Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 151 (2003) 379–388
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

A variable neighborhood search for graph coloring

Cédric Avanthay [a], Alain Hertz [b], Nicolas Zufferey [c,*]

[a] *Private business corporation, Lausanne CH-9000, Switzerland*
[b] *Département de mathématiques et de génie industriel, École Polytechnique, Montréal, Québec, Canada H3C 3A7*
[c] *Department of Mathematics, Swiss Federal Institute of Technology, Lausanne CH-1015, Switzerland*

Abstract

Descent methods for combinatorial optimization proceed by performing a sequence of local changes on an initial solution which improve each time the value of an objective function until a local optimum is found. Several metaheuristics have been proposed which extend in various ways this scheme and avoid being trapped in local optima. For example, Hansen and Mladenovic have recently proposed the variable neighborhood search method which has not yet been applied to many combinatorial optimization problems. The aim of this paper is to propose an adaptation of this new method to the graph coloring problem.

Keywords: Variable neighborhood search; Graph coloring

1. Introduction

Let S denote the set of solutions of a combinatorial optimization problem. For a solution $s_i \in S$ let $N(s_i)$ denote the *neighborhood* of s_i which is defined as the set of *neighbor solutions* in S obtained from s_i by performing a *local change* on it. A descent method usually visits a sequence $s_0, \ldots, s_n$ of solutions, where s_0 is an initial solution, s_n is a local optimum, and $s_{i+1} \in N(s_i)$, $\forall i = 0, \ldots, n-1$. In recent years, local search techniques have been proposed which extend this scheme in order to get out of local optima. These techniques (e.g. tabu search [16,17], simulated annealing [22], GRASP [8], adaptative multi-start [2]) have led to much improved results for many combinatorial problems. Hansen and Mladenovic have recently [18,23] proposed a new optimization technique called variable neighborhood search (VNS for short). The main idea of this new method is to use various neighborhoods during the search. Given an incumbent s, a neighbor solution s' is generated according to one of these neighborhoods, and a local search is then applied to s' in order to obtain a local optimum s''. If s'' is better than s, then s'' becomes the new incumbent; otherwise, a different neighborhood is considered in order to try to improve upon solution s. This process is repeated until no neighborhood leads to an improvement of the incumbent.

Given a graph $G = (V, E)$ with vertex set V and edge set E, and given an integer k, a *k-coloring* of G is a function $c : V \to \{1, \ldots, k\}$. The value $c(x)$ of

* Corresponding author.
E-mail addresses: cav@elca.ch (C. Avanthay), alain.hertz@gerad.ca (A. Hertz), nicolas.zufferey@epfl.ch (N. Zufferey).

0377-2217/$ - see front matter © 2003 Published by Elsevier B.V.
doi:10.1016/S0377-2217(02)00832-9

a vertex x is called the *color* of x. The vertices with color r ($1 \leqslant r \leqslant k$) define a *color class*, denoted V_r. If two adjacent vertices x and y have the same color r, vertices x and y are called *conflicting vertices*, the edge $[x, y]$ is called a *conflicting edge*, and r is called a *conflicting color*. If there is no conflicting edge, then the color classes are called *stable sets*, and the k-coloring is said *legal*. The graph coloring problem (GCP for short) is to determine the smallest integer k (called *chromatic number* of G) such that there exists a legal k-coloring of G. The aim of this paper is to propose an adaptation of the VNS to the GCP.

The paper is organized as follows. In Section 2, we review some famous heuristic methods for graph coloring. We then present in Section 3 the basic VNS as well as some possible extensions. The proposed adaptation of the VNS method to the GCP is described in Section 4, and computational experiments are reported in Section 5.

2. Heuristic methods for graph coloring

The GCP is an NP-hard problem [12]. Exact solution methods have been developed by several researchers (e.g. [3,4,24]) but all these exact methods can only be applied to problems of relatively small size (no more than 100 vertices). This certainly explains why many heuristic methods have been proposed for getting an upper bound on the chromatic number of a graph. A survey of the most famous heuristic graph coloring methods can be found in [25]. The simplest ones are *construtive* methods which color each vertex in turn with the smallest possible color. The quality of the resulting coloring strongly depends on the order in which the vertices are scanned. A well-known such constructive method is the DSATUR algorithm [3], which is based on a dynamic ordering of the vertices: the next vertex to be colored is the one having the largest number of different colors already assigned to its adjacent vertices.

