Discrete Applied Mathematics 156 (2008) 2551–2560

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Variable space search for graph coloring

Alain Hertz [a], Matthieu Plumettaz [b], Nicolas Zufferey [c,*]

[a] *École Polytechnique de Montréal, Département de mathématiques et de génie industriel, Montréal (QC) H3C 3A7, Canada*

[b] *École Polytechnique Fédérale de Lausanne (EPFL), Chaire de Recherche Opérationnelle Sud-Est (ROSE), CH-1015 Lausanne, Switzerland*

[c] *HEC-Université de Genève, Faculté des Sciences Économiques et Sociales, 1211 Genève 4, Switzerland*

ARTICLE INFO

Article history:
Received 19 December 2006
Received in revised form 6 July 2007
Accepted 6 March 2008
Available online 28 April 2008

Keywords:
Graph coloring
Local search
Variable space search

ABSTRACT

Let $G = (V, E)$ be a graph with vertex set V and edge set E. The k-coloring problem is to assign a color (a number chosen in $\{1, \ldots, k\}$) to each vertex of G so that no edge has both endpoints with the same color. We propose a new local search methodology, called Variable Space Search, which we apply to the k-coloring problem. The main idea is to consider several search spaces, with various neighborhoods and objective functions, and to move from one to another when the search is blocked at a local optimum in a given search space. The k-coloring problem is thus solved by combining different formulations of the problem which are not equivalent, in the sense that some constraints are possibly relaxed in one search space and always satisfied in another. We show that the proposed algorithm improves on every local search used independently (i.e., with a unique search space), and is competitive with the currently best coloring methods, which are complex hybrid evolutionary algorithms.

1. Introduction

The *Graph Coloring Problem* (GCP for short) is a well known NP-hard problem [11]. Given a graph $G = (V, E)$, with vertex set V and edge set E, the GCP is to assign a color to every vertex, such that no edge has both endpoints with the same color, while minimizing the number of used colors. The smallest number of colors needed to color G is called the *chromatic number* of G and is denoted $\chi(G)$. The applications include scheduling, frequency assignment, register allocation and stock management [25]. Although many exact algorithms have been devised for this problem [3–5,14,17,19,23], such algorithms can only be used to solve small instances (up to 100 vertices). Heuristics coloring algorithms, on the other hand, can be used on much larger instances, but only to get an upper bound on $\chi(G)$. The most efficient heuristic algorithms are local search methods (e.g., [1,2,15]) and population based methods (e.g. [7,8,16,18,22]). For more information about such algorithms, the reader may refer to [9].

We propose in this paper a new local search methodology, called *Variable Space Search* (VSS for short). It is an extension of the well known Variable Neighborhood Search (VNS for short) [20]. While VNS uses several neighborhoods to escape from local optimum in a search space, we propose to use different formulations of the same problem, each one being associated with its proper search space, neighborhoods and objective function. VSS moves from a search space to another when it is trapped in a local optimum.

In the next section, we describe VSS with more details, while Section 3 contains three formulations of the graph coloring problem, each one being associated with a search space, neighborhoods and an objective function. We also describe how to translate a solution from a search space to another. Section 4 demonstrates how the search spaces complement each other.

* Corresponding address: HEC - University of Geneva, Boulevard du Pont-d'Arve 40, 1211Geneva 4, Switzerland.
E-mail addresses: alain.hertz@gerad.ca (A. Hertz), matthieu.plumettaz@epfl.ch (M. Plumettaz), nicolas.zufferey@hec.unige.ch (N. Zufferey).

0166-218X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2008.03.022

More precisely, we give examples where a solution in a search space is a local optimum, while translating this solution into another search space makes it possible to improve the solution. In Section 5, we describe the proposed adaptation of VSS to the graph coloring problem. Section 6 is devoted to computational experiments and we conclude with final remarks.

2. Variable Space Search

Three ingredients must be defined when designing a local search for a particular problem: a *search space* S, an *objective function* $f(s)$ that measures the quality of each solution in S, and a *neighborhood structure* $N(s)$. A local search generates a sequence $s_0, s_1, \ldots, s_r$ of solutions in S, where s_0 is an initial solution and each s_i $(i > 0)$ belongs to $N(s_{i-1})$. The transformation from s_i to s_{i+1} is called a *move*. Tabu Search (TS for short) is one of the most famous local search algorithms. In order to avoid cycling, TS uses a *tabu list* that contains forbidden moves. Hence, a move m from s_{i-1} to s_i can only be performed if m does not belong to the tabu list, unless $f(s_i) < f(s^*)$, where s^* is the best solution encountered so far. For more details on Tabu Search, the reader may refer to [13].

In 1997, Mladenović and Hansen [20] proposed the VNS algorithm that uses several neighborhoods to better diversify the search and better escape from local optima. We propose to use not only several neighborhoods, but also several objective functions and several search spaces.

Consider a set of search spaces $\{S_1, S_2, \ldots, S_r\}$ with their respective objective functions $\{f_1, f_2, \ldots, f_r\}$. For each search space S_i, consider a set $\mathcal{N}_i$ of neighborhoods which can be used in S_i for minimizing f_i. Consider finally a set of translators T_{ij} that transform any solution in S_i into a solution in S_j. The following algorithm, called Variable Space Search (or VSS for short), performs a local search in the different search spaces, always using the associated neighborhoods and objective function.

