Available online at www.sciencedirect.com
SCIENCE DIRECT®
Discrete Applied Mathematics 145 (2004) 117–125
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Variable neighborhood search for the maximum clique

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

[a] GERAD and HEC Montréal 3000 ch. de la Côte-Sainte-Catherine, Montréal, Canada H3T 1V6
[b] Mathematical Institute, Serbian Academy of Science, Kneza Mihajla 35, Belgrade 11000, Serbia, Montenegro, Yugoslavia

Received 27 February 2001; received in revised form 17 June 2003; accepted 19 September 2003

Abstract

Maximum clique is one of the most studied NP-hard optimization problem on graphs because of its simplicity and its numerous applications. A basic variable neighborhood search heuristic for maximum clique that combines greedy with the simplicial vertex test in its descent step is proposed and tested on standard test problems from the literature. Despite its simplicity, the proposed heuristic outperforms most of the well-known approximate solution methods. Moreover, it gives solution of equal quality to those of the state-of-the-art heuristic of Battiti and Protasi in half the time.

Keywords: Combinatorial optimization; Graphs; Maximum clique; Heuristics; Variable neighborhood search

1. Introduction

Let $G=(V,E)$ denote a graph with vertex set $V=\{v_1,v_2,\ldots,v_n\}$ and edge set $E=\{e_1,e_2,\ldots,e_m\}$. A set $C\subseteq V$ of vertices is a *clique* if any two of them are adjacent. A set $S\subseteq V$ of vertices is a *stable* (or independent) set if any two of them are not adjacent. A set $T\subseteq V$ of vertices is a *transversal* (or vertex cover) if any edge of E contains at least one vertex of T. Clearly, a clique of G is a stable set of the *complementary graph* $\bar{G}=(V,\bar{E})$ of G, in which a pair of vertices is joined by an edge if and only if it is not so in G. Moreover, any minimal transversal T of G is the vertex complement of a *maximal stable set* S of G (to which no vertex can be added without losing stability) i.e., a vertex belongs to T if and only if it does not belong to S. Obviously, a maximal stable set need not necessarily be maximum.

A clique C (or a stable set S) is *maximum* if it has the largest possible number of vertices. The *clique number* $\omega(G)$ of G (respectively, the *stability number* $\alpha(G)$ of G) is equal to the cardinality of a maximum clique (respectively, a maximum stable set). So finding a maximum clique is tantamount to finding a maximum stable set or a minimum transversal in the complementary graph.

Finding maximum cliques or stable sets, or minimum transversals are classical problems in applied graph theory. They have many applications in various fields, e.g., experimental design; information retrieval systems; pattern recognition; coding theory; signal transmission analysis; computer vision; sociological structures; economy; forest planning, and so on (see e.g. the surveys [5,7,26] for references).

The maximum clique problem (MCP) has many mathematical programming formulations, including linear integer and nonlinear programming ones (see e.g. [7]). The simplest integer programming expression is the so-called *edge formulation*:

$$\max_x \sum_{i=1}^{n} x_i, \tag{1}$$

$$\text{s.t.} \quad x_i + x_j \leqslant 1, \quad \forall (i,j)\in \bar{E}$$

$$x_i \in \{0,1\}, \quad i=1,\ldots,n, \tag{2}$$

E-mail address: gerad@crt.umontreal.ca (P. Hansen).

0166-218X/$ - see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.dam.2003.09.012

where $x_i = 1$ if vertex v_i belongs to a clique and 0 otherwise. Among several formulations which use continuous variables, we only mention the earliest one, that has been suggested by Motzkin and Strauss [23] in 1965 (see also [7] for a discussion of this and similar formulations): solve

$$\max_x g(x) = x^{\mathrm{T}} A x, \tag{3}$$

where A is the adjacency matrix of G. The clique number can then be found as $\omega(G) = 1/(1 - g(x^*))$, where x^* is a global maximizer of g on the standard simplex in $\mathbb{R}^n$, from which a maximum clique may be deduced.

The MCP is NP-hard [11] and numerous heuristic or exact algorithms have been proposed to solve it. The exact algorithms are usually enumerative in nature. Efficient exact algorithms have been proposed by Balas and Yu [5], Pardalos and Rodgers [25], Friden et al. [10], Carraghan and Pardalos [8], Babel and Tinhofer [2], Hansen and Mladenović [16], Tomita et al. [28] and others. An empirical comparison of exact methods suggested in [5,8,10,16] has been performed in [17]. Further empirical results are given in [21].

The list of heuristic methods is even larger. The Tabu search (TS) approach is applied in [6,10,12,14,15,27], Genetic algorithms (GAs) are proposed in [1,3], GRASP rules are applied in [9], a continuous based heuristic (CBH) is suggested in [13], etc. As an illustration of the large number of heuristics for MCP, let us mention that fifteen new heuristics were presented in 1993 at the DIMACS workshop on "Cliques, Coloring and Satisfiability" [21] alone. Further references to applications of metaheuristics to maximum clique or equivalent problems may be found in the detailed bibliography of Osman and Laporte [24].