Given a fixed integer k, we consider the optimization problem, called k-GCP, which aims to determine a k-coloring of G such that the number of conflicting edges is minimized. If the optimal value of the k-GCP is zero, this means that G has a legal k-coloring. The chromatic number of G can be determined by first computing an upper bound (for example by means of a constructive method) and then solving a series of k-GCPs with decreasing values of k until no legal k-coloring can be obtained. This strategy will be called the *partition approach* since each k-GCP aims to determine a partition of the vertex set V of G into a fixed number k of stable sets. For a fixed integer k, and a k-coloring c, let $f(c)$ denote the number of conflicting edges induced by c. Many local search methods have been proposed for minimizing f. For example, a tabu search is described in [19], and simulated annealing algorithms can be found in [5,20]. Genetic algorithms and hybrid methods have also been successfully applied to this problem (e.g. [6,9,13]).

Another approach for solving the GCP is to successively build the stable sets of a legal k-coloring by repeatedly identifying a maximal stable set in G and removing it from the graph. With this strategy, the GCP is reduced to the solution of successive maximal stable set problems. Various efficient heuristic methods have been proposed for finding a large stable set in a graph (e.g. [10,11, 14]). The partition approach can be combined with this strategy by first removing stable sets until the remaining graph G' has no more than a given number of vertices, and then applying the partition approach on G'. Such combined methods are described for example in [9,19]. Nowadays, the most efficient heuristic method for the GCP is due to Galinier and Hao who have proposed an hybrid method that combines a genetic algorithm with a tabu search [13].

3. Description of the VNS method

Let $N^{(t)}$ $(t = 1, \ldots, t_{\max})$ denote a finite set of neighborhoods, where $N^{(t)}(s)$ is the set of solutions in the tth neighborhood of s. Most local search methods use only one type of neighborhood, i.e. $t_{\max} = 1$. The basic VNS [18,23], which is described in Table 1, tries to avoid being trapped in local minima with the help of more than one neighborhood.

Table 1
The basic VNS

1. Initialisation
- Determine an initial solution s
- Set $t = 1$

2. Repeat the following until a stopping condition is met
- 2.a *Shaking*. Generate a point s' at random from the tth neighborhood of s ($s' \in N^{(t)}(s)$)
- 2.b *Local search*. Apply some local search method with s' as initial solution; let s'' be the so obtained local optimum
- 2.c *Move or not*. If s'' is better than the incumbent s, move there (i.e. set $s = s''$), and continue the search with $N^{(1)}$ (i.e. set $t = 1$); otherwise set $t = (t \bmod t_{\max}) + 1$

The stopping condition of a VNS may be, for example, an upper bound on the CPU time, a maximum number of iterations, or a maximum number of iterations between two improvements of the incumbent. Observe that a solution s' generated in step 2.a is obtained by randomly choosing it in the tth neighborhood. This way of doing may avoid cycling which might occur if any deterministic rule was used. The local search method used in step 2.b can be a simple descent method, or a more powerful technique such as a tabu search or a simulated annealing.

The above basic VNS can be viewed as a descent algorithm since the incumbent s is modified only if the local optimum s'' obtained in step 2.b is better than s. Without much additional effort, it is possible to transform this basic VNS into a *descent–ascent* method: in step 2.c set also $s = s''$ with some probability even if s'' is worse than the incumbent. Another possible variant of the basic VNS is to choose solution s' in step 2.a as the best solution in a subset of neighbors randomly generated in $N^{(t)}(s)$.

4. Adaptation of VNS to the k-GCP

In order to solve the GCP, we have decided to follow the partition approach. For a fixed k, we consider as a solution to the problem any k-coloring c. By denoting E_r the collection of edges of G with both endpoints in V_r, the objective function f is defined as the total number of conflicting edges, i.e. $f(s) = \sum_{r=1}^{k} |E_r|$. The initial solution generated in step 1 of the VNS algorithm is randomly chosen among all possible solutions.

We stop the VNS algorithm when $\mathrm{Max}_{\mathrm{VNS}}$ iterations have been performed without improving the incumbent s. We have set $\mathrm{Max}_{\mathrm{VNS}} = |V|$. Preliminary experiments have shown that a larger value for $\mathrm{Max}_{\mathrm{VNS}}$ does not lead to a significant improvement of the algorithm. As local search method in step 2.b we use the *Tabucol* algorithm described in [19] which is a tabu search that follows the partition approach. Given a solution s, Tabucol generates a neighbor solution s' by changing the color of a conflicting vertex. This means that a conflicting vertex x in a color class V_r is moved to a new color class V_j, and the pair (x, r) is introduced in the tabu list (i.e. vertex x cannot be moved back to V_r for some iterations). Tabucol is stopped when $\mathrm{Max}_{\mathrm{tabu}}$ consecutive moves have been performed without improving the best solution found so far. A large value for $\mathrm{Max}_{\mathrm{tabu}}$ induces a large CPU time for the VNS algorithm, while a too low value does not offer enough time to Tabucol to improve on s'. Here again, preliminary experiments have shown that $\mathrm{Max}_{\mathrm{tabu}} = 10 \cdot |V|$ is a good compromise. Also, we have chosen value 10 for the length of the tabu list.