Algorithm 1 Variable Space Search

```
Set i := 1
Generate an initial solution s ∈ S_1
while no stopping criterion is met do
    Perform a local search in S_i, with objective function f_i, using the neighborhoods in 𝒩_i, and starting from s; let s′ be the
    resulting solution;
    Translate s′ into a solution s ∈ S_j using T_ij, where j = (i mod r) + 1;
    Set i ← (i mod r) + 1;
end while
```

The above algorithm can be modified in various ways, for example by choosing the next search space according to the quality of the solutions it provided in the past. The idea of using more than one search space has been already proposed and used in [21], where a circle packing problem is solved using two formulations, one with Cartesian and the other one with polar coordinates. Their algorithm, called *Reformulation Descent*, is however different from VSS. First of all, the search spaces considered in [21] both contain the same set of solutions since they only differ in the way of coding a solution. For comparison, VSS does not require a one-to-one correspondence between the solutions in S_i and those in S_j $(i \neq j)$. For example, a constraint can be relaxed in one search space S_i (and violations are then penalized in the objective function f_i), while it can be always satisfied in another. As a consequence, a neighborhood which is appropriate for a solution space S_i possibly generates non-feasible solutions for another search space. This is not the case in the Reformulation Descent of [21] since the same kind of moves to neighbor solutions is considered in all search spaces. Notice also that the Reformulation Descent algorithm uses a descent algorithm in each search space, while VSS can use any local search technique (e.g., tabu search, simulated annealing).

3. Three search spaces for graph coloring

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 $c : V \longrightarrow \{1, \ldots, k\}$. The value $c(x)$ of a vertex x is called the color of x. The vertices with color i $(1 \leq i \leq k)$ define a *color class*, denoted V_i. If two adjacent vertices x and y have the same color i, vertices x and y, the edge $[x, y]$ and color i are said to be *conflicting*. A k-coloring without conflicting edges is said to be *legal* and its color classes are called *stable sets*. The Graph Coloring Problem (*GCP* for short) is to determine the smallest integer k, called the *chromatic number* of G and denoted $\chi(G)$, such that there exists a legal k-coloring of G.

Given a fixed integer k, the optimization problem $k - GCP$ is to determine a k-coloring of G that minimizes the number of conflicting edges. If the optimal value of the $k - GCP$ is zero, this means that G has a legal k-coloring. A local search algorithm for the GCP can be used to solve the $k - GCP$ by simply stopping the search as soon as a legal k-coloring is met. Also, an algorithm that solves the $k - GCP$ can be used to solve the GCP, by starting with an upper bound k on $\chi(G)$, and then decreasing k as long as a legal k-coloring can be found.

We now describe three search spaces that we use within a VSS to solve the GCP and the $k - GCP$. A solution to the $k - GCP$ must satisfy two constraints: no edge can have both endpoints with the same color, and all vertices must be colored. The first two considered search spaces relax one of the two constraints, while the third one satisfies all of them. More precisely, let

S_1 denote the set of all (non-necessarily legal) k-colorings of G, and let $f_1(s)$ be the number of conflicting edges in a solution $s \in S_1$. For every k-coloring $s \in S_1$, define $N_1(s)$ as the set of k-colorings obtained by changing the color of exactly one vertex in s. The famous TabuCol algorithm [15], developed by Hertz and de Werra in 1987, is a tabu search algorithm for the $k-GCP$, the aim being to minimize f_1 over S_1 using neighborhood N_1. It is a simple, quick and efficient algorithm that is often used as a subroutine in various methods, such as the hybrid evolutionary algorithms in [7], the genetic algorithm in [8], the adaptive memory algorithm in [16], and the VNS in [1].

Instead of relaxing the constraint that the endpoints of an edge should have different colors, one may relax the constraint imposing that all vertices should be colored. In 1996, Morgenstern [22] proposed the following strategy for the solution of the $k-GCP$. He considers the set, which we denote by S_2, of partial legal k-colorings which are defined as legal k-coloring of a subset of vertices of G. Such colorings can be represented by a partition of the vertex set into $k+1$ subsets $V_1, \dots, V_{k+1}$, where $V_1, \dots, V_k$ are k disjoint stable sets (i.e. legal color classes) and V_{k+1} is the set of non-colored vertices. The objective can be to minimize the number of vertices in V_{k+1} or, as suggested by Morgenstern [22], to minimize $f(s) = \sum_{v \in V_{k+1}} d(v)$, where $d(v)$ denotes the number of edges incident to v. A neighbor solution can be obtained by moving a vertex v from V_{k+1} to a color class V_i, and by moving to V_{k+1} each vertex in V_i that is adjacent to v. Such a move is called an i-swap. Recently, Bloechliger and Zufferey [2] have obtained very good results using a reactive tabu search based on this strategy, with $f_2(s)$ being equal to the number of non-colored vertices in $s \in S_2$. We denote by $N_2(s)$ the set containing all solutions in S_2 that can be obtained from s with an i-swap.

For the third search space, there is no fixed number of colors, and we do not relax any constraints. The following definitions are helpful in describing this search space. A *digraph* is a graph with an orientation on each edge. An edge $[u, v]$ oriented from u to v is called an *arc*, is denoted $u \to v$, and u is its *tail* while v is its *head*. An *orientation* of a graph G is a directed graph, denoted $\vec{G}$, obtained from G by choosing an orientation $u \to v$ or $v \to u$ for each edge $[u, v]$ in G. Gallai, Roy and Vitaver [10,24,26] had independently proved in the sixties that the length of a longest path in an orientation of a graph G is at least equal to the chromatic number of G. As a corollary, the problem of orienting the edges of a graph so that the resulting digraph $\vec{G}$ is circuit-free and the length $\lambda(\vec{G})$ of a longest path in $\vec{G}$ is minimum, is equivalent to the problem of finding the chromatic number of G. Indeed, given a $\chi(G)$-coloring c of a graph G, one can easily construct a circuit-free orientation $\vec{G}$ with $\lambda(\vec{G}) \le \chi(G)$ by simply orienting each edge $[u, v]$ from u to v if and only if $c(u) < c(v)$. Conversely, given a circuit-free orientation $\vec{G}$ of G, one can build a $\lambda(\vec{G})$-coloring of G by assigning to each vertex v a color $c(v)$ equal to the length of a longest path ending at v in $\vec{G}$. Such an equivalence has recently been analyzed in [12] in the context of a local search. More precisely Gendron, Hertz and St-Louis propose to define the search space S_3 as the set containing all circuit-free graph orientations $\vec{G}$ of G, the objective being to minimize $f_3(\vec{G}) = \lambda(\vec{G})$. They propose several neighborhoods including the following one. Given a solution $\vec{G} \in S_3$, let $\vec{G}_\lambda$ denote the digraph obtained by removing all arcs that do not belong to a longest path in $\vec{G}$. A neighbor of $\vec{G}$ can be obtained by choosing a vertex x and changing the orientation of all arcs with head x in $\vec{G}_\lambda$, or of all arcs with tail x in $\vec{G}_\lambda$. It is proved in [12] that such a move does not create any circuit, and increases the length of a longest path by at most one unit. We will use this neighborhood, denoted N_3, to minimize f_3 over S_3.

