INFORMS Journal on Computing
Vol. 20, No. 2, Spring 2008, pp. 302–316
ISSN 1091-9856 | EISSN 1526-5528 | 08 | 2002 | 0302

DOI 10.1287/ijoc.1070.0245

A Metaheuristic Approach for the Vertex Coloring Problem

Enrico Malaguti
Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, 40136 Bologna, Italy, enrico.malaguti@unibo.it

Michele Monaci
Dipartimento di Ingegneria dell'Informazione, University of Padova, 35131 Padova, Italy, monaci@dei.unipd.it

Paolo Toth
Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, 40136 Bologna, Italy, paolo.toth@unibo.it

Given an undirected graph $G = (V, E)$, the *vertex coloring problem* (VCP) requires to assign a color to each vertex in such a way that colors on adjacent vertices are different and the number of colors used is minimized. In this paper, we propose a metaheuristic approach for VCP that performs two phases: the first phase is based on an evolutionary algorithm, whereas the second one is a postoptimization phase based on the set covering formulation of the problem. Computational results on a set of DIMACS instances show that the overall algorithm is able to produce high-quality solutions in a reasonable amount of time. For four instances, the proposed algorithm is able to improve the best-known solution while for almost all the remaining instances, it finds the best-known solution in the literature.

Key words: evolutionary algorithm; set covering; heuristics
History: Accepted by Michel Gendreau, Area Editor for Heuristic Search and Learning; received March 2005; revised October 2006; accepted September 2007. Published online in *Articles in Advance* March 12, 2008.

1. Introduction

Given an undirected graph $G = (V, E)$, the *vertex coloring problem* (VCP) requires to assign a color to each vertex in such a way that colors on adjacent vertices are different and the number of colors used is minimized.

Vertex coloring is a well-known NP-hard problem (see Garey and Johnson 1979) with real-world applications in many engineering fields, including scheduling (Leighton 1979), timetabling (de Werra 1985), register allocation (Chow and Hennessy 1990), frequency assignment (Gamst 1986), and communication networks (Woo et al. 2002). This suggests that effective algorithms would be of great importance. Despite VCP's relevance, few exact algorithms have been proposed and are able to solve consistently only small instances (see Sager and Lin 1991, Mehrotra and Trick 1996, Sewell 1996, Diaz and Zabala 2002 for computational results on benchmark instances). On the other hand, several heuristic and metaheuristic algorithms have been proposed that are able to deal with graphs of hundreds or thousands of vertices. We review below, after some useful definitions, the most important classes of known heuristics and metaheuristics proposed for VCP.

Let n and m be the cardinalities of vertex set V and edge set E, respectively; let $\delta(v)$ be the degree of a given vertex v. A subset of V is called an *independent set* if no two adjacent vertices belong to it. A *clique* of a graph G is a complete subgraph of G. A *k coloring* of G is a partition of V into k independent sets. An optimal coloring of G is a k coloring with the smallest possible value of k (the *chromatic number* $\chi(G)$ of G). The *chromatic degree* of a vertex is the number of different colors of its adjacent vertices.

The first approaches to VCP were based on greedy constructive algorithms. These algorithms sequentially color the vertices of the graph, following some rule for choosing the next vertex to color and the color to use. They are generally very fast but produce poor results that can be very sensitive to some input parameter, like the ordering of the vertices. Beyond the simple greedy *sequential* algorithm (SEQ), the best-known techniques are the *maximum saturation degree* (DSATUR) and the *recursive largest first* (RLF) procedures proposed by Brèlaz (1979) and by Leighton (1979), respectively (see §1.2.1 for a short description of these algorithms). Culberson and Luo (1996) proposed the *iterated greedy algorithm* (IG), which can be combined with various techniques. Bollobàs and Thomason (1985) proposed the algorithm MAXIS that recursively selects the maximum independent set from the set of uncolored vertices.

Many effective metaheuristic algorithms have been proposed for VCP. They are mainly based on simulated annealing (Johnson et al. 1991 compared different neighborhoods and presented extensive computational results on random graphs; Morgenstern 1996 proposed a very effective neighborhood search) or tabu search (Hertz and de Werra 1987, Dorne and Hao 1998). Caramia and Dell'Olmo (1999) proposed a local search with priorities rules, inspired from tabu search techniques. Funabiki and Higashino (2000) proposed one of the most effective algorithms for the problem, which combines a tabu search technique with different heuristic procedures, color fixing, and solution recombination in the attempt to expand a feasible partial coloring to a complete coloring. Hybrid algorithms integrating local search and diversification via crossover operators were proposed (Fleurent and Ferland 1996; Galinier and Hao 1999 proposed to combine an effective crossover operator with tabu search), showing that diversification is able to improve the performance of local search.

This paper is organized as follows: In the remaining part of this section, a new two-phase heuristic approach for VCP is presented. Sections 2 and 3 describe the first phase, which is based on an evolutionary algorithm, and the second phase, which is a postoptimization procedure based on the set covering formulation of the problem, respectively. Extensive computational experiments on literature instances are presented in §4. Concluding remarks are discussed in §5.

1.1. The Heuristic Algorithm MMT

The approach we propose performs, in sequence, an initialization step and two optimization phases. In the initialization step, some fast lower bounding procedures and greedy heuristics from the literature are executed to derive a lower and an upper bound (LB and UB, respectively) on the optimal solution value. In the first optimization phase, an evolutionary algorithm is executed. This algorithm works in decision version (i.e., given as input the number k of colors to use, it looks for a k coloring in the graph G), trying to improve on the best solution found by the greedy procedures executed in the initialization step. Sometimes the evolutionary algorithm is able to find a solution that can be proved to be optimal; in any case, this algorithm generally improves the best incumbent solution and, during the search, generates a very large number of independent sets. When optimality of the incumbent solution is not proved, such independent sets are stored in a family $\mathcal{S}'$. The second optimization phase (*column optimization*) considers the set covering problem (SCP) in which each *column* has *cost* equal to one and corresponds to an independent set in $\mathcal{S}'$, and each *row* corresponds to a vertex in V (thus, each feasible solution of SCP corresponds to a feasible solution of VCP). Such SCP is heuristically solved through the Lagrangian heuristic algorithm CFT proposed by Caprara et al. (1999), improving many times the best incumbent solution.