In the three next subsections we describe the various neighborhoods used in our adaptation of the VNS method to the k-GCP. These neighborhoods can be divided into three groups:

- the *vertex neighborhoods* which change the color of some conflicting vertices,
- the *class neighborhoods* which change the color of some or all vertices of a conflicting color,
- the *non-increasing neighborhoods* which change the color of some vertices without increasing the total number of conflicting edges.

4.1. Vertex neighborhoods

We describe below five neighborhoods which modify the incumbent s by changing the color of some conflicting vertices.

(1) *The basic neighborhood:* From a solution s we generate a neighbor solution in $N^{(1)}(s)$ as follows. We randomly choose a conflicting vertex x and move it from its current color class to the *best* possible other one (i.e. the new color class for x is chosen among those having the smallest number of vertices adjacent to x). Such a move may create *new conflicting vertices*. We successively apply i such changes on s to obtain a neighbor solution s'. On the one hand, a low value for i induces a neighbor solution s' which is very close to s. On the other hand, a large value for i will create a neighbor solution s' which is very different from s, and the generation of a neighbor solution can therefore be seen, in this case, as a restart. We have decided to randomly choose i in $\{10,\ldots,i^{(1)}_{\max}\}$, where $i^{(1)}_{\max}$ depends on the number I_{VNS} of iterations which have been performed without improving the incumbent s as follows: we set $i^{(1)}_{\max}=100$ each time the incumbent s is improved (i.e. when I_{VNS} is set equal to 0), and we linearly decrease $i^{(1)}_{\max}$ down to the value 20 which is reached when $I_{\mathrm{VNS}}=\mathrm{Max}_{\mathrm{VNS}}$. Such a choice means that we perform large changes on the incumbent s when I_{VNS} has a small value, while small changes are preferred when s has not been improved for a long time. The idea behind this choice is to perform a kind of diversification each time s is improved, and to gradually switch to intensification when I_{VNS} increases. The upper and lower bounds on $i^{(1)}_{\max}$ (as well as on $i^{(2)}_{\max},\ldots,i^{(5)}_{\max}$ defined below) result from preliminary experiments. Notice also that we have imposed a lower bound of value 10 on i. The reason is that we use the *basic neighborhood* for moving to a new region of the solution space, while the task of Tabucol (which uses $i=1$) is to carefully explore this new region. Notice that when generating a neighbor of s, we take care of not changing the color of a vertex more than once.

(2) *The chain neighborhood:* From a solution s we generate a neighbor solution in $N^{(2)}(s)$ as follows. We first randomly choose a conflicting vertex x (called *origin vertex*) and move it into the best possible other color class V_j. Since s is a local optimum, this move will create new conflicting vertices in V_j. We then choose at random a new conflicting vertex $y\in V_j$ and assign to it the best possible new color l. This second move will probably create new conflicts in V_l in which case we assign the best possible color to a new conflicting vertex in V_l. This sequence of changes is executed as long as possible, taking care of not changing the color of a vertex more than once. We repeat this process by performing successively i such sequences of changes, with i origin vertices, where i is randomly chosen in $\{1,\ldots,i^{(2)}_{\max}\}$. The upper bound $i^{(2)}_{\max}$ on i decreases gradually from 20 to 5 when I_{VNS} increases from 0 to $\mathrm{Max}_{\mathrm{VNS}}$.

(3) *The grenade neighborhood:* From a solution s we generate a neighbor solution in $N^{(3)}(s)$ as follows. We first randomly choose a conflicting vertex x (called *grenade*) and we move it into the best possible other color class V_j. We then sequentially move each new conflicting vertex from V_j into the best possible other color class. This process is repeated with i different grenades, where i is randomly chosen in $\{1,\ldots,i^{(3)}_{\max}\}$. The upper bound $i^{(3)}_{\max}$ on i decreases gradually from 40 to 1 when I_{VNS} increases from 0 to $\mathrm{Max}_{\mathrm{VNS}}$.

(4) *The firework neighborhood:* From a solution s we generate a neighbor solution in $N^{(4)}(s)$ as follows. We first randomly choose a conflicting vertex x (called *firework*) and move it into the best possible other color class V_j. We then consider every new conflicting vertex as a grenade (see above). This process is repeated with i different fireworks, where i is randomly chosen in $\{1,\ldots,i^{(4)}_{\max}\}$. The upper bound $i^{(4)}_{\max}$ on i decreases gradually from 30 to 1 when I_{VNS} increases from 0 to $\mathrm{Max}_{\mathrm{VNS}}$.

