx	51,680.00	0.31	15,754	52,000.00	0.93	14,276
A1C1S1	11,759.88	−0.63	4034	11,551.19	−2.39	18,041
A2C1S1	10,951.40	−2.66	3554	10,889.14	−3.22	8418
B1C1S1	25,659.21	−0.81	10,470	24,566.52	−5.04	14,209
B2C1S1	27,480.77	4.50	16,726	26,073.78	−0.85	16,263
tr12-30	130,628.00	0.02	16,245	130,979.00	0.29	15,395
sp97ar	666,368,944.96	−0.20	14,425	667,358,765.44	−0.06	18,355
sp97ic	432,400,213.12	−1.05	5610	429,892,049.60	−1.62	8615
sp98ar	530,480,924.48	−0.27	6119	530,916,867.40	−0.19	12,993
sp98ic	450,442,580.16	0.12	2321	449,226,843.52	−0.15	14,794
Average		2.12			−1.37	

It appears that: (i) negative average gaps for VNS methods indicate that their results are better in average than those obtained by the three methods reported in [4]; (ii) the score between BI-VNS and LB is: 19 times VNS was better, 8 times BI, with 2 ties; (iii) running time for all three methods were similar.

An anonymous referee suggested that further experimental tests be done in order to see if the above mentioned improvements are due, at least in part, to the fact that we used a more recent version of CPLEX (i.e., 8.1) than Fischetti and Lodi (i.e., 7.1). Indeed, it is well known that CPLEX is constantly being improved, embedding many new ideas from the mathematical programming community.

So, we ran again the 29 test problems described above with the same parameter setting, using Fischetti and Lodi's LB code with CPLEX 8.1, on our computer. This ensures differences in machine, compiler and general-purpose MIP solver are eliminated. Results are presented in Table 3. The first column gives

Table 4
Comparison with recent pivot and shift (PS) heuristic

Instance	Objective value				%gap or difference		
name	PS(18′)	LB(5 h)	BI(18′)	BI(5 h)	PS	BI(18′)	BI
sp97ic	442,866,080.00	436,984,608.00	435,761,056.00	429,892,064.00	1.346	−0.280	−1.623
sp98ic	461,222,592.00	449,915,168.00	460,654,656.00	449,226,848.00	2.513	2.387	−0.153
rail507	199.00	175.00	192.00	176.00	24	17	1
rail2536c	767.00	691.00	701.00	690.00	76	10	−1
B2C1S1	27,500.22	27,622.24	33,439.48	26,073.78	−0.442	21.060	−5.606
roll3000	12,936.00	13,065.00	13,201.14	12,890.00	−0.987	1.042	−1.339
glass4	1,900,018,690.00	1,587,515,780.00	1,583,340,670.00	1,400,013,700.00	19.685	−0.263	−11.811

again problem names, the second the obtained value, the third the percentage deviation from the best known value (before results of our paper) and the fourth the time at which this solution was found.

It appears that improved solutions, as compared to those of [4], are obtained in 10 cases and that in average the best solution found are 2.12% above the best known one. This is an improvement upon the 4.44% average obtained by CPLEX 7.1. To compare with results of our VNS, the corresponding results are reproduced on columns 5–7 of Table 3. It appears, that in one case where LB got an improved result (sp98ar) its solution is better than that one obtained by VNS, in one case there is a tie and in the 8 other cases the VNS solutions are better. So, our conclusions remain essentially the same.

Since this paper was submitted for publication, two papers on related topics have appeared. Fischetti et al. [12] begin to explore the "possibility that LB be used to design a genuine MIP metaheuristic framework akin to Tabu Search (TS) or Variable Neighborhood Search (VNS) based on an external MIP solver". They "address MIPs with binary variables and propose a variant of the classical VNS scheme that [they] call *Diversification*, *Refining and Tight-refining* (DRT)". The proposed method is successfully applied to hard facility location problem arising in telecommunication network design. Of particular interest is a method given there to detect automatically the presence of first-level variables the fixation of which produce easier to solve but still hard subproblems in the remaining or second level variables.