Both optimization phases can be stopped as soon as a solution which is proven to be optimal is found, i.e., if its value is equal to a lower bound for the original problem.

The overall algorithm MMT is structured as follows:

begin
Initialization step
1. compute lower bound LB;
2. compute upper bound UB; if $LB = UB$ **stop**;
3. $\mathcal{S}' := \varnothing$;
Phase 1: Evolutionary algorithm
(*evolutionary generation*)
4. **while** (not time limit)
   apply evolutionary algorithm;
   update UB and $\mathcal{S}'$;
   if $LB = UB$ **stop**
5. **endwhile**;
Phase 2: Column optimization
6. apply heuristic algorithm CFT to the set covering instance corresponding to subfamily $\mathcal{S}'$ with a given time limit (possibly updating UB)

end.

1.2. Initialization Step

We compute a fast lower bound LB as the cardinality of a maximal clique of G obtained by executing a greedy algorithm having time complexity $O(n^2)$. Although this is a straightforward lower bound for the problem, our computational experiments (see §4) show that it can be computed in negligible time and, for some instances, it allows to prove optimality of the best solution found.

1.2.1. Upper Bounding. To derive an initial upper bound UB for the problem, we perform one iteration of the greedy procedures SEQ, DSATUR, and RLF (see Johnson et al. 1991).

SEQ is the simplest greedy algorithm for VCP. Assume that the vertices are labelled $v_1, \ldots, v_n$. Vertex v_1 is assigned to the first color class, and thereafter vertex v_i $(i = 2, \ldots, n)$ is assigned to the lowest indexed color class that contains no vertices adjacent to v_i.

DSATUR (see Brèlaz 1979, Johnson et al. 1991) is similar to SEQ but dynamically chooses the vertex to color next, picking the first vertex that is adjacent to the largest number of distinctly colored vertices (i.e., the vertex with maximum chromatic degree).

The *recursive largest first* (RLF) algorithm (see Leighton 1979, Johnson et al. 1991) colors the vertices, one class at a time, in the following greedy way. Let

C be the next color class to be constructed, V' the set of uncolored vertices that can legally be placed in C, and U the set (initially empty) of uncolored vertices that can not legally be placed in C.

• Choose the first (according to the labelling $v_1, \dots, v_k$) vertex $v \in V'$ that has the maximum number of adjacent vertices in V'. Place v in C and move all the vertices $u \in V'$ that are adjacent to v from V' to U.

• While V' remains nonempty, do the following: choose the first vertex $v \in V'$ that has the maximum number of adjacent vertices in U; add v to C and move all the vertices $u \in V'$ that are adjacent to v from V' to U.

We use algorithms SEQ and DSATUR also during the initialization of the evolutionary algorithm (see §2.2.1).

1.2.2. Complexity. The time complexity of the initialization procedure is analyzed in the following.

• In the implementation of the SEQ algorithm, a data structure of size $O(n^2)$ is used to store, for every vertex v_i and for every color h, if at least one vertex adjacent to v_i has color h. When trying to assign vertex v_i to a color class, the algorithm checks in this data structure if the color is available for v_i. So, the algorithm has to check $O(n)$ colors for n vertices, with a total time complexity of $O(n^2)$. Every time a new vertex is colored, the information stored in the data structure is updated for all its adjacent vertices, with a total complexity of $O(m)$. The overall time complexity of the SEQ algorithm is $O(n^2)$.

• The DSATUR algorithm uses the same data structure, the only difference being that for n times, the vertex with the maximum chromatic degree is picked (with total complexity of $O(n^2)$). The update of the chromatic degree for every vertex is performed together with the update of the data structure, and the corresponding total complexity is $O(m)$. The overall time complexity of the DSATUR algorithm is $O(n^2)$.

• The RLF algorithm chooses every time the next vertex to be colored as the vertex v_i with the maximum number of adjacent vertices (in V' or in U), with a total complexity of $O(n^2)$. For every chosen vertex v_i, all its adjacent vertices v_j are moved in U with a total complexity of $O(m)$. Every time one adjacent vertex v_j is moved, its adjacent vertices are adjacent to one more vertex in U. The update of this information has a total complexity of $O(m^2/n)$ because for every chosen vertex, the algorithm has to retrieve all its adjacent vertices and all the vertices adjacent to them. The overall complexity of the RLF algorithm is $O(m^2/n)$.

2. Phase 1: Evolutionary Algorithm

To find high-quality solutions, our first idea was to use a tabu search procedure, a metaheuristic technique that showed a very good experimental behavior on hard combinatorial optimization problems. The first results were quite encouraging but showed some drawbacks of our approach. In particular, for several instances, the tabu search procedure was unable to explore different regions of the whole solution space. So we decided to use it as a component of a more complex evolutionary algorithm.

2.1. Tabu Search Algorithm

A local search procedure can be seen as the result of three main components: (i) the definition of a solution S; (ii) the solution evaluating function $f(S)$; and (iii) the solution neighborhood $N(S)$. In the simple local search procedures, given a solution S, the algorithm explores its neighborhood $N(S)$ and moves to the best (according to the evaluating function $f(S)$) improving solution $S' \in N(S)$. If a solution S is the best of its neighborhood, i.e., it is a local optimum, the local search algorithm is not able to move and the search is stopped. In tabu search procedures (see Glover and Laguna 1997), to avoid local optimum traps, the algorithm moves to the best solution S' in the neighborhood, even if it is not improving the current solution. To avoid cycling, some attributes of solution S' are stored in a *tabu list*; for a specified number of iterations (the so-called *tabu tenure*), a solution that presents tabu attributes is declared tabu and is not considered, except in the case where it would improve the best incumbent solution (*aspiration criterion*). Most of the tabu search algorithms proposed so far for VCP move between infeasible solutions; i.e., they partition the set V in subsets that are not necessary independent sets, trying to reduce the number of infeasibilities in every subset. Following an idea by Morgenstern (1996), we propose a tabu search procedure which moves between *partial feasible colorings*, i.e., solutions in which each vertex subset is an independent set but not all vertices are assigned to subsets. Morgenstern (1996) defines the *impasse class neighborhood*, a structure used to improve a partial k coloring to a complete coloring of the same value. The *impasse class* requires a target value k for the number of colors to be used. A solution S is a partition of V in $k+1$ color classes $\{V_1, \dots, V_k, V_{k+1}\}$ in which all classes, but possibly the last one, are independent sets. This means that the first k classes constitute a partial feasible k coloring, while all vertices that do not fit in the first k classes are in the last one. Making this last class empty gives a complete feasible k coloring. To move from a solution S to a new solution $S' \in N(S)$, one can randomly choose an uncolored vertex $v \in V_{k+1}$, assign v to a different color class, say h, and move to class $k+1$ all vertices v' in class h that are adjacent to v. This ensures that color class h remains feasible. Class h is chosen by comparing different target classes by means of the evaluating function $f(S)$.