We now describe how we translate a solution from S_i to a solution in S_j with $i \neq j$. Translator T_{12} builds a legal partial k-coloring in S_2 from a k-coloring in S_1 by randomly choosing an endpoint of each conflicting edge, and inserting these chosen vertices into V_{k+1}. Translator T_{21} builds a k-coloring in S_1 from a legal partial k-coloring in S_2 by considering the vertices in V_{k+1} one by one, in a random order, and giving to each of them the color in $\{1, \dots, k\}$ that creates the smallest number of conflicting edges.

Translators T_{13} and T_{23} build an orientation $\vec{G} \in S_3$ from a solution in S_1, S_2 by labeling the vertices of $G = (V, E)$ from 1 to $|V|$, and then by considering every pair of adjacent vertices $x \in V_i$ and $y \in V_j$, and orienting $[x, y]$ from x to y if and only if $i < j$, or $i = j$ and the label of x is smaller than the label of y.

Finally, given any solution in S_3, let V_i be the set of vertices x such that the longest path ending at x contains i vertices. Translator T_{31} builds a k-coloring in S_1 by giving color i to every vertex in V_i, with $1 \le i \le k$, and then coloring the remaining vertices sequentially, in a random order, each one receiving a color in $\{1, \dots, k\}$ that creates the smallest number of conflicting edges. Translator T_{32} first relabels the indices of the sets V_i so that $|V_i| \ge |V_j|$ whenever $i < j$. Then all sets $V_{k+1}, \dots, V_{\lambda(\vec{G})}$ are merged into one set V_{k+1}.

4. Complementariness of the search spaces

In this section we demonstrate the usefulness of each search space by giving examples where a strict local but not global optimum s according to N_i and f_i in S_i (i.e., a solution $s \in S_i$ that is not optimal while $f_i(s) < f_i(s')$ for all $s' \in N_i(s)$) can be translated into a solution $s' \in S_j$ with $i \neq j$ such that s' can be transformed into an optimal solution in S_j using N_j, and without increasing f_j. We however do not claim that given any graph G, a local optimum s in some search space S_i can always be translated into a solution s' in another search space S_j $(j \neq i)$ where s' can be transformed into a global optimum without increasing f_j.

The numbers inside the vertices in the following figures refer to colors (hence vertices in V_{k+1} have no number), and bold edges represent conflicting edges. The top graph of Fig. 1 is a 2-coloring $s \in S_1$ with $f_1(s) = 4$ conflicting edges. All neighbors $s' \in N_1(s)$ have $f_1(s') = 5$ conflicting edges, which proves that s is a strict local optimum in S_1. The four possible translations (up to symmetry) obtained using T_{12} are represented at the bottom of Fig. 1. They all have 4 non-colored vertices.

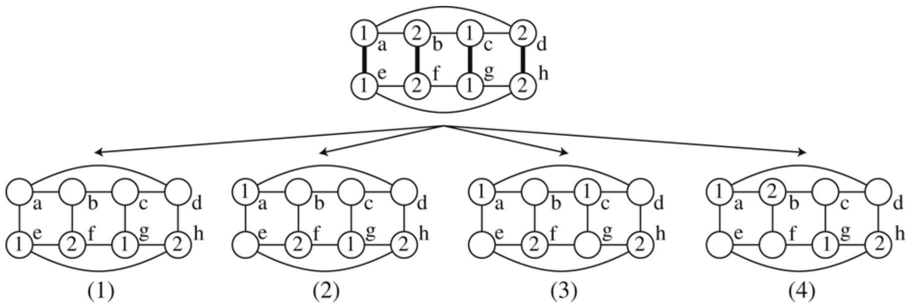

Fig. 1. $S_1 \to S_2$.

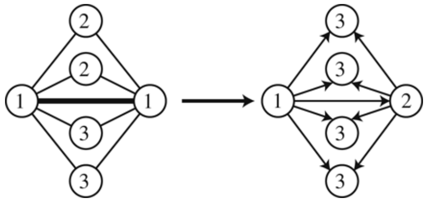

Fig. 2. $S_1 \to S_3$.

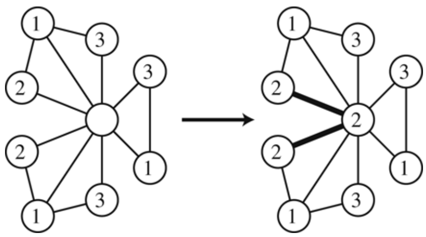

Fig. 3. $S_2 \to S_1$.

- In case (1), the graph can be transformed into a legal 2-coloring by successively coloring a, b, c and d, decreasing f_2 from 4 to 0.
- In case (2), vertex c is not adjacent to any vertex of color 2, and a neighbor s_1 with $f_2(s_1) = 3$ can therefore be obtained by giving color 2 to c. Then all 3 non-colored vertices b, d and e have only one neighbor (vertex a) with color 1 and two with color 2. Color 1 is therefore assigned to one of them, while the color on a is removed. The resulting graph is a neighbor $s_2 \in N_2(s_1)$ with $f_2(s_2) = 3$. Finally, s_2 can be transformed into a legal 2-coloring of G by successively coloring the three non-colored vertices, decreasing f_2 from 3 to 0.
- In case (3), all non-colored vertices are adjacent to one vertex with one of the colors in $\{1, 2\}$ and to two vertices with the other color. Without loss of generality, one may assume that color 1 is assigned to vertex g while the color on c is removed. The resulting solution s' has $f_2(s') = 4$, and it corresponds to case (2) for which we have already shown how to get a legal 2-coloring without increasing f_2.
- In case (4), all non-colored vertices are adjacent to one vertex with color 1 and to one vertex with color 2. Without loss of generality, one may assume that color 2 is assigned to vertex f, while color 2 is removed from b, and we are again in case (2).