The very recent *reactive local search* [6] (RLS) can be considered as the state-of-the-art heuristic, since it provides the best results to date (in terms of objective function values) on standard DIMACS test instances [21]. This heuristic is based on local search and a feedback scheme to determine the amount of diversification. The reaction acts on the time during which selected moves in the neighborhood will be prohibited. Observe that contrary to other TS heuristics for MCP (such as [27]) RLS uses, as does VNS, a metric function to measure distance between any two solutions.

The paper is organized as follows. In the next section we first give the rules of the basic variable neighborhood search (VNS); in Section 3 we describe a local search procedure we use within VNS in solving the MCP. In Section 4, a comparison with other heuristics on DIMACS Benchmark Instances [21] is presented.

2. Basic principles of VNS

The basic idea of the recent metaheuristic called VNS is to change systematically neighborhoods in the search for a better solution (see [22], and for recent surveys on VNS [18] or [19]). To construct different neighborhood structures and to perform a systematic search, one needs to have a way for finding the distance between any two solutions, i.e., one needs to supply the solution space with some metrics (or quasi-metrics) and then induce neighborhoods from them.

Let us denote with $\mathcal{N}_k$, $(k = 1, \ldots, k_{\max})$, a finite set of pre-selected neighborhood structures, and with $\mathcal{N}_k(X)$ the set of solutions in the k^{th} neighborhood of X. (Most local search heuristics use only one neighborhood structure, i.e., $k_{\max} = 1$.) Steps of the basic VNS are presented in Fig. 1.

The stopping condition may be, e.g. maximum CPU time allowed, maximum number of iterations, or maximum number of iterations between two improvements. Often successive neighborhoods $\mathcal{N}_k$ will be nested. Observe that point

Initialization. Select the set of neighborhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{max}$, that will be used in the search; find an initial solution X; choose a stopping condition;

Repeat the following sequence until the stopping condition is met:
(1) Set $k \leftarrow 1$;
(2) Until $k = k_{max}$, repeat the following steps:
(a) Shaking. Generate a point X' at random from the k^{th} neighborhood of X $(X' \in \mathcal{N}_k(X))$;
(b) Local search. Apply some local search method with X' as initial solution; denote with X'' the so obtained local optimum;
(c) Move or not. If this local optimum is better than the incumbent, move there $(X \leftarrow X'')$, and continue the search with $\mathcal{N}_1$ $(k \leftarrow 1)$; otherwise, set $k \leftarrow k + 1$;

Fig. 1. Steps of the basic VNS.

Initialization. Select the set of neighborhood structures $\mathcal{N}'_k$, $k = 1, \ldots, k'_{max}$, that will be used in the descent; find an initial solution X;

Repeat the following until no improvement is obtained:
(1) Set $k \leftarrow 1$;
(2) Until $k = k'_{max}$, repeat the following steps:
(a) Exploration of neighborhood. Find the best neighbor X' of X ($X' \in \mathcal{N}'_k(X)$);
(b) Move or not. If the solution thus obtained X' is better than X, set $X \leftarrow X'$ and $k' \leftarrow 1$; otherwise, set $k \leftarrow k + 1$.

Fig. 2. Steps of the basic VND.

X' is generated at random in step 2a in order to avoid cycling, which might occur if any deterministic rule was used.

As a local optimum within some neighborhood is not necessarily one within another, change of neighborhoods can also be performed during the local search phase. This local search is then called variable neighborhood descent (VND) and its steps are presented in Fig. 2. Usually VND is used as a local search routine within VNS.

Further variants and extensions of the basic VNS are presented in [18,19].

3. VNS for the MCP

In this section, we first recall some definitions and then explain the VND and shaking steps used within the basic VNS given in Fig. 1.

3.1. Preliminaries

Let $C \subset V$ be any clique of G and let $S = C$ and $T = V \setminus S$ be the corresponding stable set and transversal in $\bar{G}$, respectively. Throughout the text, we consider interchangeably a stable set S in $\bar{G}$ or corresponding clique C in G. The solution space $\mathcal{S}$ can then be represented as the set of all complete subgraphs of G or, which is equivalent, of all stable sets or transversals of $\bar{G}$. In our implementation, the minimum transversal or minimum vertex cover problem is considered. Therefore, the solution space $\mathcal{S}$ is represented by the set of all transversals T in $\bar{G}$. We need to find a transversal T_{opt} which has a minimum cardinality, and covers all vertices of S (i.e., any vertex of S must be adjacent to at least one vertex of T in $\bar{G}$).

Let us now introduce the distance $\rho(T, T')$ between any two solutions T and T' as the symmetric difference between these sets. Therefore, we say that distance between the two solutions from $\mathcal{S}$ is equal to k if and only if the symmetric difference between their vertex sets has cardinality k, and the neighborhood $\mathcal{N}_k(T)$ consists of all solutions at distance k from T:

$$T' \in \mathcal{N}_k(T) \rho(T, T') = k. \tag{4}$$

It is clear that this function is a metric, and thus $\mathcal{S}$ is a metric space.