Rather than simply minimizing $|V_{k+1}|$, it seems a better idea to minimize the value:

$$f(S) = \sum_{v \in V_{k+1}} \delta(v). \quad (1)$$

This forces vertices having small degree, which are easier to color, to enter class $k+1$. Morgenstern (1996) uses the impasse class neighborhood together with a procedure for the recombination of the solutions to build a simulated annealing algorithm. We use the same neighborhood within a tabu search approach. At every iteration, we move from a solution S to the best solution $S' \in N(S)$ (even if $f(S) < f(S')$). To avoid cycling, we use the following tabu rule: A vertex v can not take the same color h that it took at least one of the last T iterations; for this purpose, we store in a tabu list the pair (v, h). While pair (v, h) remains in the tabu list, vertex v cannot be assigned to color class h. We also use an *aspiration criterion*: A tabu move can be performed if it improves on the best solution encountered so far. A tabu search algorithm based on the same neighborhood structure was experimented by Blöchliger and Zufferey (2008): differently from our choice to select at random the next vertex to color, in their work, the next vertex to color is selected so that it, entering the best color class, produces the best solution in the neighborhood. This approach increases the size of the neighborhood, reducing at the same time the randomness introduced in the search. Thus, to avoid premature convergence, the authors use an evaluating function that simply minimizes $|V_{k+1}|$.

Our tabu search algorithm takes in input: (i) graph $G(V, E)$; (ii) the target value k for the coloring; (iii) a feasible partial k coloring; (iv) the maximum number L of iterations to be performed; and (v) the tabu tenure T.

If the algorithm solves the problem within L iterations, it gives on output a feasible coloring of value k; otherwise, it gives on output the best scored partial coloring found during the search.

Let S be the current solution and S^* the best incumbent solution. The tabu search algorithm works as follows:

begin
1. initialize a solution $S := \{V_1, \ldots, V_k, V_{k+1}\}$;
2. $S^* := S$;
3. *tabu list* $:= \varnothing$;
4. **for** (*iterations* $= 1$ **to** L)
5. randomly select an uncolored vertex $v \in V_{k+1}$;
6. **for each** $j \in \{1, \ldots, k\}$ (explore the neighborhood of S)
7. $V'_j := V_j \backslash \{w \in V_j : (v, w) \in E\} \cup \{v\}$;
8. $V'_{k+1} := V_{k+1} \backslash \{v\} \cup \{w \in V_j : (v, w) \in E\}$;
9. $S_j := S \backslash \{V_j, V_{k+1}\} \cup \{V'_j, V'_{k+1}\}$
10. **end for**;
11. $h := \arg\min_{j \in \{1,\ldots,k\}} \{f(S_j)$: $(v, j) \notin$ *tabu list* **or** $f(S_j) < f(S^*)\}$;
12. **if** no such h exists **then** $h := \arg\min_{j \in \{1,\ldots,k\}} f(S_j)$;
13. $S := S_h$;
14. insert (v, h) in *tabu list*, (v, h) is tabu for T iterations;
15. **if** $f(S) < f(S^*)$ **then** $S^* := S$;
16. **if** $V_{k+1} = \varnothing$ **then return** S^*
17. **end for**;
18. **return** S^*

end.

At line 11, we try to select the best color class which improves on the best solution so far or does not represent a tabu move. If all moves are tabu, at line 12 we simply select the best color class.

Our tabu search algorithm is very simple and requires as a parameter to be experimentally tuned only the tabu tenure T. At the same time, it has a good experimental behavior as it is often able to find good solutions in very short computing times. Computational experiments showed that the algorithm generally needs a small number of iterations to solve the problem, and when this does not occur, seldom is the algorithm able to solve the problem even if a bigger number of iterations is allowed. This behavior can be explained by the aggressive strategy adopted: We start with a partial feasible coloring and iteratively try to insert uncolored vertices in color classes. If a colored vertex is not conflicting (adjacent) with an uncolored one, its color is not changed, and possibly it will never be changed during the execution of the algorithm. The main drawback of this strategy is that in some cases it is not able to explore different regions of the whole solution space. This can be explained with an example: Suppose that a pair of vertices belonging to different color classes are not conflicting with any of the uncolored vertices nor conflicting each other. Their assignment to a color class will never be changed, and the algorithm will not explore the feasible solutions where the two vertices are in the same class. This consideration suggests that this tabu search scheme could be much more effective if combined with a suitable diversification strategy.

2.1.1. Complexity. The tabu search procedure represents the most time-consuming part of the proposed approach. Actually, millions of tabu search iterations are performed to color hard instances. An iteration is composed by four main operations: (i) the random choice of the vertex v to color, performed in constant time; (ii) the computation of how much it would cost (according to the evaluating function (1)) to insert v in each color class, requiring the retrieval of all the vertices adjacent to v, with an average complexity of $O(m/n)$ ($O(n)$ if the graph is complete); (iii) the choice

of the best color class V_h, which is not tabu with a complexity of $O(k)$; and (iv) the update of the current coloring (i.e., the movement of the vertices adjacent to v, which are in color class V_h to color class V_{k+1}), requiring the retrieval of all the vertices adjacent to v, performed on average in $O(m/n)$ (in $O(n)$ if the graph is complete). Thus, the total time complexity of one tabu search iteration is $O(n)$ in the worst case.