The left graph of Fig. 2 is a 3-coloring s with $f_1(s) = 1$ conflicting edge. Since all solutions $s' \in N_1(s)$ are obtained by changing the color of one of the vertices with color 1, they all have $f_1(s') = 2$ conflicting edges. Solution s is therefore a local optimum in S_1. The right graph of Fig. 2 is the translation of s obtained using T_{13} and corresponds to a legal 3-coloring.

The left graph of Fig. 3 is a legal partial 3-coloring s with $f_2(s) = 1$ non-colored vertex. Since the non-colored vertex is adjacent to two vertices with color 2, and three vertices with colors 1 and 3, all neighbors $s' \in N_2(s)$ will have at least two non-colored vertices, which means that s is a strict local optimum in S_2. The right graph of Fig. 3 is the translation s' of s obtained using T_{21}. It contains $f_1(s') = 2$ conflicting edges which can be removed by assigning color 3 to the non-common endpoints of these two edges.

The left graph of Fig. 4 is a legal partial 3-coloring s with $f_2(s) = 1$ non-colored vertex. Since the non-colored vertex is adjacent to two vertices with colors 1, 2 and 3, all neighbors $s' \in N_2(s)$ will have $f_2(s') = 2$, which means that s is a strict local optimum in S_2. The right graph of Fig. 4 is the translation of s obtained using T_{23} and corresponds to a legal 3-coloring.

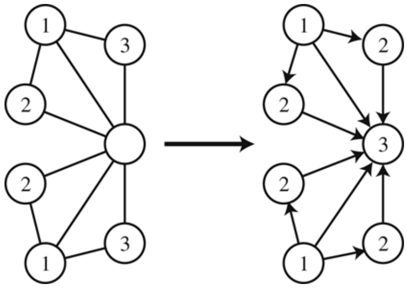

Fig. 4. $S_2 \to S_3$.

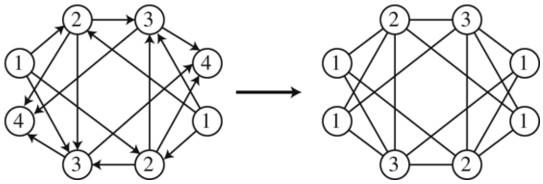

Fig. 5. $S_3 \to S_1$.

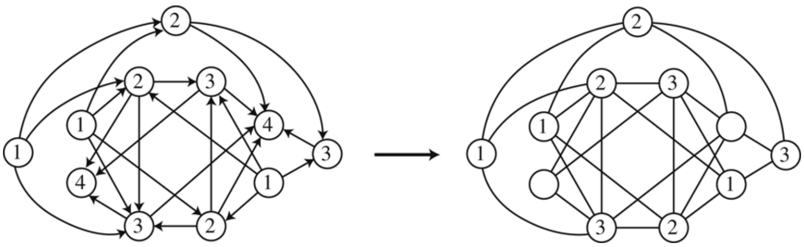

Fig. 6. $S_3 \to S_2$.

The left graph of Fig. 5 is a local optimum $s \in S_3$ with $f_3(s) = 4$ since it can easily be verified that all neighbors $s' \in N_3(s)$ have a longest path with $f_3(s') = 5$ vertices. The right graph of Fig. 5 is the translation of s obtained using T_{31} and corresponds to a legal 3-coloring.

Finally, it can be checked that the left graph of Fig. 6 is a local optimum $s \in S_3$ with $f_3(s) = 4$ since all neighbors $s' \in N_3(s)$ have a longest path with $f_3(s') = 5$ vertices. The right graph of Fig. 6 is the translation of s obtained using T_{32}. It contains two non-colored vertices to which color 1 can be assigned to get a legal 3-coloring, and thus decreasing f_2 from 2 to 0.

5. VSS for graph coloring

We now show how we have adapted VSS to solve the $k - GCP$. After some preliminary experiments, we have found that the sequence $S_1 \to S_3 \to S_2 \to S_1$ of search spaces, called a *cycle*, appears as a good choice, each translation from an S_i to its successor being easy to perform.

The first search space we use is S_1 with neighborhood N_1 and objective function f_1, the aim being to determine a legal k-coloring of a graph G with a fixed k. We have implemented the tabu search algorithm TabuCol described in [15]. The tabu list contains pairs (v, c) with the meaning that it is forbidden for some iterations to assign color c to v. A move from a solution s to a neighbor $s' \in N_1(s)$ consists in changing the current color c_1 of a vertex v for a new color c_2, where v is the endpoint of at least one conflicting edge. When such a move is performed, the pair (v, c_1) is introduced in the tabu list. As proposed in [8], the pair (v, c_1) is considered as tabu for $0.6 \cdot n_c + RANDOM(0, 9)$, where n_c is the number of conflicting vertices in the current solution, and $RANDOM(0, 9)$ is a function providing a random integer in $\{0, 1, \ldots, 9\}$. TabuCol is applied until I_T iterations have been performed without improvement of the best encountered solution (where I_T is a parameter). Let s_{TC} be the resulting solution.

We then remove the conflicting edges in s_{TC} from G to get a legal k-coloring of a partial subgraph G' of G, and translate the legal k-coloring of G' into an orientation $\vec{G}'$ of G', using T_{13}. Notice that $\lambda(\vec{G}') \leq k$, the inequality being possibly strict. For example, the left graph of Fig. 7 has one conflicting edge, and by removing it and translating the legal 3-coloring of the resulting partial subgraph G' of G, using T_{13}, one gets an orientation $\vec{G}'$ with $\lambda(\vec{G}') = 2$.