Note that $\mathcal{N}_1(T)$ is the neighborhood usually used in local and TS methods for finding the maximum clique. It corresponds to addition or deletion of vertices one at the time in the current solution: $\mathcal{N}_1(T) = \mathcal{N}_1^+(T) \cup \mathcal{N}_1^-(T)$. In general

$$\mathcal{N}_k(T) \supset \mathcal{N}_k^+(T) \cup \mathcal{N}_k^-(T) \tag{5}$$

for $k \geqslant 2$, and these last sets correspond to addition or deletion of vertices k at a time.

Note also that $\rho(\cdot, \cdot)$ can be viewed as the Hamming distance, if the solution T is represented by a 0-1 array of length n. In [6], the same distance function is used in the diversification step.

A vertex v of an arbitrary graph G is called *simplicial* if all vertices adjacent to v are pairwise adjacent, i.e., if the subgraph induced by v and its neighbors is a clique. The *size* of the simplicial vertex is ℓ, if the clique number of such a clique is equal to ℓ. In [16] it has been observed that any vertex which is simplicial in the complementary graph $\bar{G} = (V, \bar{E})$ of a given graph G belongs to at least one maximum clique of G. This property is exploited in two exact

Simplicial vertex test. Call the Simplicial vertex test (SVT) procedure (see [10]); denote by $V_t, \bar{E}_t, T_t, C_t$ the inputs and outputs;

Solution test. If $V_t = \emptyset$, go to *interchange* step (T_t is a minimal transversal and C_t a maximal clique);

Add step - Greedy selection rule. Select a vertex v to be added to the current clique C_t or to the current transversal T_t; in case of tie, apply some *tie-breaking* rule (see below); if v is added to C_t, add all its adjacent vertices to T_t;

Subproblem reduction. Delete all edges from $\bar{E}_t$ incident to those new vertices in T_t; return to *Simplicial vertex test* step.

Interchange step - Plateau search. Find set K_t by (6). If $K_t \neq \emptyset$ then interchange in turn each (*one missing*) vertex $v'' \in K_t$ with corresponding vertex v' in the clique, i.e. $(v'', v') \in \overline{E}_t$ and check if an improving move is possible. In other words, check if there is another vertex v''' from K_t again with corresponding vertex v' in C_t and such that $(v'', v''') \notin \bar{E}_t$. If so, a better solution is found: $C_t := C_t \cup \{v'', v'''\} \setminus \{v'\}$; $T_t := T_t \cup \{v'\} \setminus \{v'', v'''\}$;

Termination. If $K_t = \emptyset$ or there is no improvement in the previous step, Stop (a local optimum with respect to both the *add* and *1-Interchange* neighborhoods is found). Otherwise, return to the *interchange* step.

Fig. 3. Steps of the VND local search.

algorithms, where the maximum size $\ell_{\max}$ considered for ℓ was set to 2 or 3. Here we use a simplicial vertex test as part of our VND heuristic.

In solving MCP, the set of vertices at iteration t is usually divided into three disjoint subsets: C_t—vertices in the current clique; T_t—vertices that belong to the current transversal and V_t—vertices not yet distributed to the other two sets. The current subproblem is defined on the subgraph $\bar{G}_t = \bar{G}(V_t, \bar{E}_t)$, where $\bar{E}_t$ is the subset of edges of $\bar{E}$ with both endpoints belonging to V_t. In the algorithm a minimal transversal will be obtained when $V_t = \emptyset$. Let us define an array $\Phi(v)$, which reports for any v not in the clique C_t ($v \notin C_t$) the number of its adjacent vertices that belong to the clique (with respect to $\bar{G}_t$). Let us also define sets A_t and K_t as

$$A_t = \{v \notin C_t | \Phi(v) = 0\}; \quad K_t = \{v \notin C_t | \Phi(v) = 1\}. \tag{6}$$

Note that in [6] sets A_t and K_t are called *possible add* and *one missing*, respectively.

We now give a short description of our VNS heuristic. An initial solution is obtained by a VND heuristic (see Figs. 2 and 3). The same VND heuristic is used later on as a local search within the basic VNS (Step 2b from Fig. 1). (Note that for solving MCP there is no need to apply a constructive heuristic at the same time as a local search one, since a clique $C = \emptyset$ is already a feasible solution). The set of neighborhood structures is defined by (4). Using them, three types of moves are considered: (i) *drop*; (ii) *add* and (iii) *interchange* (which occurs when drop and add moves are performed at the same time). Our VND uses *add* moves in its descent phase, and *interchange* moves if the best solution in the neighborhood has the same cardinality as T (or C), i.e., in a 'plateau' phase. The *drop* moves are used in the *Shaking* step of VNS (Step 2a from Fig. 1).