2.2. Evolutionary Diversification

To improve the performance of the tabu search procedure, we use it together with a diversification operator, trying to extend the search to the whole solution space. Diversification is usually used in genetic algorithms, in which a pool of solutions (*population*) is stored during the computation. Solutions in the pool evolve through interactions with other solutions during the *diversification phase* when they mix together (*parent solutions*) to generate new solutions (*offspring*). In addition, they evolve by themselves during the *mutation phase* when, to avoid premature convergence and to preserve diversity, they are randomly perturbed. In general, solutions in the pool are improved by using some local search technique. Every solution is evaluated according to a fitness function so that when new good solutions are generated, the worst solutions can be removed from the population. As shown by Davis (1998), the classical genetic algorithms give poor results for VCP.

A recent development of these algorithms is represented by evolutionary algorithms. In this case, the evolution of the population is obtained by means of two elements: an efficient local search procedure and a specialized *crossover operator*. The crossover operator should be able to create new and potentially good solutions, to be improved through the local search procedure. For this reason, it cannot be a general operator, but it must be designed specifically for the considered problem. In addition, it must be able to transmit interesting properties from parents to new offspring. The main idea behind the use of a specialized crossover operator is that good solutions share part of their structure with optimal ones, and a specialized crossover should be able to identify properties that are meaningful for the problem.

In our algorithm, we start with an initial pool of partial feasible solutions of value k (in the following, simply *solutions*) obtained by using different methods (greedy and tabu search procedures initialized with different parameters). Then, we apply the tabu search algorithm to improve these solutions during the local search phase. We implemented a variation of the specialized crossover operator *greedy partition crossover* proposed by Galinier and Hao (1999) to generate new solutions and diversify the search. Our purpose is to extend a feasible partial k coloring to a complete coloring. The general procedure is summarized as follows: Given a pool of solutions, we randomly choose two parents from the pool and generate an offspring, which is improved by means of the tabu search algorithm described in §2.1 and finally inserted in the pool, deleting the worst parent. After the initialization, the generation-improvement-insertion procedure is iterated until the problem is solved (i.e., a partial feasible solution is extended to a complete solution) or the number of iterations equals a given threshold.

2.2.1. Initialization. We initialize the pool by generating PS initial solutions (partial feasible k colorings). In this phase, it is crucial to start with solutions that are far from each other, thus exploring the whole search space and avoiding premature convergence of the search. For this purpose, we generate the initial pool by using three different algorithms: (i) the sequential algorithm SEQ applied with different random orderings of the vertices to generate the first third of the solutions in the pool; (ii) the maximum saturation degree algorithm DSATUR applied with different random orderings of the vertices to generate the second third of the solutions in the pool; and (iii) the tabu search algorithm applied starting from a dummy solution (all vertices in class $k+1$) to generate the last third of the solutions in the pool (the random choice of the next vertex to color in the tabu search procedure leads the algorithm to obtain different initial partial colorings).

For each application of both SEQ and DSATUR, only the first k color classes are considered (the remaining vertices are inserted in class $k+1$). Every initial solution is improved with tabu search before being inserted in the pool. The use of an offline procedure to compute diversity in the pool confirms that this choice is able to generate a well-diversified pool.

2.2.2. Crossover Operator. Given two parent solutions randomly chosen from the pool, the crossover operator generates an offspring sharing "interesting properties" with the parents. A solution is a partition of the vertices in $k+1$ sets, where the first k are independent sets. It seems reasonable that interesting structures of the parents could be identified in these independent sets (in the following, we will refer to independent sets or color classes). Galinier and Hao (1999) proposed a crossover operator that, given two parents (partition of the vertices in k sets, not necessarily independent), alternatively considers each parent to generate the next color class of the offspring in this way: The color class of maximum cardinality of the considered parent becomes the next color class of the offspring and all the vertices in this color class are deleted from both parents. When k steps are performed, some vertices are possibly unassigned.

These vertices are then assigned to a randomly chosen class. We modified this operator according to our purpose. Indeed, in our case, the offspring must be a (possibly partial) k coloring. Given two parents $S^1 = \{V_1^1, \dots, V_k^1, V_{k+1}^1\}$ and $S^2 = \{V_1^2, \dots, V_k^2, V_{k+1}^2\}$, the crossover operator outputs the offspring $S^3 = \{V_1^3, \dots, V_k^3, V_{k+1}^3\}$ as follows:

begin
1. $CurrentColor := 1$;
2. **while** ($CurrentColor \le k$ **and** $V_1^1 \cup \cdots \cup V_k^1 \cup V_1^2 \cup \cdots \cup V_k^2 \neq \varnothing$)
3. $A := SelectParent()$;
4. $h := \arg\max_{i=1,\dots,k} |V_i^A|$;
5. $V_{CurrentColor}^3 := V_h^A$;
6. remove the vertices of V_h^A from S^1 and S^2;
7. $CurrentColor := Currentcolor + 1$
8. **end while**;
9. **for each** vertex $v \in V \setminus (V_1^3 \cup \cdots \cup V_k^3)$ try to color v in a greedy way (i.e., try to insert v in one of the k color classes $V_1^3, \dots, V_k^3$);
10. $V_{k+1}^3 := V \setminus (V_1^3 \cup \cdots \cup V_k^3)$

end.

Function *SelectParent*( ), which returns the parent chosen to generate the next color class, works as follows:

begin
1. **if** $((V_1^1, \dots, V_k^1) \neq \varnothing$ **and** $(V_1^2, \dots, V_k^2) \neq \varnothing)$ **then**
2. **if** *CurrentColor* is odd **then** $A := 1$ **else** $A := 2$
3. **else if** $(V_1^1, \dots, V_k^1) \neq \varnothing$ **then** $A := 1$ **else** $A := 2$

end.