(5) *The permutation neighborhood:* From a solution s we generate a neighbor solution in $N^{(5)}(s)$ as follows. We first randomly choose a conflicting vertex and move it from its current color class V_r into the best possible other color class V_j. Among the new conflicting vertices in V_j, we choose one having the smallest number of adjacent vertices in V_r, and move it into V_r. We successively perform i such permutations, where i is randomly chosen in $\{1,\ldots,i^{(5)}_{\max}\}$. The upper bound $i^{(5)}_{\max}$ on i decreases

gradually from 50 to 10 when I_{VNS} increases from 0 to Max_{VNS}.

4.2. Class neighborhoods

In these neighborhoods, we modify a solution by first selecting a conflicting color, and then changing the color of some or all vertices having that color. We propose below four such neighborhoods. In what follows, we denote V^* the color class which contains the largest number of conflicting vertices, and p its number of vertices.

(6) *The empty–refill neighborhood:* From a solution s we generate a neighbor solution in $N^{(6)}(s)$ as follows: we first empty V^* by successively moving each of its vertices into the best possible other color class. We then refill V^* by successively choosing p other vertices (new conflicting vertices when possible) and moving all of them into V^*.

(7) *The stable set neighborhood:* From a solution s we generate a neighbor solution in $N^{(7)}(s)$ as follows: we first randomly choose a conflicting vertex $x \in V^*$, and create an ordered list $L = (v_1, \ldots, v_{|V|})$ of the vertices in V, where $v_1 = x$, the $p-1$ last elements of L are the vertices of $V^* - \{x\}$, and the elements $v_2, \ldots, v_{|V|-p}$ cover the remaining vertices of V (in a random order). We then construct a maximal stable set W in a greedy fashion, by scanning the ordered list L. Finally, we move each vertex which is in V^* but not in W into the best possible other color class, and we move into V^* each vertex which is in W but not yet in V^*.

(8) *The empty class neighborhood:* From a solution s we generate a neighbor solution in $N^{(8)}(s)$ as follows: we first empty V^* by successively moving each of its vertices into the best possible other color class. We thus obtain a partition of V into $k-1$ color classes. We then briefly apply the Tabucol algorithm (Max_{tabu} is set equal to $|V|$) in order to minimize the number of conflicts in these $k-1$ color classes (i.e. the addition of a vertex into V^* is considered as a tabu move).

(9) *The tabu class neighborhood:* From a solution s we generate a neighbor solution in $N^{(9)}(s)$ as follows. We first reduce V^* to a stable set by sequentially moving each of its conflicting vertices into the best possible other color class. We then briefly apply the Tabucol algorithm to this solution (Max_{tabu} is set equal to $|V|$), while forbidding any change on V^* (i.e. the addition of a vertex into V^* and the removal of a vertex from V^* are considered as tabu moves).

4.3. Non-increasing neighborhoods

In these neighborhoods, we generate a neighbor solution by changing the color of some vertices, but without increasing the total number of conflicting edges. This means that the neighbor solution s' is such that $f(s') \leqslant f(s)$.

(10) *The Culberson neighborhood* [6,7]*:* From a solution $s = (V_1, \ldots, V_k)$ we generate a neighbor solution in $N^{(10)}(s)$ as follows. We first remove the $f(s)$ conflicting edges in G. We thus get a partial subgraph G' for which the partition $(V_1, \ldots, V_k)$ corresponds to a legal k-coloring c. We then consider any permutation π of the k colors and any ordering of the vertices such that vertex x is before vertex y if $\pi(c(x)) < \pi(c(y))$. We finally construct a new k-coloring $(V'_1, \ldots, V'_k)$ of G' by coloring each vertex in turn with the smallest possible color. The partition $s' = (V'_1, \ldots, V'_k)$ is a neighbor solution of s with at most $f(s)$ conflicting edges. This process is illustrated in Fig. 1.

(11) *The subgraph neighborhood* [15]*:* From a solution $s = (V_1, \ldots, V_k)$ we generate a neighbor solution in $N^{(11)}(s)$ as follows. We first sequentially remove conflicting vertices from G in order to obtain a maximal induced subgraph G' of G without any conflicting edge. We then permute the colors in G' according to the permutation π which minimizes the number of conflicting edges having one endpoint in G' and one outside G'. This best permutation is obtained by solving a *bipartite weighted matching problem* (also called *assignment problem*) [1]. More precisely, consider the complete bipartite graph $B = (N_1 \cup N_2, A)$ where $N_1 = N_2 = \{1, \ldots, k\}$, and the cost C_{ij} of the edge $(i,j) \in A$ linking $i \in N_1$ with $j \in N_2$ is equal to the number of edges linking a vertex of color i outside G' to a vertex of color j in G'. Let M be a perfect matching of minimum cost in B. The total cost of the edges in M corresponds to the number of conflicting edges in the k-coloring obtained from s by assigning color j to a vertex of color i in G' if and only if the edge linking $i \in N_1$ to $j \in N_2$ belongs to