As shown in [12], a local search in S_3 using neighborhood N_3 and objective function f_3 is rather slow, and not competitive with other local search coloring algorithms. It can however be very useful in changing the color of many vertices simultaneously, and constitutes therefore an interesting diversification strategy. For this purpose, we randomly choose an endpoint v for each conflicting edge in s_{TC}, and either inverse the orientation of all arcs with head v in $\vec{G'}_\lambda$, or of all arcs with tail v in $\vec{G'}_\lambda$, the choice being random. We then modify the resulting orientation by randomly generating neighbors using N_3, until at least M_A arcs have been inversed (where M_A is a parameter). Finally, we sequentially reinsert the edges which have been removed from G, giving to each of them the orientation that minimizes the length of the longest path. Let s_{OR} be the resulting solution.

We then translate s_{OR} into a partial legal k-coloring using T_{32} and use the tabu search algorithm PartialCol, proposed in [2], to improve the solution using neighborhood N_2 and objective function f_2. As mentioned in Section 3, a neighbor $s' \in N_2(s)$ of a solution in $s \in S_2$ is obtained by moving a vertex v from V_{k+1} to a color class V_i ($1 \leq i \leq k$), and by moving to V_{k+1} each vertex in V_i that is adjacent to v. When performing such a move, vertex v is introduced in the tabu list to prevent its reinsertion into V_{k+1}. As proposed in [2], a vertex is considered as tabu for $0.6 \cdot n_c + RANDOM(0, 9)$, where n_c is the number of vertices in V_{k+1} in the current solution. Let s_{PC} be the resulting solution.

We finally translate s_{PC} into a (non-necessarily legal) k-coloring using T_{21}, and start a new cycle with TabuCol. We stop the algorithm when a time limit T_{MAX} is reached. Fig. 8 shows the global scheme of the proposed algorithm.

Notice that the search spaces do not play the same role. It has been demonstrated that while TabuCol is an efficient algorithm, it can have difficulties in exploring all regions of S_1. The moves in S_3 aim to diversify the search by inversing the orientation of many arcs on longest paths, and thus changing the color of many vertices without deteriorating too much the quality of the solution. The aim of PartialCol is to quickly reduce the number of uncolored vertices after having translated the resulting solution in S_3 into a partial legal k-coloring. TabuCol can then restart a new search from a solution that belongs hopefully to a region of S_1 that has not yet been explored. The pseudo-code of VSS-Col is shown in Algorithm 2. It uses the four parameters I_T, I_P, M_A and T_{MAX}.

Algorithm 2 VSS-Col

Require: A graph G and a number k of colors;
  Generate an initial k-coloring $s^1_{\text{init}} \in S_1$;
  while no legal k-coloring of G is found and T_{MAX} is not reached **do**

    (Search in S_1)
    Apply TabuCol starting from s^1_{init}, until I_T iterations have been performed without improvement of the best encountered solution; let s_{TC} be the resulting solution;

    (Translation T_{13})
    Remove the conflicting edges in s_{TC} from G to get a legal k-coloring of a partial subgraph G' of G, and translate the coloring of G' into an orientation $s^3_{\text{init}} \in S_3$, using T_{13};

    (Search in S_3)
    Randomly choose an endpoint v for each conflicting edge in s_{TC} and either inverse the orientation of all arcs with head v in $\vec{G'}_\lambda$, or of all arcs with tail v in $\vec{G'}_\lambda$, the choice being random;
    Modify the resulting orientation by randomly generating neighbors using N_3, until at least M_A have been modified in s^3_{init};
    Sequentially reinsert the edges which have been removed from G, giving to each of them the orientation that minimizes f_3, and let s_{OR} be the resulting solution;

    (Translation T_{32})
    Translate s_{OR} into a legal partial k-coloring $s^2_{\text{init}} \in S_2$, using T_{32};

    (Search in S_2)
    Apply PartialCol starting from s^2_{init}, until I_P iterations have been performed without improvement of the best encountered solution; let s_{PC} be the resulting solution;

    (Translation T_{21})
    Translate s_{OR} into a k-coloring $s^1_{\text{init}} \in S_1$, using T_{21};
  end while

6. Results

Our algorithm was implemented in C++ and run on a 2 GHz Pentium 4 with 512 MB of RAM. We made two series of tests with two maximal computational times, the first one with T_{MAX} equal to 1 h, and the second one with a 10 h limit. After some preliminary experiments, we decided to fix the values of the parameters as follows. When the time limit is 1 h, we use $I_T = 100{,}000$, $I_P = 20{,}000$ and $M_A = 10$ for graphs with at most 500 vertices, and $I_T = 200{,}000$, $I_P = 20{,}000$ and $M_A = 20$ for larger graphs. Moreover if the graph has a density smaller or equal to 0.1, we multiply I_P by 50 and divide M_A by 2, because

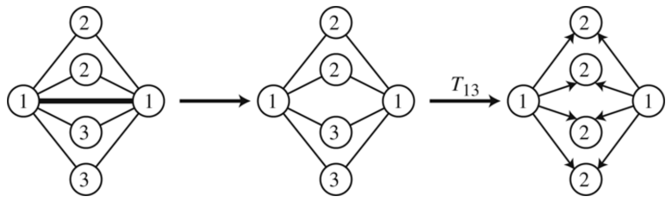

Fig. 7. Illustration of a transformation used in VSS-Col.

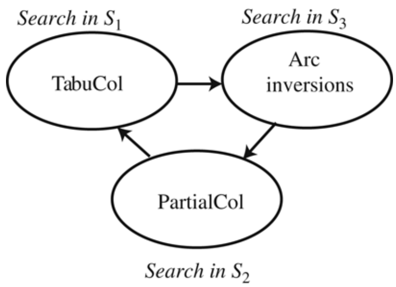

Fig. 8. Cyclic scheme of the VSS algorithm for the $k - GCP$.

too many changes in S_3 tend to create solutions in S_2 with large values of $|V_{k+1}|$. When the time limit is 10 h, we multiply the above values of I_T and I_P by 10. It is probably possible to choose a better setting of parameters for each graph, but our goal is to have generic parameters which use only general characteristics of the graphs, and not to propose a specific set of parameters for each instance.