This function takes into account that one parent can terminate the available colored vertices. In this case, it considers only color classes from the parent who still has colored vertices. When both parents terminate the available colored vertices or the offspring has used k colors, we try to insert each uncolored vertex v in one of the offspring color classes in the following sequential greedy way:

begin
1. **for each** color class $h = 1, \dots, k$
2. **if** $\nexists w \in V_h^3$: $(v, w) \in E$ **then** $V_h^3 := V_h^3 \cup \{v\}$ and **exit**
3. **end for**

end.

2.2.3. Solution Evaluation. The quality (score) of every solution S in the pool is evaluated through the function $f(S)$ defined by Equation (1) and used during the tabu search algorithm. This allows us to compare the solutions and to tune the quality of the pool during the computation.

2.2.4. Pool Update. We say that two solutions are *similar* if they have the same score and the same number of uncolored vertices.

For pool updating, every offspring is first of all improved by means of the tabu search algorithm and then inserted in the pool, substituting the worst parent. If the offspring is similar to one of the parents or to a solution in the pool, with a probability $p_{\texttt{greedy}}$ proportional to the percentage number of colored vertices in the population, we do not insert the offspring in the population, but we insert a completely new greedy solution, avoiding premature convergence. This new solution is built by using a sequential greedy algorithm that gives priority to the vertices that, during the computation, were more often left uncolored (i.e., inserted in class $k+1$). We call this algorithm *priority greedy*. More in detail, we order the vertices according to nonincreasing values of the number of times they were left uncolored in the pool. We locally perturb this ordering: With a probability $p = 0.5$, we swap every vertex with the next one in the ordering and then apply the SEQ algorithm. This perturbation prevents the generation of the same coloring at different calls of the SEQ algorithm. In this way, we build different solutions where vertices that were more often left uncolored are in color classes of low order, while in class $k+1$ we have vertices more often colored during the computation (see how SEQ works).

To summarize, our evolutionary algorithm (EA) takes in input: (i) graph $G(V, E)$; (ii) the target value k for the coloring; (iii) the maximum number L of tabu search iterations between the application of two consecutive crossover operators; (iv) the cardinality of the pool PS ; (v) the tabu tenure T; and (vi) the *timelimit*. The algorithm works as follows:

begin
1. generate the initial *pool* of solutions;
2. **if** $\exists S^h \in pool$: $V_{k+1}^h = \varnothing$ **then stop**;
3. **while** (not *timelimit*)
4. randomly select two solutions S^1 and S^2 from the *pool*;
5. generate $S^3 := Crossover(S^1, S^2)$;
6. **if** $V_{k+1}^3 = \varnothing$ **then stop**;

Improve the offspring
7. $S^3 := TabuSearch(S^3)$;
8. **if** $V_{k+1}^3 = \varnothing$ **then stop**;

Update the pool
9. **if** S^3 is similar to a solution in the *pool* **then** with probability $p_{\texttt{greedy}}$, set $S^3 := PriorityGreedy()$;
10. **if** $V_{k+1}^3 = \varnothing$ **then stop**;
11. insert S^3 in the *pool*, delete the worst parent
12. **end while**

end.

2.3. Evolutionary Algorithm as Part of the Overall Algorithm

As anticipated in the previous section, we apply our evolutionary algorithm (EA) in the first phase of the

overall algorithm MMT. It must be noted that EA works in decision version, while we are approaching the problem from the optimization point of view. In other words, EA requires as input the value k (the number of colors to be used) while we are trying to minimize this value. To solve this problem, we use the information obtained from the initial greedy heuristics: If the current upper bound *UB* for the problem is $k+1$, we apply EA with k as input parameter. If EA solves the problem for the target value k within the given *timelimit*, we apply it again with $k-1$ as input parameter and iterate until EA is unable to solve the problem.

EA is very effective but largely dependent on the input parameters, i.e., L (number of tabu search iterations between two crossover steps) and PS (size of the pool). Computational experiments (see §4.1) show that difficult instances require a longer tabu search phase and the use of a wider population. In general, high-density graphs with many vertices tend to be difficult to color, but there seems to exist no explicit correlation between effective input parameters and some intrinsic property of the graph.

Thus, we implemented a procedure that dynamically modifies these input parameters for every execution of EA, based on the results obtained in the previous execution. Suppose we are solving an instance using k colors: If the algorithm finds a solution within *UpdateLimit* applications of the crossover operator, we consider the instance to be easy, and we do not update the input parameters when trying to solve the same instance using $k-1$ colors. Otherwise, we increase the values of L and PS of Δ_L and Δ_{PS}, respectively. *UpdateLimit* is a parameter dependent on the graph properties (see computational analysis in §4).

3. Phase 2: Column Optimization

If the incumbent solution found in Phase 1 (i.e., during the execution of EA) is not proved to be optimal, a further optimization phase is executed to improve the value of the solution. This phase is based on an integer linear programming (ILP) formulation of VCP and uses a subset of the independent sets found by EA during Phase 1.

A natural ILP model for VCP is the one having a binary variable for each independent set and a constraint for each vertex. This model is often referred to as the *set covering* (or *set partitioning*) formulation. The relevance of this model lays in the fact that it can describe all those problems in which one is required to partition a given set of items into subsets having special features and minimizing the sum of the costs associated with the subsets. This formulation has been used to solve exactly VCP by Mehrotra and Trick (1996), in which a column generation method for implicit optimization of the linear program at each node of a branch-and-bound tree is used.

Let $\mathcal{S}$ be the family of all the independent sets of G. Each independent set (column) $s \in \mathcal{S}$ has associated a binary variable x_s having value one iff all the vertices of s receive the same color. VCP can be formulated as follows:

$$\min \sum_{s \in \mathcal{S}} x_s \tag{2}$$

$$\sum_{s \in \mathcal{S}: i \in s} x_s \geq 1 \quad (i \in V) \tag{3}$$

$$x_s \in \{0, 1\} \quad (s \in \mathcal{S}). \tag{4}$$

The objective function (2) asks to minimize the total number of independent sets (and hence of colors) used. Constraints (3) state that every vertex i in the graph must belong to at least one independent set (i.e., must receive at least one color). Indeed, if a vertex i is assigned to more than one independent set in a feasible solution, it can be removed from all the independent sets but one (in other words, if a vertex is assigned to more than one color, a feasible solution of the same value can be obtained using any one of these colors for the vertex). Finally, constraints (4) impose variables x_s to be binary.