3.2. VND for the MCP

There are several questions that should be addressed in designing any local search heuristic for solving the MCP: (i) *greedy selection rule*—how to choose the next vertex (or vertices) to be added to C_t (or T_t) in the subproblem defined by graph $\bar{G}(V_t, \bar{E}_t)$? (ii) *tie-breaking rule*—if there are more than one vertex that satisfy the selection rule above, how should the choice be made? (iii) *plateau search*—should the search be continued when the best solution in the neighborhood has the same cardinality as the incumbent (i.e., if $|C_{t+1}| = |C_t|$ and $V_{t+1} = V_t = \emptyset$) and how? We now specify the rules of our VND (at a current iteration t), and then give answers to questions (i)–(iii).

Simplicial vertex test: Detailed pseudo-code for SVT is given in [16]. Here we describe briefly the main idea. In the initial graph $\bar{G}_t$ we search first for the isolated vertices (simplicial vertices with size zero, $\ell = 1$) and add them to the

clique, if any; then we check if there are leaves ($\ell = 2$). If so, one among them (say v_1) is chosen (at random or by some other *tie-breaking rule*) and added to the clique, its adjacent vertex v_1' added to T_t, and all edges connected to v_1' deleted from the graph. In the subgraph so obtained (i.e., without vertices that belong to either clique or transversal) we examine again if there are isolated vertices ($\ell = 1$). Otherwise, we look for simplicial vertices with size two ($\ell \leftarrow \ell + 1$). If there are such vertices we choose one at random (v_2), place it in the clique C_t and two of its adjacent vertices v_2' and v_2'' are added to T_t. We again delete all edges incident to v_2' and v_2'' to get a new subgraph.

The SVT procedure stops when the minimal transversal (the maximal clique) is found or when there is no simplicial vertex of size $\ell_{\max}$ in the current subgraph. In the former case a local minimum is found and the local search procedure stops; in the later case, we proceed with a different local search (a new vertex is selected by some *greedy selection rule*).

Greedy selection rules: The efficiency of a local search heuristic for MCP will largely depend on the choice of the vertex selected as a new member of C_t or T_t. We list here several rules. Experience suggests that selecting the right one is a main tool to solve 'tricky' MCP instances:

(i) Add *min degree* ($\bar{G}_t$) to C_t . The usual way to select a vertex v to be added to C_t is to choose that one with the maximum degree in G_t (see for example [6,12,20]) or, which is equivalent, to add to C_t a minimum degree vertex in $\bar{G}_t$. As a consequence, all neighbors of v in $\bar{G}_t$ are added to T_t;
(ii) Add *max degree* ($\bar{G}_t$) to T_t. This move (which appears to be new) can be considered as a "smaller size" move than the previous one since no vertex is added to C_t after it takes place; it could be expected that this rule works better for dense graphs G (i.e., for sparse $\bar{G}$);
(iii) Combination of (i) and (ii): (a) with a given probability p (a parameter); i.e., if a random number uniformly distributed in the interval (0,1) is less then p, vertex v is selected by rule (i), otherwise by rule (ii); (b) combination of (i) and (ii) as a function of density of the current $\bar{G}_t$; for example, if $dens(\bar{G}) < p$, apply (i), otherwise (ii);
(iv) Add a randomly chosen vertex from V_t to C_t.

Tie-breaking rules: These rules are applied if there are more than one vertex that satisfy certain of conditions listed above. They can be used in the following situations: (a) in SVT, when there are more than one simplicial vertex with the same degree; (b) in Greedy selection described above:

(i) Choose vertex v at random. A simplest way is to choose one vertex v among those with the same minimum degree at random;
(ii) Choose v from T_t with $\min \Phi(v)$; if there are more than one such vertices, choose one at random.

Plate search: From Fig. 3 it is clear that we use another neighborhood structure (i.e., interchange) to deal with *plateau* search. An efficient way to find the best solution in the interchange neighborhood is to add to T_t all vertices from C_t that correspond to vertices in K_t, and then return to the *Simplicial vertex test*. Therefore, *plateau search* can also be seen as intensified shaking (in $\mathcal{N}_1$ neighborhood), since shaking is defined by dropping vertices from the current clique C_t.

3.3. Shaking

Jumping from T_t to some $T_t' \in \mathcal{N}_k(T_t)$ is defined by dropping k vertices from the current clique C_t; in that way a new subproblem is defined on the subgraph $\bar{G}_{t+1}$ of $\bar{G}$, where set of vertices are represented by vertices removed from C_t and vertices from T_t. For each k we could perform two types of shaking: (i) Take k vertices from C_t at random and add them to V_t; (ii) Rank array $\Phi(v)$, $\forall v \in T_t$ in nondecreasing order $\Phi_1 \leqslant \Phi_2 \leqslant \cdots \leqslant \Phi_{|T_t|}$. Then $k = \Phi_j$, $j = 1, \ldots, |T_t|$; note that the number k of vertices to be dropped is not known in advance. In the experiments described below, the first rule was applied.

4. Computational results

The VNS heuristic described above, as well as some variants, have been coded in C++ and run on Pentium II, 450 MHz, 64 Mb RAM station. The 37 hard DIMACS test instances have been chosen to carry out computational experience and comparison with other recent heuristics from the literature.