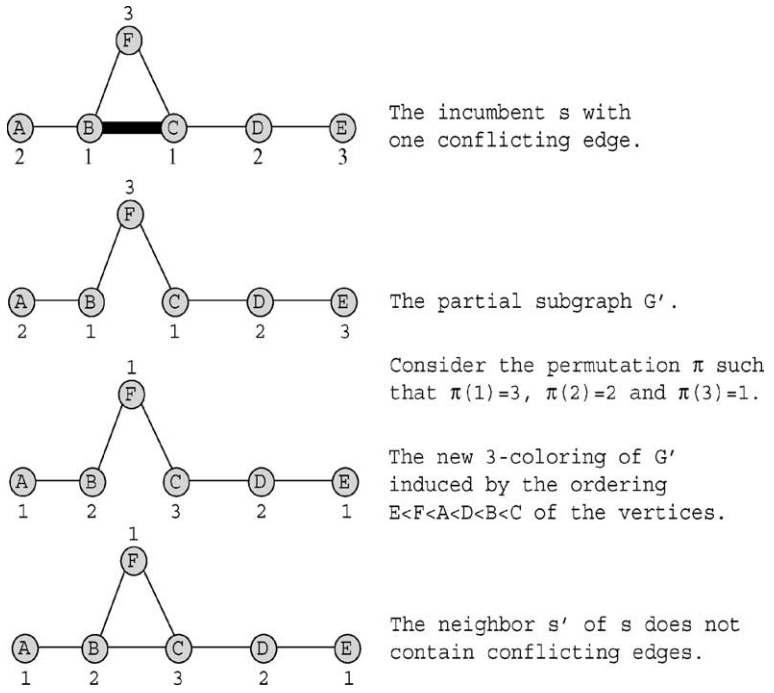

Fig. 1. The Culberson neighborhood.

M. Many polynomial algorithms have been proposed for the solution of the bipartite weighted matching problem (see for example [1]). This process is illustrated in Fig. 2.

4.4. Comparison of the neighborhoods

In order to analyze the efficiency of each proposed neighborhood, we have implemented eleven different VNS algorithms, each one using a single neighborhood (i.e., $t_{\max} = 1$). As mentioned in [13], differences between efficient algorithms can not be observed on instances of small size (with less than 300 vertices). The comparison of the neighborhoods has therefore been performed on nine instances of medium size. We have first generated four random graphs with 500 vertices and edge density 0.5, denoted *R500.1*, *R500.2*, *R500.3* and *R500.4*. We have then taken the five following instances from [21].

- Two *flat graphs*: *flat300.28* and *flat300.26*. They both have 300 vertices and a known chromatic number (28 and 26, respectively).
- Two *Leighton graphs*: *le450.15c* and *le450.15d*. They both have 50 vertices and a known chromatic number 15.
- One *random graph*: *DSJC500.5* with 500 vertices, edge density 0.5, and with unknown chromatic number.

Notice that each one of these nine instances defines in reality a set of k-GCPs with different values of k. In the second line of Table 2, we indicate the value of k for which we try to find a legal k-coloring. Then, for each neighborhood, we give the average number of conflicting edges obtained after 4 runs of the VNS algorithm (with the fixed k of the second line), using only the considered neighborhood. In the last line we give, for comparison, the average results obtained with Tabucol, also after 4 runs on each instance. To get comparable CPU times we have stopped each run of Tabucol after $\mathrm{Max}_{\mathrm{VNS}} \cdot \mathrm{Max}_{\mathrm{tabu}} = 10 \cdot |V|^2$ iterations without improvement of the best solution obtained so far. Bold numbers indicate that the considered VNS algorithm has found solutions which are in