We ran our algorithm on 16 graphs from the DIMACS Challenge [6]. After a preliminary set of experiments, and in adequation with the literature (e.g. [2,16]), we selected those graphs because they are the most challenging ones. The considered graphs are described below.

- Six DSJC*n.d* graphs: the DSJCs are random graphs with n vertices and a density of $\frac{d}{10}$. It means that each pair of vertices has a probability of $\frac{d}{10}$ to be adjacent. We use the DSJC graphs with $n \in \{500, 1000\}$ and $d \in \{1, 5, 9\}$.
- Two DSJR*n.r* graphs: the DSJRs are geometric random graphs. They are constructed by choosing n random points in the unit square and two vertices are connected if they are distant by less than $\frac{r}{10}$. Graphs with an added end letter 'c' are the complementary graphs. We use two graphs with $n = 500$ and, respectively, $r = 1$ and $r = 5$.
- Four flat$n$_χ_0 graphs: the flat graphs are constructed graphs with n vertices and a chromatic number χ. The end number '0' means that all vertices are incident to the same number of vertices.
- Four le$n$_χx graphs: the Leighton graphs are graphs with n vertices and a chromatic number χ equal to the size of the largest clique (i.e., the largest number of pairwise adjacent vertices). The end letter 'x' stands for different graphs with similar settings.

We first report the results obtained by using VSS-Col on these 16 graphs, and then compare our algorithm with TabuCol [15], PartialCol [2], as well as with three graph coloring algorithms which are among the most effective ones today: the GH algorithm in [8], the MOR algorithm in [22], and the MMT algorithm in [18]. GH, MOR and MMT are all hybrid evolutionary algorithms. GH uses TabuCol to improve offspring solutions, whereas MMT uses a procedure close to PartialCol. MOR works in the same search space S_2 as PartialCol, but uses Simulated Annealing instead of Tabu Search, and more complicated moves than i-swaps.

Table 1 reports the results obtained with VSS-Col with a time limit of one hour. The first column indicates the name of the graph, and the second column contains two numbers, the first one being the chromatic number (we put a "?" when it is not known), and the second one the best known upper bound. We ran VSS-Col 10 times on each graph with different values of k. The third column reports various values of k ranging from the smallest number for which we had at least one successful run, to the smallest number for which we had 10 successful runs. The next columns respectively contain the number of successful runs and the number of tries, the average number of iterations in thousands (i.e., the total number of moves performed using the 3 neighborhoods, divided by 1000) on successful runs, the average number of cycles made by the algorithm, and the average CPU-time used (in seconds).

We observe that on five graphs (namely DSJC500.1, flat1000_50_0, flat1000_60_0 and the two le450_15 graphs), we find a k-coloring on every run, with k equal to the chromatic number or the best known upper bound. On four other graphs (namely DSJC1000.1, DSJC500.9, DSJC500.5 and DSJR500.1c), we reach the best known solutions in at least one run.

Tables 2 and 3 give the same information as for VSS-Col, but for TabuCol and PartialCol. They are taken from [2] where all experiments have been performed on the same computer, with the same compilation options and the same time limit, and with 50 runs for every graph and value of k. We only show results for the smallest k with which at least one of the 50 runs was successful, and for all other larger values of k that also appear in Table 1.

Table 1
Detailed results of VSS-Col with a time limit of 1 h

Graph	χ, $k^\star$	k	Succ./run	10^3 iter	Cycles	time
DSJC1000.1	?, 20	20	3/10	285,624	211	2396
		21	10/10	757	1	11
DSJC1000.5	?, 83	88	8/10	55,971	229	2028
		89	10/10	22,852	91	820
DSJC1000.9	?, 224	224	1/10	48,348	209	3326
		225	5/10	21,667	90	1484
		226	10/10	27,429	116	1751
DSJC500.1	?, 12	12	10/10	19,799	17	97
DSJC500.5	?, 48	48	3/10	78,667	622	1331
		49	10/10	10,524	82	162
DSJC500.9	?, 126	126	8/10	76,927	623	1686
		127	10/10	7,754	62	169
DSJR500.1c	?, 85	85	9/10	48,530	397	736
		86	10/10	20,020	165	291
DSJR500.5	?, 122	126	9/10	61,849	409	1409
		127	10/10	9,066	60	183
flat1000_50_0	50, 50	50	10/10	625	1	318
flat1000_60_0	60, 60	60	10/10	1,242	2	694
flat1000_76_0	76, 82	87	4/6	48,609	199	1689
		88	10/10	36,924	150	1155
flat300_28_0	28, 28	29	1/10	45,611	296	867
		30	2/10	21,7647	1724	2666
		31	10/10	4,173	32	39
le450_15c	15, 15	15	10/10	497	4	6
le450_15d	15, 15	15	10/10	4,761	39	44
le450_25c	25, 25	26	10/10	183	1	1
le450_25d	25, 25	26	10/10	117	1	1

Table 2
Detailed results for TabuCol with a time limit of 1 h

Graph	χ, $k^\star$	k	Succ./run	10^3 iter	Time
DSJC1000.1	?, 20	20	14/50	224,021	1855
		21	50/50	161	1
DSJC1000.5	?, 83	89	48/50	17,482	1224
DSJC1000.9	?, 224	227	48/50	7,198	1520
DSJC500.1	?, 12	12	50/50	8,878	48
DSJC500.5	?, 48	49	11/50	69,803	1550
DSJC500.9	?, 126	127	50/50	7,198	328
DSJR500.1c	?, 85	85	1/50	55,458	685
DSJR500.5	?, 122	126	5/50	56,818	746
		127	12/50	10,387	154
flat1000_50_0	50, 50	50	50/50	732	421
flat1000_60_0	60, 60	60	49/50	2,099	1415
flat1000_76_0	76, 82	88	46/50	16,532	1173
flat300_28_0	28, 28	31	50/50	32,521	378
le450_15c	15, 15	16	50/50	847	4
le450_15d	15, 15	15	1/50	2,246	12
le450_25c	25, 25	26	49/50	954	9
le450_25d	25, 25	26	50/50	1,313	12