In all results reported below, the single parameter $k_{\max}$ is equal to the current clique number, i.e., $|C_t|$. (This entails a slight but obvious extension of the basic scheme of Fig. 1.) We always use 10 different neighborhoods in the search (or $|C_t|$ neighborhoods if $|C_t| < 10$), thus, the step between two successive neighborhoods is defined by $[|C|/10]$, where $[a]$ denotes the smallest integer greater than or equal to a.

Table 1
Experiments with different variants of VND

Pr. No	Pr. name	H_0		H_1		H_2		H_3		H_4		H_5		BR
		$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$
1	C125.9	34	0.02	34	0.03	34	0.02	34	0.02	34	0.02	34	0.03	34*
2	C250.9	44	0.22	44	0.44	44	0.34	44	0.23	44	0.26	44	0.19	44*
3	C500.9	57	3.62	56.7	26.81	57	22.28	57	5.22	57	5.17	57	4.73	57
4	C1000.9	68	70.96	66.4	46.91	67.4	264.51	67.8	154.12	68	84.56	68	50.84	67
5	C2000.9	77.2	311.54	74.7	293.79	76.1	543.71	76.3	306.13	77.6	314.21	77.2	158.40	75
29	p_hat300-1	8	0.02	8	0.30	8	0.03	8	0.02	8	0.02	8	0.02	8*
30	p_hat300-2	25	0.01	25	0.03	25	0.01	25	0.01	25	0.01	25	0.01	25*
31	p_hat300-3	36	0.07	36	0.41	36	0.07	36	0.08	36	0.11	36	0.08	36*
32	p_hat700-1	11	0.52	11	12.77	11	1.82	11	0.69	11	0.56	11	0.83	11*
33	p_hat700-2	44	0.05	44	0.40	44	0.06	44	0.06	44	0.06	44	0.06	44*
34	p_hat700-3	62	0.06	62	0.47	62	0.06	62	0.06	62	0.06	62	0.06	62
35	p_hat1500-1	12	415.68	12	601.45	12	431.10	12	511.89	12	519.70	12	545.63	12*
36	p_hat1500-2	65	0.28	65	2.74	65	0.28	65	0.28	65	0.30	65	0.28	65
37	p_hat1500-3	94	0.58	94	6.70	94	5.87	94	1.85	94	1.85	94	1.10	94
Average		45.5	68.02	45.2	70.95	45.4	90.72	45.4	70.05	45.6	66.21	45.5	54.44	45.3

Table 2
Experiments with different variants of VND

Pr. No	Pr. name	H_0		H_1		H_2		H_3		H_4		H_5	
		$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time
1	j12-3-5	79.00	5.36	76.90	5.33	79.00	6.41	77.90	2.82	78.60	4.34	79.10	8.26
2	j12-4-5	78.90	5.89	76.90	5.33	78.70	6.12	77.90	2.83	78.90	5.66	78.90	5.66
3	j12-4-6	132.00	0.04	132.00	0.04	132.00	0.04	132.00	0.04	132.00	0.04	132.00	0.04
4	j12-5-6	22.00	0.02	22.00	0.01	22.00	0.01	22.00	0.02	22.00	0.02	22.00	0.01
5	j13-4-5	120.80	6.67	118.30	6.57	120.00	10.10	119.30	11.23	121.70	11.52	120.80	10.30
6	j13-5-6	25.00	5.71	24.40	3.49	24.80	5.09	24.00	0.20	25.20	4.24	25.40	5.32
7	j14-3-4	89.40	7.00	86.90	6.21	85.90	8.80	88.30	7.17	89.40	9.55	88.90	8.22
8	j14-4-5	161.60	8.84	159.00	9.96	160.60	4.24	160.10	12.48	161.30	6.77	160.80	7.79
9	j15-3-4	101.40	4.49	100.40	6.22	100.90	5.74	100.60	6.89	101.20	7.55	103.00	8.76
10	j16-3-4	140.00	3.78	140.00	2.37	137.60	2.43	140.00	1.08	140.00	1.69	140.00	3.63
11	j17-2-3	680.00	0.03	680.00	0.03	680.00	0.03	680.00	0.03	680.00	0.03	680.00	0.03
12	j17-3-4	152.20	7.60	150.30	7.68	152.20	5.26	151.40	5.03	152.50	6.02	152.40	7.51
Average		148.53	4.62	147.26	4.44	147.81	4.52	147.79	4.15	148.57	4.79	148.61	5.46

In Tables 1 and 2 we present result for several variants of VND, as described in the previous section. Each variant is determined by four factors: (1) whether SVT is used or not; (2) what greedy selection rule is used; (3) what tie-breaking rule is implemented; (4) whether *interchange* local search is used or not. The basic version (denoted with H_0 in Tables 1 and 2 and used also in Tables 3 and 4 below), has SVT and *interchange* local search, uses *min degree* ($\bar{G}$) selection rule and random tie-breaking. In the description of the other search strategies H_1–H_5 which follows we only indicate in what they differ from the basic version H_0:

H_1: H_0 without *interchange* local search;

H_2: H_0 with *max degree* ($\bar{G}$) selection rule (instead of *min degree* ($\bar{G}$));

H_3: H_0 without SVT;

H_4: H_0 with *greedy selection rule* (iii a) and $p = 0.5$;

H_5: H_0 with $\min \Phi(v)$ tie-breaking (instead of random);

For each variant, cardinality of the largest clique found is given, together with the time to find it. The last column gives the best value published at the time of writing for each problem (i.e. not including best values found by the heuristic of Battiti and Protasi, given in Table 4).