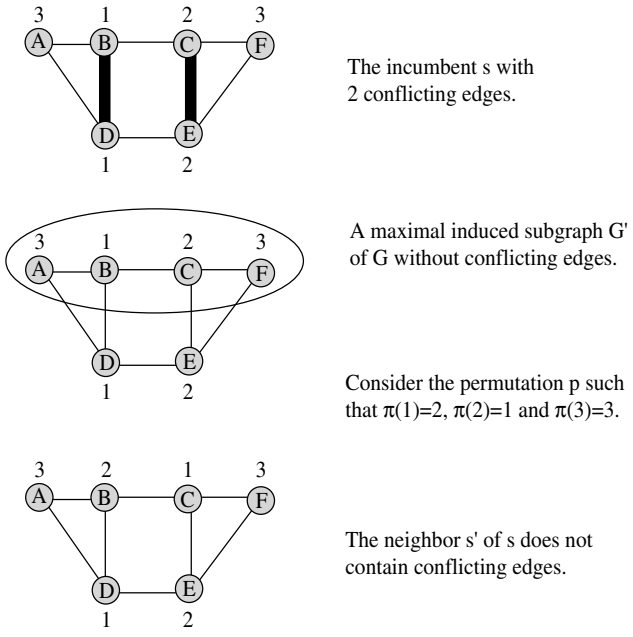

Fig. 2. The subgraph neighborhood.

Table 2
Comparison of the neighborhoods

	Instance								
	R500.1	*R500.2*	*R500.3*	*R500.4*	*DSJC500.5*	*le450.15c*	*le450.15d*	*flat300.28*	*flat300.26*
k	49	49	49	49	49	15	15	31	31
Chain	2.5	**4**	**3.75**	**3.25**	**4.5**	**0**	**0**	**1.75**	**0**
Firework	**1.75**	**5**	**4.5**	4	**4**	**0**	**0**	**2.25**	**0**
Grenade	2	**4.5**	**3.25**	3.75	**4.25**	**0**	**0.25**	**1.5**	**0**
Permutation	**1.75**	**5.5**	**4.5**	**3.5**	**6**	**0.25**	**2**	**2**	**0**
Basic	2	**5.5**	**5**	7.75	**8**	**83.75**	**46**	**2.5**	**0**
Stable set	2.5	**4.5**	**3.75**	**3**	**5.5**	**2.5**	**1**	**2.5**	**1.25**
Empty–refill	2.5	**3.5**	**5.5**	3.75	**4.25**	**2**	**0**	**2**	**1**
Empty class	2.5	**3**	**5.25**	3.75	**4.75**	**11.5**	**16.75**	**1.25**	**0**
Tabu class	7	9.75	9.25	10	13	**26.5**	**17.75**	6	4.75
Culberson	8	12.5	17.5	10.75	13.25	**162.25**	163.5	7.25	6.25
Subgraph	12.5	16.25	14	16.5	16.75	**160.75**	168	8	6.75
Tabucol	1.75	6	8	3.5	9	162.25	160.75	2.5	1.5

average at least as good as those produced by Tabucol.

Notice first that for each instance, except *R500.1*, we have been able to improve on Tabucol results by using a VNS approach. Table 2 clearly shows that the *chain*, *firework* and *grenade* neighborhoods are the best ones among the vertex neighborhoods, while the *stable set*, *empty–refill* and *empty class* neighborhoods are the best among the class neighborhoods. The non-increasing neighborhoods are not competitive at all. Considering the above results, we have decided to use only these six best neighborhoods. Tests on larger instances have confirmed that the five rejected

neighborhoods do not help improving the performance of the VNS algorithm.

4.5. The proposed VNS algorithm

We have combined the six best neighborhoods $N^{(2)}(s)$, $N^{(3)}(s)$, $N^{(4)}(s)$, $N^{(6)}(s)$, $N^{(7)}(s)$ and $N^{(8)}(s)$ to obtain the proposed VNS algorithm which is described in Table 3. Since the algorithm stops when the incumbent has not been improved for $\mathrm{Max}_{\mathrm{VNS}} = |V|$ iterations, we change from a neighborhood to another one when $|V|/6$ iterations are performed with a same neighborhood without improving s. The order in which the six neighborhoods are considered can influence the quality of the solution produced by the VNS algorithm. We have tested the six following permutation of $\{2,3,4,6,7,8\}$ (among the 720 possible ones):

- vertex first, class second permutations: $\pi_1 = 2,3,4,6,7,8$, $\pi_2 = 4,3,2,8,7,6$;
- class first, vertex second permutations: $\pi_3 = 6,7,8,2,3,4$, $\pi_4 = 8,7,6,4,3,2$;
- alternating permutations: $\pi_5 = 2,6,3,7,4,8$, $\pi_6 = 6,2,7,3,8,4$.