The advantage of the set covering formulation with respect to alternative descriptive formulations is that it avoids symmetries in the solution and its continuous relaxation leads to tighter lower bounds. The main drawbacks are that the number of variables can grow exponentially with the cardinality of vertex set V (even if one is allowed to consider only the *maximal* independent sets in the definition of $\mathcal{S}$) and that SCP is an NP-hard problem whose exact solution could require very large computing times. Our approach to the problem is heuristic in the sense that during Phase 1 we store only a subfamily $\mathcal{S}' \subseteq \mathcal{S}$ of all the independent sets of graph G, and that in Phase 2 we solve the model (Equations (2)–(4)) corresponding to subfamily $\mathcal{S}'$ through a heuristic algorithm from the literature.

In particular, subfamily $\mathcal{S}'$ is defined during the execution of EA in the following way. For each (partial or complete) feasible k coloring found in Phase 1, we can generate k independent sets (columns) for Phase 2. Every independent set (not necessary maximal) is first of all completed to a maximal independent set by using a greedy procedure that simply tries to insert in the set vertices that are currently not in it, following the input order of the vertices. This ordering is perturbed at every call of the procedure; thus, if the procedure is called on the same set different times, it is generally able to complete the set by introducing different vertices and hence obtaining different columns.

Generally, the global number of independent sets (columns) generated in Phase 1 is very large and could ask for excessive memory requirements. Hence, we chose to insert in $\mathcal{S}'$ only the independent sets generated by the initial greedy algorithms and those corresponding to the feasible solutions found by EA as well as those solutions contained in the pool at the beginning and at the end of each iteration of EA. We also insert all the independent sets corresponding to solutions built to preserve diversity during the computation. A hashing technique is used to remove identical columns (see Monaci and Toth 2006 for further details). Computational experiments showed that this choice, that privileges independent sets corresponding to solutions that tend to have high diversity to each other, did not affect the effectiveness of Phase 2 while reducing considerably the computation time and avoiding memory problems.

As to the solution of the corresponding set covering instance, we use the Lagrangian heuristic algorithm CFT proposed by Caprara et al. (1999). This iterative algorithm can handle very large set covering instances, producing good (possibly optimal) solutions within a reasonable amount of computing time. Moreover, algorithm CFT computes an "internal" lower bound (not valid for VCP) on the value of the optimal solution of the corresponding set covering instance, and its execution can be stopped as soon as this lower bound equals the value of the best incumbent solution for VCP. Of course, optimality for SCP does not imply optimality for the original problem, because we do not enumerate all the independent sets of G.

A similar approach has been used to derive effective heuristic algorithms for bin-packing problems in Monaci and Toth (2006), the main difference between the two approaches being the aim of Phase 1. While in Monaci and Toth (2006) the first phase (column generation phase) was mainly aimed at generating "good" columns for the second phase and the effectiveness of the approach was essentially due to the second phase (column optimization phase), in the current algorithm the first phase is crucial for its effectiveness. This aspect is stressed by the computational results (see §4), showing that the evolutionary algorithm of Phase 1 is very effective on the instances of our test bed and that Phase 2 can be considered as a postoptimization tool which turned out to improve the upper bound on a subset of instances for which Phase 1 fails in proving optimality of the solution.

4. Computational Analysis

The evolutionary algorithm EA and the overall algorithm MMT described in §§2 and 3, respectively, were run on a Pentium IV 2.4 GHz with 512 MB RAM under Windows XP and tested on the subset of the *DIMACS benchmark graph instances* (see Johnson and Trick 1996; all instances are available at ftp://dimacs.rutgers.edu/pub/challenge/graph/) considered by the papers describing the most effective heuristic algorithms for VCP. In particular, this set of instances contains random graphs (DSJC), geometric random graphs (DSJR and R), "quasi-random" graphs (flat), artificial graphs (le and latin_square_10), and graphs from real-life applications (school). All the computing times reported in this section are expressed in seconds of our machine. To allow a meaningful—although approximate—comparison on results obtained with different machines, a benchmark program (dfmax) together with a benchmark instance (r500.5) are available. Computing times obtained on different machines can be scaled with respect to the performance obtained on this program (our machine spent seven seconds user time).

4.1. Performance of the Evolutionary Algorithm

In this section, we report the experimental results obtained with the evolutionary algorithm EA described in §2. We fix a common set of parameters working for most of the instances, but we have to adjust parameters PS (pool size) and L (number of tabu search iterations between two crossover steps) to obtain good results on hard instances, while we use a fixed tabu tenure T of 45 for all instances. Because EA uses random numbers, for each instance we performed four runs with four different seeds for the random number generator. EA works in decision version, and it requires as input the target value k of the coloring. All the computational experiments concerning the behavior of the most effective heuristic algorithms working in decision version (see, e.g., Morgenstern 1996, Galinier and Hao 1999, Blöchliger and Zufferey 2008) are performed giving in input a "good" target value k corresponding either to the chromatic number χ or to the best-known solution value in the literature when χ is unknown. In addition, the reported computing times correspond to the average times for the successful runs. To have a fair comparison with such algorithms, we adopt the same policy for what concerns both the choice of the initial target value k and the reported computing times. In particular, we report experimental results starting from the value of k for which we have at least one successful run and ending with the value of k for which we have four successful runs.