We observe that VSS-Col finds better colorings than TabuCol on seven graphs (namely DSJC1000.5, DSJC1000.9, DSJC500.5, DSJC500.9, flat1000_76_0, flat300_28_0 and le450_15c). On three other graphs (namely DSJR500.1c, DSJR500.5 le450_15d), VSS-Col and TabuCol find solutions of the same quality, but VSS-Col has a better success rate. Both algorithms find the same number of colors, with the same success rate, on the six remaining graphs, but TabuCol is faster than VSS-Col on DSJC500.1 and DSJC1000.1, while VSS-Col is faster than TabuCol on the four other graphs (namely flat1000_50_0, flat1000_60_0, le450_25c and le450_25d). VSS-Col can therefore clearly be considered as more effective than TabuCol.

PartialCol finds a legal 28-coloring on flat300_28_0 whereas VSS-Col can only find a legal 29-coloring. On seven other graphs (namely DSJC1000.5, DSJC1000.9, DSJC500.5, DSJC500.9, flat1000_76_0, le450_25c and le450_25d) VSS-Col finds better solutions than PartialCol. On DSJC1000.1, DSJR500.1c and DSJR500.5, VSS-Col has better success rates than PartialCol, and on DSJC500.1, VSS-Col is faster than PartialCol. The four remaining graphs (namely flat1000_50_0, flat1000_60_0, le450_15c and le450_15d) are solved in a very short time by both algorithms, while PartialCol is a little bit faster than VSS-Col. Although PartialCol finds a better coloring on one graph, we can say that VSS-Col outperforms PartialCol.

In Table 4, we compare VSS-Col with TabuCol, PartialCol, GH, MMT and MOR. For every algorithm, we report the smallest k with which a legal k-coloring could be found. The results for GH, MMT and MOR are taken from [2]. Comparisons must

Table 3
Detailed results for PartialCol with a time limit of 1 h

Graph	χ, $k^\star$	k	Succ./runs	10^3 iter	Time
DSJC1000.1	?, 20	20	3/50	292,947	2301
		21	50/50	277	2
DSJC1000.5	?, 83	89	6/50	45,502	2786
DSJC1000.9	?, 224	228	30/50	14,826	2275
DSJC500.1	?, 12	12	50/50	38,819	193
DSJC500.5	?, 48	49	1/50	55,679	811
DSJC500.9	?, 126	127	1/50	43,409	1680
DSJR500.1c	?, 85	85	3/50	56,980	989
DSJR500.5	?, 122	126	28/50	79,620	1544
		127	44/50	34,271	631
flat1000_50_0	50, 50	50	50/50	107	26
flat1000_60_0	60, 60	60	50/50	390	91
flat1000_76_0	76, 82	88	9/50	40,543	2376
flat300_28_0	28, 28	28	13/50	154,261	1878
		29	35/50	133,092	1398
		30	46/50	131,767	1221
		31	49/50	79,871	652
le450_15c	15, 15	15	50/50	615	3
le450_15d	15, 15	15	50/50	4,682	22
le450_25c	25, 25	27	50/50	1,583	10
le450_25d	25, 25	27	50/50	1,151	7

Table 4
Comparisons between VSS-Col and five other algorithms

Graph	χ, $k^\star$	VSS-Col	TabuCol	PartialCol	GH	MMT	MOR
DSJC1000.1	?, 20	20	20	20	20	20	21
DSJC1000.5	?, 83	88	89	89	83	83	88
DSJC1000.9	?, 224	224	227	228	224	226	226
DSJC500.1	?, 12	12	12	12	12	12	12
DSJC500.5	?, 48	48	49	49	48	48	49
DSJC500.9	?, 126	126	127	127	126	127	128
DSJR500.1c	?, 85	85	85	85	–	85	85
DSJR500.5	?, 122	126	126	126	–	122	123
flat1000_50_0	50, 50	50	50	50	50	50	50
flat1000_60_0	60, 60	60	60	60	60	60	60
flat1000_76_0	76, 82	87	88	88	83	82	89
flat300_28_0	28, 28	29	31	28	31	31	31
le450_15c	15, 15	15	16	15	15	15	15
le450_15d	15, 15	15	15	15	15	15	15
le450_25c	25, 25	26	26	27	26	25	25
le450_25d	25, 25	26	26	27	26	25	25

therefore be done carefully because the conditions of experimentation are not the same. For example, our algorithm has a 1 h time limit, while MMT uses a limit of 100 min. In addition, the performances of the computers could be different, and contrary to GH and MMT, we do not adjust the parameters of VSS-Col on each instance. We can observe that

- VSS-Col is better than GH on flat300_28_0 and worse on DSJC1000.5 and flat1000_76_0.
- VSS-Col is better than MMT on three graphs (namely DSJC1000.9, DSJC500.9 and flat300_28_0) and worse on five (namely DSJC1000.5, DSJR500.5, flat1000_76_0, le450_25c and le450_25d).
- VSS-Col is better than MOR on six graphs (namely DSJC1000.1, DSJC1000.9, DSJC500.5, DSJC500.9, flat1000_76_0 and flat300_28_0), and worse on three graphs (namely on DSJR500.5, le450_25c and le450_25d).

While local search algorithm can usually hardly compete with hybrid evolutionary algorithms in terms of solution quality, we observe from Table 4 that VSS-Col produces, in one hour, results which are competitive with the currently most efficient graph coloring algorithms.