The six permutations have been tested on the nine instances described in Section 4.4. The results are reported in Table 4. We first recall in the first lines the results obtained using only one neighborhood. The best average number of conflicting edges is indicated with bold numbers. We then report the results obtained by combining the six

Table 3
The proposed VNS for the k-GCP

1. Initialisation
1.a Randomly generate a k-coloring of G
1.b Set $I_{\mathrm{VNS}} = 0$ and $t = 1$
1.c Consider a permutation π of $\{2,3,4,6,7,8\}$

2. Repeat until $I_{\mathrm{VNS}} = |V|$
Set $It_{\mathrm{VNS}} = It_{\mathrm{VNS}} + 1$
2.a *Shaking*. Generate a random solution s' in $N^{(\pi(t))}$
2.b *Local search*. Apply Tabucol to s' and stop when $10 \cdot |V|$ have been performed without improving the best known solution. Let s'' be the so obtained local optimum
2.c *Move or not*. If s'' is better than s, then set $s = s''$, $t = 1$ and $I_{\mathrm{VNS}} = 0$; otherwise, set $t = t + 1$ if I_{VNS} is a multiple of $\lceil \frac{|V|}{6} \rceil$

Table 4
Comparison of the orderings

	Instance								
	R500.1	*R500.2*	*R500.3*	*R500.4*	*DSJC500*	*le450.15c*	*le450.15d*	*flat300.28*	*flat300.26*
k	49	49	49	49	49	15	15	31	31
Chain	2.5	4	3.75	**3.25**	4.5	**0**	**0**	1.75	**0**
Firework	**1.75**	5	4.5	4	**4**	**0**	**0**	2.25	**0**
Grenade	2	4.5	**3.25**	3.75	4.25	**0**	0.25	1.5	**0**
Stable set	2.5	4.5	3.75	3	5.5	0.25	2	2	**0**
Empty–refill	2.5	3.5	5.5	3.75	4.25	2	**0**	2	1
Empty class	2.5	**3**	5.25	3.75	4.75	11.5	16.75	**1.25**	**0**
π_1	2.5	3.75	4.5	**2.5**	**3.25**	**0**	**0**	2.5	**0**
π_2	**1.5**	4.5	**3**	**2.25**	5	**0**	**0**	2.25	**0**
π_3	2	**3**	5.25	3.75	4.25	**0**	**0**	2.25	**0**
π_4	3.5	4.5	**2.75**	**2.25**	5.75	**0**	**0**	2.75	**0**
π_5	2	3.5	3.75	**1.75**	5	**0**	**0**	2.25	**0**
π_6	2.25	3.25	5.25	3.75	5.75	**0**	**0**	**0.75**	**0**

neighborhoods. Bold numbers indicate that the considered permutations has produced solutions which are in average at least as good as those obtained using only one kind of neighborhood. Notice first that the average number of conflicting edges has been reduced for instances *R500.1*, *R500.3*, *R500.4*, *DSJC500* and *flat300.28*. It can however be observed that no permutation clearly appears as a winner. We have therefore decided to choose a random permutation of $\{2, 3, 4, 6, 7, 8\}$ at point 1.c of the above VNS algorithm.

5. Comparisons between VNS and other methods

To further analyze the performance of the proposed VNS algorithm, we have compared the results it produces with those obtained by Tabucol and the genetic hybrid algorithm (GH for short) proposed by Galinier and Hao [13], which combines a tabu search with a genetic algorithm. The three algorithms have produced the same results on all benchmark problems in [21], except on the random graph *DSJC1000.5* (with 1000 vertices and edge density 0.5), on the flat graph *flat1000.76* (with 1000 vertices and chromatic number 76), and on the instances studied in Section 4. We therefore only report results obtained on the nine instances of Section 4, as well as on these two additional graphs with 1000 vertices.

The proposed VNS algorithm as well as Tabucol have been run 4 times on each instance, and we report in Table 5 the smallest k with which each algorithm had at least one successful run (a successful run is one which finds a legal k-coloring for the fixed k). The five results reported for GH are taken from [13]. We do not know what can be obtained by this algorithm on the 6 other instances. In column *best known* we indicate the smallest value of k for which a legal k-coloring has ever been found by an algorithm [13]. Average computational times for the VNS algorithm (over the 4 runs) are given in the last column, for each instance. Notice that Tabucol requires approximately the same CPU time since we stop each run of Tabucol after $\mathrm{Max}_{\mathrm{VNS}} \cdot \mathrm{Max}_{\mathrm{tabu}} = 10 \cdot |V|^2$ iterations without improvement of the best solution obtained so far. We have used a Silicon Graphics Indigo2 machine (195 MHz, IP28 processor). To get a point of comparison, we have run program *dfmax* found at ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/volume/Machine/.

Our machine respectively needed 0.02, 0.69, 6.06, 37.65, 145.50 seconds to solve instances *r*100.5.*b*, *r*200.5.*b*, *r*300.5.*b*, *r*400.5.*b*, *r*500.5.*b*, and the Sun Sparc 10/Model 41 (using gcc and the "-O" optimization option) needed 0.04, 1.04, 9.06, 56.33, 218.69 seconds, respectively.