Table 3
GA1, GA2; CBH; VNS

Pr. No	Problem name	GA1		GA2		CBH		VNS		BR
		$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$
10	MANN_a27	126	2.55	126	8	121	4.06	126	0.07	126*
11	MANN_a45	343	34.38	343.59 (343–344)	383	343	69.95	344.5 (344–345)	20.79	345*
13	brock200_2	11	0.30	9.92 (9–11)	17	12	1.80	11.3 (11–12)	1.05	12*
14	brock200_4	16	0.25	16.04 (16–17)	24	16	0.42	16.9 (16–17)	6.80	17*
15	brock400_2	24	2.57	24.00 (23–25)	79	24	7.18	27.4 (25–29)	57.19	29*
16	brock400_4	24	1.59	24.13 (23–25)	12	24	3.79	33	36.90	33*
17	brock800_2	19	4.77	18.47 (18–20)	47	19	21.42	21	11.78	21
18	brock800_4	19	10.28	19.05 (19–20)	550	19	21.63	21	43.28	21
24	hamming8-4	16	0.33	16	1	16	0.55	16	0.01	16*
25	hamming10-4	33	15.68	39.18 (38–40)	46	35	9.55	40	0.26	40
26	keller4	11	0.18	11	1	10	0.18	11	0.01	11*
27	keller5	25	11.96	25.76 (25–27)	54	21	9.68	27	0.38	27
29	p_hat300-1	8	0.63	7.69 (7–8)	2	8	1.65	8	0.02	8*
30	p_hat300-2	25	0.93	25	1	25	1.41	25	0.01	25*
31	p_hat300-3	36	0.37	35.85 (34–36)	1	36	2.67	36	0.07	36*
32	p_hat700-1	9	4.74	9.45 (9–11)	180	11	21.64	11	0.52	11*
33	p_hat700-2	44	11.34	44	3	44	14.74	44	0.05	44*
34	p_hat700-3	62	4.32	62	3	60	18.78	62	0.06	62
35	p_hat1500-1	10	12.53	10.10 (9–11)	107	11	162.19	12	415.68	12*
36	p_hat1500-2	59	56.00	65	10	63	76.30	65	0.28	65
37	p_hat1500-3	92	56.24	93.44 (93–94)	860	94	286.40	94	0.58	94
Average		48.2	11.04	49.13 (48.59–19.77)	563.05	48.2	35.05	50.1 (49.9–50.2)	28.39	50.2

Three sets of test instances are used to compare H_0–H_5. Average results for ten runs obtained on so-called *C* and *p_hat* graphs (Table 1), and on random *johnson* graphs (Table 2) are reported. Test problems from Table 1 belong to DIMACS hard test instances and the first two columns give the problem number and name respectively. The values of parameters m_1, m_2 and m_3 used in generating *johnson* graphs are given in the second column, and the resulting number of nodes n in the third column of Table 2.

The general conclusion arising from Table 1 is that efficiency of the VNS heuristic depends, to a certain extent regarding values found and considerably regarding computation times, on which components are chosen for its VND local search. Moreover, there is no version that dominates others, in terms of both values found and computing times, for all test instances. More precise observations are as follows: (i) *Interchange* local search is important; average results obtained by H_1 were the worst on all three test instances; for example, compare H_0 and H_1 for *p_hat* instances, where H_0 is always faster, sometimes more than 10 times, i.e. in problems 29 and 37; (ii) *min degree* selection rule is better than *max degree* in average when used alone (compare columns H_0 and H_2); however, their combination (in column H_4) seems to be the best choice; (iii) in some instances use of SVT is not necessary (see H_3 for problems # 1, 29, 30, 31, 33, 34 and 36 where same solutions as those provided by H_0 are obtained in similar computing time); however, in all other cases, heuristic H_3 is dominated by H_0; (iv) the intensified *tie-breaking* rule does not seem to be not necessary in these test instances, but the results obtained are not very different from those obtained with the basic version.

The hard DIMACS test instances have been chosen to carry out a computational comparison between our VNS and the following recent heuristics:

(i) GA by Aggarwal et al. [1] (GA1 for short) and by Balas and Niehaus [4] (GA2 for short);
(ii) CBH by Gibbons et al. [13] (CBH for short);
(iii) TS by Soriano and Gendreau [27];
(iv) RLS by Battiti and Protasi [6].

Note that several other heuristics of the DIMACS challenge are not included in the comparison as their performance is not as the same level as those listed above.