In Table 5, we finally report some results with a time limit of 10 h. We only report results for graphs for which VSS-Col could find better colorings when compared to Table 1. We observe that VSS-Col has determined a legal 28-coloring of flat300_28_0, as PartialCol did within one hour. Also, we now match the chromatic numbers of le450_25c and le450_25d. The results for DSJC1000.5, DSJR500.5 and flat1000_76_0 have been improved but are still worse than those obtained by GH or MMT. For the other 10 graphs, we have improved the success rate but not reduced the number of colors.

Table 5
Results for VSS-Col with a time limit of 10 h

Graph	$\|V\|$	χ, $k^\star$	Succ./run	k	10^3 iter	Cycles	Time
DSJC1000.5	1000	?, 83	8/10	87	271,144	117	11,087
			2/10	86	273,944	118	12,728
DSJR500.5	500	?, 122	10/10	125	722,310	526	14,300
flat1000_76_0	1000	76, 82	9/10	86	460,465	199	17,692
			2/10	85	544, 600	235	24,413
flat300_28_0	300	28, 28	2/10	28	363,230	203	9,623
le450_25c	450	25, 25	9/10	25	1623,177	1105	16,429
le450_25d	450	25, 25	6/10	25	2198,280	1528	23,012

7. Conclusion

We have proposed a new general optimization methodology called Variable Space Search, that uses various search spaces, neighborhoods and objective functions, and combines them in a single algorithm. We have also presented an adaptation of the Variable Space Search to the $k - GCP$. The computational experiments, carried out on a set of challenging DIMACS graphs [6], show that VSS-Col is more effective than TabuCol and PartialCol which are local search algorithms used in VSS-Col, but working in a single search space. VSS-Col appears to be also competitive with and a good alternative to the current best hybrid evolutionary graph coloring algorithms.

Notice that the Variable Space Search can support more than one neighborhood within each search space. For example, the search we made in S_1 with TabuCol could be replaced by a VNS in S_1, using for example the algorithm proposed in [1]. Also, the search in S_2 could combine the i-swaps of PartialCol with more elaborated neighborhoods designed in [22], and the search in S_3 could use the four different neighborhoods defined in [12]. While we keep this for future work, we think we have demonstrated that VSS is a simple and effective strategy to improve on complementary local search methods for the same problem.

It is important to notice that the search spaces do not need to contain the same type of solutions. Relaxed constraints in a search space can be imposed in another one. This is therefore an extension to the Reformulation Descent proposed in [21]. In conclusion, we think that the VSS methodology is a new interesting and challenging approach for the solution of complex optimization problems.

References

[1] C. Avanthay, A. Hertz, N. Zufferey, A variable neighborhood search for graph coloring, European Journal of Operational Research 151 (2003) 379–388.
[2] I. Bloechliger, N. Zufferey, A graph coloring heuristic using partial solutions and a reactive tabu scheme, Computers & Operations Research 35 (2008) 960–975.
[3] J.R. Brown, Chromatic scheduling and the chromatic number problem, Management Science 19/4 (1972) 456–463.
[4] C. Desrosiers, P. Galinier, A. Hertz, Efficient algorithms for finding critical subgraphs. Discrete Applied mathematics, 2005. (in press).
[5] I.M. Diaz, P. Zabala, A branch-and-cut algorithm for graph coloring, in: Proceedings of the Computational Symposium on Graph Coloring and its Generalization, Ithaca, New York, 2002.
[6] DIMACS Website. ftp://dimacs.rutgers.edu/pub/challenge/graph/.
[7] C. Fleurent, J.A. Ferland, Genetic and hybrid algorithms for graph coloring, Annals of Operations Research 63 (3) (1996) 437–461.
[8] P. Galinier, J.K. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization 3 (4) (1999) 379–397.
[9] P. Galinier, A. Hertz, A survey of local search methods for graph coloring, Computers & Operations Research 33 (2006) 2547–2562.
[10] T. Gallai, On directed paths and circuits, in: P. Erdös, G. Katobna (Eds.), Theory of Graphs, Academic Press, New York, 1968, pp. 115–118.
[11] M. Garey, D.S. Johnson, Computer and Intractability, San Francisco, Freeman, 1979.
[12] B. Gendron, A. Hertz, P. St-Louis, On edge orienting methods for graph coloring, Journal of Combinatorial Optimization 13 (2007) 163–178.
[13] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, 1997.
[14] F. Herrmann, A. Hertz, Finding the chromatic number by means of critical graphs, ACM Journal of Experimental Algorithmics 7/10 (2002) 1–9.
[15] A. Hertz, D. de Werra, Using Tabu search techniques for graph coloring, Computing 39 (1987) 345–351.
[16] A. Hertz, P. Galinier, N. Zufferey, An adaptive memory algorithm for the k-coloring problem, Discrete Applied Mathematics 156 (2008) 267–279.
[17] M. Kubale, B. Jackowski, A generalized implicit enumeration algorithm for graph coloring, Communications of the ACM 28/4 (1985) 412–418.
[18] E. Malaguti, M. Monaci, P. Toth, A metaheuristic approach for the vertex coloring problem, Technical Report OR/05/3, University of Bologna, Italy, 2005.
[19] A. Mehrotra, M.A. Trick, A column generation approach for exact graph coloring, INFORMS Journal on Computing 8 (1996) 344–354.
[20] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & Operations Research 24 (1997) 1097–1100.
[21] N. Mladenović, F. Plastria, D. Urošević, Reformulation descent applied to circle packing problems, Computers & Operations Research 32 (2005) 2419–2434.
[22] C. Morgenstern, Distributed coloration neighborhood search, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26 (1996) 335–357.
[23] J. Peemöller, A correction to Brélaz's modification of Brown's coloring algorithm, Communications of the ACM 26/8 (1983) 593–597.
[24] B. Roy, Nombre chromatique et plus longs chemins d'un graphe, Revue AFIRO 1 (1967) 127–132.
[25] M.A. Trick. http://mat.gsia.cmu.edu/color/color.html.
[26] L.M. Vitaver, Determination of minimal coloring of vertices of a graph by means of boolean powers of the incidence matrix, Doklady Akademii Nauk SSSR147 (1962) 758–759 (in Russian).