In Table 1, we report for every considered instance the following information:

- name of the instance, n and m;
- the best-known solution value χ^* ever found in the literature (in bold when it is the proven optimal value);

Table 1 Performance of EA and of the Most Effective Heuristic Algorithms Working in Decision Version

				EA					*Impasse*		HCA			PC-RPC		
Instance	*n*	*m*	χ^*	*PS*	*L*	*#hit*	*k*	Time	*k*	Time	*#hit*	*k*	Time	*#hit*	*k*	Time
DSJC125.1	125	736	5	10	10,000		5	0								
DSJC125.5	125	3,891	17	10	10,000		17	1	17	1						
DSJC125.9	125	6,961	44	10	10,000		44	0								
DSJC250.1	250	3,218	8	10	10,000		8	1								
DSJC250.5	250	15,668	28	10	10,000		28	28	28	22	9/10	28	13			
DSJC250.9	250	27,897	72	10	10,000		72	193								
DSJC500.1	500	12,458	12	10	20,000		12	78						10/10	12	120
DSJC500.5	500	62,624	48	10	20,000	3/4	48	84	49	660	5/10	48	268	1/10	49	720
				10	20,000		49	27								
DSJC500.9	500	112,437	127	30	40,000		127	967						2/10	127	1,560
DSJC1000.1	1,000	49,629	20	10	20,000		20	303			na	20	na	1/10	20	2,640
DSJC1000.5	1,000	249,826	83	10	300,000	2/4	83	22,573	89	1,148	na	83	2,258	2/10	88	14,400
				40	50,000	3/4	84	1,632								
				40	50,000		85	845								
DSJC1000.9	1,000	449,449	224	50	50,000	2/4	226	3,340			na	224	na	4/10	226	18,000
				50	50,000	1/4	227	1,734								
				50	50,000		228	3,235								
DSJR500.1	500	3,555	**12**	10	10,000		**12**	0	**12**	0						
DSJR500.1C	500	121,275	85	20	20,000		85	142	85	5						
DSJR500.5	500	58,862	**122**	20	10,000		**122**	30	123	14						
le450_15a	450	8,168	**15**	10	10,000		**15**	0	**15**	0						
le450_15b	450	8,169	**15**	10	10,000		**15**	0	**15**	0						
le450_15c	450	16,680	**15**	10	10,000		**15**	0	**15**	5	6/10	**15**	8	10/10	**15**	2
le450_15d	450	16,750	**15**	10	10,000		**15**	1	**15**	3				10/10	**15**	8
le450_25c	450	17,343	26	10	10,000		**25**	1,321			10/10	26	55	10/10	27	1
le450_25d	450	17,425	26	10	10,000		**25**	424						10/10	27	1
le450_5a	450	5,714	**5**	10	10,000		**5**	0								
le450_5b	450	5,734	**5**	10	10,000		**5**	0								
le450_5d	450	9,757	**5**	10	10,000		**5**	0								
r125.1	125	209	**5**	10	10,000		**5**	0	**5**	0						
r125.1c	125	7,501	**46**	10	10,000		**46**	0	**46**	0						
r125.5	125	3,838	**36**	10	10,000		**36**	0	**36**	0						
r250.1	250	867	**8**	10	10,000		**8**	0	**8**	0						
r250.1c	250	30,227	**64**	10	10,000		**64**	0	**64**	0						
r250.5	250	14,849	**65**	10	10,000		**65**	8	**65**	7						
r1000.1	1,000	14,378	**20**	10	10,000		**20**	0	**20**	1						
r1000.1c	1,000	485,090	98	20	20,000		98	101	98	46						
r1000.5	1,000	238,267	237	30	10,000	3/4	237	168	241	77						
				30	10,000		238	4								
school1	385	19,095	**14**	10	10,000		**14**	0								
school1_nsh	352	14,612	**14**	10	10,000		**14**	0								
latin_square_10	900	307,350	99	50	50,000	3/4	103	3,263								
				50	50,000		104	1,820								
flat300_20_0	300	21,375	**20**	10	10,000		**20**	0	**20**	0				10/10	**20**	0
flat300_26_0	300	21,633	**26**	10	10,000		**26**	4	**26**	1				10/10	**26**	0
flat300_28_0	300	21,695	**28**	10	10,000		31	54	31	156	6/10	31	20	3/10	**28**	420
flat1000_50_0	1,000	245,000	**50**	10	100,000		**50**	33	**50**	0				10/10	**50**	18
flat1000_60_0	1,000	245,830	**60**	10	200,000		**60**	73	**60**	0				10/10	**60**	90
flat1000_76_0	1,000	246,708	83	10	250,000	1/4	82	34,056	89	897	4/5	83	1,471	5/10	87	18,000
				10	200,000	3/4	83	2,226								
				10	200,000		84	1,387								

• the input parameters (*PS* and *L*), the number of successful runs (*#hit*), the target value *k*, and the average computing time for the successful runs of EA. These results have been obtained with a time limit of 6,000 seconds with the exceptions of instance DSJC1000.5, for which we tried a big population with a time limit of 40,000 seconds, and instance flat1000_76_0, for which we experimented a long computation with a time limit of 40,000 seconds.

Moreover, Table 1 compares EA with the heuristic algorithms working in decision form that, to the best of our knowledge, represent the state of the art

for VCP. In particular, we report the computational results of:

- *Impasse*, the algorithm proposed in Morgenstern (1996), which is actually composed by three different algorithms based on the idea of *impasse class neighborhood*. All these algorithms require as input the target value k for the coloring and a couple of other parameters that are tuned for every instance separately. For each instance considered in Morgenstern (1996), we report the smallest value of k for which no failure occurred over five runs as well as the average running time to solve the instance, scaled with respect to the benchmark problem.
- HCA (*hybrid coloring algorithm*), the algorithm proposed in Galinier and Hao (1999). HCA requires as input the target value k for the coloring and a couple of other parameters that are tuned for every instance separately. For each instance considered in Galinier and Hao (1999), we report the smallest value of k for which there was at least one successful run over 10 (or 5), the number of successful runs (#*hit*) and an approximate average running time for the successful runs. Since results in Galinier and Hao (1999) were obtained on an UltraSPARC-IIi 333 MHz with 128 MB RAM, and the authors did not provide information on computing times for the benchmark instance on that system, a comparison of computing times is far from trivial. Hence, we considered the performance ratio reported by Dongarra (2006) for machines similar to those used in Galinier and Hao (1999) and in our experiments, and accordingly, we divided the computing times for algorithm HCA by a factor of six. For the instances for which no information was given about the number of successful runs and running times, we report "na."
- PC (*PartialCol*) and RPC (*React-PartialCol*), the algorithms proposed in Blöchliger and Zufferey (2008), are tabu search algorithms based on the idea of *impasse class neighborhood* by Morgenstern (1996), which implement a dynamic and a reactive tabu tenure, respectively, and require as input parameter only the target value k for the coloring. To summarize the performances of the two algorithms, for each instance considered in Blöchliger and Zufferey (2008), we report the behavior of the algorithm which finds the best solution value, breaking ties by considering the number of successful runs for that value of k and possibly the computing time. For such algorithm, we give the smallest value of k for which there was at least one successful run over 10, the number of successful runs (#*hit*), and an approximate average running time for the successful runs. The computational experiments were carried out on different Linux systems running on a Pentium IV 2 GHz with 512 MB RAM, whose performance is similar (and directly comparable) to that of the machine used in our experiments.