Since all results could not fit in one table, they are divided in two: in Table 3 VNS is compared with GA and CBH, while in Table 4 results obtained by TS, RLS and VNS are reported. Table 3 contains only 21 problem instances, because only for them had GA results been reported for both heuristics, while in Table 4 heuristics are compared on all 37 hard

Table 4
Comparison on 37 hard DIMACS challenge instances: TS; RLS; VNS

Pr. No	Problem name	TS		RLS		VNS		BR
		$\|C\|$	Time	$\|C\|$	Time	$\|C\|$	Time	$\|C\|$
1	C125.9	34	0.13	34	0.01	34	0.02	34*
2	C250.9	43.2 (43–44)	5.32	44	0.06	44	0.22	44*
3	C500.9	55.8 (55–57)	10.25	57	5.40	57	3.62	57
4	C1000.9	65.0 (63–66)	55.73	68	80.40	68	70.96	67
5	C2000.9	72.8 (72–74)	125.18	77.6 (77–78)	1589.09	77.2 (76–78)	311.54	75
6	DSJC500.5	13	4.77	13	0.37	13	0.89	14*
7	DSJC1000.5	14.2 (14–15)	7.25	15	12.45	15	14.58	15
8	C2000.5	15	10.73	16	17.26	16	19.20	16
9	C4000.5	16.6 (16–17)	32.46	18	4213.36	18	4256.80	18
10	MANN_a27	125	0.43	126	6.01	126	0.07	126*
11	MANN_a45	342	4.20	343.6 (343–345)	769.63	344.5 (344–345)	20.79	345*
12	MANN_a81	1096	125.86	1098	5463.48	1099.3 (1098–1100)	896.95	1100
13	brock200_2	11	1.41	12	18.54	11.3 (11–12)	1.05	12*
14	brock200_4	16	0.91	17	37.62	16.9 (16–17)	6.80	17*
15	brock400_2	24.6 (24–25)	4.64	26.1 (25–29)	81.24	27.4 (25–29)	57.19	29*
16	brock400_4	24.2 (24–25)	6.21	32.4 (25–33)	209.67	33	36.90	33*
17	brock800_2	20.4 (20–21)	18.04	21	9.15	21	11.78	21
18	brock800_4	20	4.59	21	12.92	21	43.28	21
19	gen200_p0.9_44	40	0.70	44	0.07	44	0.91	44*
20	gen200_p0.9_55	54.8 (54–55)	3.71	55	0.03	55	0.24	55*
21	gen400_p0.9_55	52.0 (51–54)	24.05	55	2.32	54.8 (54–55)	34.28	55
22	gen400_p0.9_65	58.6 (50–65)	16.95	65	0.10	65	1.22	65
23	gen400_p0.9_75	65.8 (52–75)	19.86	75	0.10	75	1.00	75
24	hamming8-4	16	0.00	16	0.01	16	0.01	16*
25	hamming10-4	38.8 (38–40)	4.34	40	0.02	40	0.26	40
26	keller4	11	0.02	11	0.00	11	0.01	11*
27	keller5	27	4.11	27	0.33	27	0.38	27
28	keller6	56.8 (55–59)	346.82	59	366.34	58.2 (57–59)	245.26	59
29	p_hat300-1	8	0.20	8	0.04	8	0.02	8*
30	p_hat300-2	25	0.23	25	0.01	25	0.01	25*
31	p_hat300-3	36	2.77	36	0.04	36	0.07	36*
32	p_hat700-1	10.8 (10–11)	3.29	11	0.36	11	0.52	11*
33	p_hat700-2	44	2.96	44	0.05	44	0.05	44*
34	p_hat700-3	62	3.86	62	0.07	62	0.06	62
35	p_hat1500-1	11	4.05	12	58.43	12	415.68	12*
36	p_hat1500-2	65	2.71	65	0.30	65	0.28	65
37	p_hat1500-3	94	63.50	94	0.37	94	0.58	94
Average		75.3 (74.4–76.1)	24.93	76.8 (76.6–77.0)	350.15	76.9 (76.7–77.1)	174.44	76.9

DIMACS problem instances. For each method and instance the value and computational time are reported. In addition, average, minimum and maximum values for the clique number are given for those methods for which such information is available (i.e., for TS, RLS and VNS). For example, in column 7 of Table 4 for problem number 28 'keller6', the average value in 10 runs, worst and best value found are given as 58.2 (57 - 59). In the last column of both tables, are the best solutions found by the 15 heuristics presented at DIMACS challenge of 1993. The * sign denotes that optimality is proven.

The VNS version used (in Tables 3 and 4) is the H_0 variant from Table 1. Exceptions are 'brock' and 'mann' instances, where *random* and *max degree* ($\bar{G}$) selection rules are chosen in the *Add* step respectively.

We translate all CPU times (obtained on the different computers on which the cited methods were tested) to DIMACS machine CPU times following standard timing routines for MCP [21]. Thus times (in seconds) reported are all from the same "DIMACS machine".