As can be observed, the VNS algorithm always produces better results than Tabucol. Notice that the VNS algorithm repeatedly uses Tabucol each time it enters step 2.b. This clearly demonstrates that the use of more than one neighborhood may be very useful for escaping local optima. The VNS algorithm is however not competitive with GH, which means that a good exchange of information

Table 5
Comparison between Tabucol, GH and our VNS method

Graph	Tabucol	VNS	GH	Best known	CPU time of VNS (minutes)
DSJC500.5	51	49	48	48	45
DSJC1000.5	94	90	83	83	180
R500.1	50	49	–	–	45
R500.2	51	50	–	–	45
R500.3	51	50	–	–	45
R500.4	51	50	–	–	45
le450.15c	18	15	15	15	5
le450.15d	18	15	–	15	5
flat300.28	32	31	31	31	15
flat300.26	32	31	–	26	15
flat1000.76	93	89	83	83	180

among a population of solutions seems to be more important than the use of several neighborhoods on a unique solution.

6. Conclusion

We have developed an adaptation of the variable neighborhood search method to the graph coloring problem. Our VNS algorithm is however not competitive with the hybrid algorithm GH proposed by Galinier and Hao, which combines a tabu search with a genetic algorithm, and which is for the moment the best known coloring algorithm. Their algorithm benefits from the advantages of both solution approaches: while genetic algorithms are well adapted for determining good regions in the search space, local search techniques have proven to be successful in determining high quality solutions in identified good regions of the search space. The next step for improving the results produced by the GH algorithm could be to replace the tabu search used in GH by a VNS method.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice Hall, Englewood Cliffs, NJ, 1993.
[2] K.D. Boese, A.B. Kahng, S. Muddu, A new adaptative multi-start technique for combinatorial global optimizations, Operations Research Letters 16 (1994) 101–113.
[3] D. Brélaz, New methods to color vertices of a graph, Communications of ACM 22 (1979) 251–256.
[4] J.R. Brown, Chromatic scheduling and the chromatic number problem, Management Science 19 (1972) 456–463.
[5] M. Chams, A. Hertz, D. de Werra, Some experiments with simulated annealing for coloring graphs, European Journal of Operational Research 32 (1987) 260–266.
[6] D. Costa, A. Hertz, O. Dubuis, Embedding of a sequential algorithm within an evolutionary algorithm for coloring problems in graphs, Journal of Heuristics 1 (1995) 105–128.
[7] J.C. Culberson, Exploring the *k*-colorable landscape with Iterated Greedy, in [21], 1996, pp. 245–284.
[8] T. Feo, M. Resende, Greedy randomized adaptative search, Journal of Global Optimization 6 (1995) 109–133.
[9] C. Fleurent, J.A. Ferland, Genetic and hybrid algorithms for graph coloring, Annals of Operations Research 63 (1996) 437–461.
[10] C. Fleurent, J.A. Ferland, Objected-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability, in [21], 1996, pp. 619–652.
[11] C. Frieden, A. Hertz, D. de Werra, Stabulus: A technique for finding stable sets in large graphs with tabu search, Computing 42 (1989) 35–44.
[12] M. Garey, D. Johnson, Computers and intractibility: A guide to the theory of NP-completeness, W.H. Freeman and Company, New York, 1979.
[13] P. Galinier, J.-K. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization 3 (1999) 379–397.
[14] M. Gendreau, P. Soriano, L. Salvail, Solving the maximum clique problem using a tabu search approach, Annals of Operations Research 41 (1993) 385–403.
[15] C. Glass, Seminar at the Department of Mathematics, Swiss Federal Institute of Technology, Lausanne, Switzerland, 1999.
[16] F. Glover, Tabu search—Part I, ORSA Journal of Computing 1 (1989) 190–206.
[17] F. Glover, Tabu search—Part II, ORSA Journal of Computing 2 (1990) 4–32.
[18] P. Hansen, N. Mladenovic, An introduction to VNS, in: S. Voss, S. Martello, I.H. Osman, C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, 1998.
[19] A. Hertz, D. de Werra, Using tabu search techniques for graph coloring, Computing 39 (1987) 345–351.
[20] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: An experimental evaluation, Part II: Graph coloring and number partitioning, Operations Research 39 (1991) 378–406.
[21] D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring and Satisfiability, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, American Mathematical Society, 1996.
[22] S. Kirkpatrick, C.D. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[23] N. Mladenovic, P. Hansen, Variable neighborhood search, Computers in Operations Research 24 (1997) 1097–1100.
[24] J. Peemöller, A correction to Brélaz's modification of Brown's coloring algorithm, Communications of ACM 26 (1983) 593–597.
[25] D. de Werra, Heuristics for graph coloring, Computing 7 (1990) 191–208.