Balas et al. [13] develop the pivot and shift (PS) heuristic for MIP, which exploits the well-known pivot-and-complement [14] heuristic for pure 0–1 programs. It is "essentially a rounding procedure amended with two variants of a neighborhood search". This search is akin to LB and done with general-purpose problem solver XPRESS. The combined procedure "finds better solutions faster than the MIP solver alone". Among the 12 difficult problems for which the solution process is reported in detail are 7 considered also in [4]. We reproduce the best solution values found in 20 min of CPU time on a PC with a 1.7 GH Pentium 4 processor for those problems in the second column of Table 4.

It appears that for 2 of them, the solutions were better than those obtained by LB, and worse for the 5 other cases (but bear in mind the differences of computing times and machines). To compare the results of VNS we reproduce values obtained after 18 min and after 5 h of computing time for those problems in columns 4 and 5, respectively, and the corresponding % gaps as difference in the two last columns. While again the results of Balas et al. [13] are better in 2 cases and worse in 5 out of 7 when computing times are comparable, solution of VNS for long computing times are always better.

5. Conclusions

We presented a VNS heuristic for solving Mixed Integer Programming problem. It is similar to the recent Local Branching method [4], which is also based on the idea of changing neighborhood during the search for a better solution. We suggested a more systematic change in both intensification and diversification phases. In that way we improve best known solutions of 14 out of 29 hard test instances from the literature. Results obtained are also better than those of the pivot and shift heuristic [13] for the 7 instances among those 29 ones considered there.

It thus appears that combining heuristic frameworks with the use of general-purpose problem solvers already gives significantly improvements in the resolution of large general MIPs as well as of specific subclasses of them. It is a topic well worth further study.

References

[1] Reeves C, editor, Modern heuristics. Boston, Dordrecht, London: Kluwer Academic Publishers, 1993.
[2] Glover F, Kochenberger G, editors. Handbook of Metaheuristics. Boston, Dordrecht, London: Kluwer Academic Press, 2003.
[3] Lokkentagen A, Glover F. Solving 0–1 mixed integer programming problems using tabu search. European Journal of Operational Research 1998;106:624–58.
[4] Fischetti M, Lodi A. Local branching. Mathematical Programming Series B 2003;98:23–47.
[5] CPLEX: ILOG CPLEX 7.1 User's manual and reference manual, 2001.
[6] Mladenović N, Hansen P. Variable neighborhood search. Computers and Operations Research 1997;24:1097–100.
[7] Hansen P, Mladenović N. Variable neighborhood search: principles and applications. European Journal of Operational Research 2001;130:449–67.
[8] Hansen P, Mladenović N. Variable neighborhood search. In: Glover F, Kochenberger G, editors. Handbook of metaheuristics. Boston, Dordrecht, London: Kluwer Academic Publisher; 2003. p. 145–84.
[9] Hansen P, Mladenović N. Developments of variable neighborhood search. In: Ribeiro C, Hansen P, editors. Essays and surveys in metaheuristics. Boston, Dordrecht, London: Kluwer Academic Publishers; 2001. p. 415–40.
[10] Glover F, Laguna M. General purpose heuristics for integer programming: Part I. Journal of Heuristics 1997;2:343–58.
[11] Glover F, Laguna M. General purpose heuristics for integer programming: Part II. Journal of Heuristics 1997;3:161–79.
[12] Fischetti M, Polo C, Scantamburlo M. A Local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. Networks 2004;61–72.
[13] Balas E, Schmieta S, Wallace C. Pivot and shift—a mixed integer programming heuristic. Discrete Optimization 2004;1:3–12.
[14] Balas E, Martin CH. Pivot and complement—a heuristic for 0–1 programming. Management Science 1980;26:86–96.