It appears that (i) VNS gives better results than recent GA1, GA2, CBH and TS heuristics; (ii) VNS is comparable in terms of solution quality with the state-of-the-art heuristic, i.e., RLS. These results confirm the observation made in

[17,18] that, for a variety of combinatorial optimization problems, VNS can provide as good or better solutions than the state-of-the-art heuristics proposed, which are often more complicated and in almost all cases involve more parameters. It is worth noting that while RLS and VNS give similar results in terms of values found, they rely partly on different principle: indeed RLS exploits systematically long-term memory, while VNS does not do so at all. This suggest the possible interest of an hybrid of RLS and VNS, which will be the topic of future search.

References

[1] C. Aggarwal, J. Orlin, R. Tai, Optimized crossover for the independent set problem, Oper. Res. 45 (1997) 226–234.
[2] L. Babel, G. Tinhofer, A branch and bound algorithm for the maximum clique problem, Zeits. Oper. Res. 34 (1990) 207–217.
[3] T. Bäck, S. Khuri, An evolutionary heuristic for the maximum independent set problem, in: Proceedings of the first IEEE Conference on Evolutionary Computation, IEEE Press, New York, 1994, pp. 531–535.
[4] E. Balas, W. Niehaus, Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problem, J. Heuristics 4 (1998) 107–122.
[5] E. Balas, C.S. Yu, Finding a maximum clique in an arbitrary graph, SIAM J. Comput. 15 (4) (1986) 1054–1068.
[6] R. Battiti, M. Protasi, Reactive local search for the maximum clique problem, Algorithmica 29 (2001) 610–637.
[7] I. Bomze, M. Budinich, P. Pardalos, M. Pelillo, The maximum clique problem, in: D.Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, Kluwer Academic Publishers, Dordrecht, 1999.
[8] R. Carraghan, P.M. Pardalos, An exact algorithm for the maximum clique problemi, Oper. Res. Lett. 9 (1990) 375–382.
[9] T. Feo, M. Resende, S. Smith, A greedy randomized adaptive search procedure for maximum independent set, Oper. Res. 42 (1994) 860–878.
[10] C. Friden, A. Hertz, D. de Werra, Tabaris: an exact algorithm based on tabu search for finding a maximum independent set in a graph, Comput. Oper. Res. 17 (1990) 437–445.
[11] M. Garey, D. Johnson, Computers and Intractability, Freeman, New York, 1979.
[12] M. Gendreau, L. Salvail, P. Soriano, Solving the maximum clique problem using a tabu search approach, Ann. Oper. Res. 41 (1993) 385–403.
[13] L.E. Gibbons, D.W. Hearn, P.M. Pardalos, Continuous based heuristic for the Maximum Clique Problemi, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26, 1996, pp. 103–124.
[14] F. Glover, Tabu search, Part i, ORSA J. Comput. 1 (1989) 190–206.
[15] P. Hansen, B. Jaumard, Algorithms for the maximum satisfiability problem, Comput. 44 (1990) 279–303.
[16] P. Hansen, N. Mladenović, Two algorithms for maximum cliques in dense graphs, Technical Report GERAD Research Report G-92-18, 1992.
[17] P. Hansen, N. Mladenović, A comparison of algorithms for the maximum clique problem, Yugoslav J. Oper. Res. 2 (1992) 3–13.
[18] P. Hansen, N. Mladenović, An introduction to variable neighborhood search, in: S. Voss (Ed.), Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Dordrecht, 1999, pp. 433–458.
[19] P. Hansen, N. Mladenović, Variable neighborhood search: principles and applications, Eur. J. Oper. Res. 130 (2001) 449–467.
[20] D. Johnson, Approximation algorithms for combinatorial problems, J. Comput. Syst. Sci. 9 (1974) 256–278.
[21] D. Johnson, M. Trick, Cliques, Coloring and Satisfiability: Second Dimacs Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26.
[22] N. Mladenović, P. Hansen, Variable neighborhood search, Comput. Oper. Res. 24 (1997) 1097–1100.
[23] T.S. Motzin, E.G. Strauss, Maxima for graphs and a new proof of a theorem of turan, Can. J. Math. 17 (1965) 533–540.
[24] I.H. Osman, G. Laporte, Metaheuristics: a bibliography, in: G. Laporte, I.H. Osman (Eds.), Metaheuristics in Combinatorial Optimization, Ann. Oper. Research 63 (1996) 513–623.
[25] P. Pardalos, G. Rodgers, A branch and bound algorithm for the maximum clique problem, J. Algorithms 7 (1986) 425–440.
[26] P. Pardalos, J. Xue, The maximum clique problem, J. Global Opt. 4 (1994) 301–328.
[27] P. Soriano, M. Gendreau, Tabu search algorithms for the maximum clique problem, in: D. Johnson, M. Trick (Eds.), Clique, Coloring and Satisfiability, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26, American Mathematical Society, Providence, RI, 1966, pp. 221–244.
[28] E. Tomita, K. Imamatsu, Y. Kohata, M. Wakatsuki, A simple and efficient branch and bound algorithm for finding a maximum clique with experimental evaluations, Syst. Comput. Japan 28 (5) (1997) 60–67.