The computational experiments show that the evolutionary diversification is effective and able to bring the tabu search algorithm to the exploration of regions of the solution space containing high quality solutions. More in detail, EA solved for the first time, to proven optimality, instances le450_25c and le450_25d and improved the best solution known in the literature for instance flat1000_76_0. In synthesis EA, on the complete set of the 42 considered instances, three times improves on the best-known solution, 36 times finds the best-known solution, and in only three instances find a worse solution.

As to the other algorithms working in decision version, EA clearly outperforms *Impasse*; this should not be surprising because it uses the same neighborhood structure in a more complex procedure. The comparison with HCA is harder due to the scarcity of instances reported in Galinier and Hao (1999). EA finds two better solutions (le450_25c and flat1000_76_0) versus one better solution found by HCA (DSJC1000.9, for which the computing time is not reported) and shows a more robust behavior. The computing times, when reported, are comparable for equivalent values of the coloring; in particular, EA spends less than one second to color instance le450_25c with 26 colors and 2,226 seconds to color instance flat1000_76_0 with 83 colors. EA spends on average 22,573 seconds to color instance DSJC1000.5 with two successful runs over four. This time is 10 times longer than the one spent by HCA, but for this instance, the number of attempts and successful runs are not reported in Galinier and Hao (1999). EA almost always finds solution values that are better or equal to those found by PC and RPC, in the latter case with computing times that are generally shorter. The only exception is instance flat300_28_0, which is solved to optimality ($k = 28$) for the first time by Blöchliger and Zufferey (2008), improving the previous best-known solution of value 31.

4.2. Performance of the Overall Algorithm

As was observed by Morgenstern (1996), reporting timings for algorithms that require the value of k and other parameters as input does not reflect the real effort needed to find the corresponding values, and this hidden cost can increase substantially the actual computing time. When we built the overall algorithm MMT (evolutionary algorithm EA and column optimization), our objective was actually to solve the problem in optimization version with a unique setting of the parameters (with the exception of the time limit of EA), able to solve any instance. The idea is that a robust algorithm should have a good performance not only on well-studied instances from the literature (for which the best bounds, the structure, etc., are known) but also on "unknown" ones.

Table 2 Parameters of Algorithm MMT

Tabu tenure T	45
Initial PS	10
Initial L	10,000
UpdateLimit	40,000/n
Δ_L	Initial $L \cdot dens \cdot 0.55$
Δ_{PS}	Initial $PS \cdot dens$
Time limit of *CFT* (s)	150
p_{greedy}	PS/n. of uncolored vertices in the pool

As described in §2.3, we implemented a procedure that, starting from a common set of parameters, dynamically modifies the parameters PS and L for every execution of EA based on the results obtained in the previous calls. In Table 2, where *dens* denotes the density of graph G, we report the values found during the experimental setup of this procedure.

For each considered instance, four runs with different seeds for the random number generator were performed. The corresponding computational results are reported in Table 3. Because the overall algorithm MMT works in optimization version, it always gives on output a feasible solution and does not require a target value k as input. For every considered instance, we report:

- name of the instance and best-known solution value χ^* ever found in the literature (in bold when it is the proven optimal value);
- the lower bound LB, the initial upper bound UB computed during the initialization step, and the corresponding computing time;
- the time limit of EA (TL_{EA}) within Phase 1 and the average value of k (k_{avg}) after Phase 1 (these values are reported only if optimality has not been proved during the initialization);
- the average value of k after Phase 2 (this value is reported only if it is better than the corresponding value obtained at the end of Phase 1);
- the best value of k (k_{best}) found during the overall computation and the average total computing time.

In comparing these results with those of the previous section, the reader should be reminded that, in this case, the only parameter to be given in input is the value of TL_{EA} and that the problem is approached from the optimization point of view. Thus, in some way, the problem is much more "difficult" because the algorithm does not take advantage of all the information available for the well-known instances from the literature. When optimality is not proven in Phase 1 (i.e., when the solution value after Phase 1 is greater than LB) and Phase 2 does not improve the incumbent solution, the evolutionary algorithm has performed one useless iteration up to the time limit, spending an important amount of computing time after the last successful iteration. For instance, in DSJC125.1, the final solution is found on average after one second but optimality is not proven and 20 seconds are spent trying to improve on this solution. On the contrary, when Phase 2 improves the solution or optimality is proven in Phase 1 (which happens for all the le450_x instances and for school1), the time of the last improvement corresponds to the total computing time.

Table 3 shows that, for 11 instances, the average value of k is improved during the execution of Phase 2, confirming that column optimization is an effective postoptimization tool. Moreover, for some large-size instances (e.g., DSJC1000.9, latin_square_10, and flat1000_76_0), giving a larger time limit to Phase 1 leads to slightly better solutions. Of course, working in the optimization version of the problem leads to computing times longer than those required by EA, although the solution quality is slightly better. More in detail, the dynamic setup of the parameters is not always able to bring EA, executed during Phase 1, to high-quality solutions as done by the ad hoc tuning reported in Table 1. This is true, in particular, for instances flat1000_50_0 and flat1000_60_0 where the performance is quite poor. For these instances, Phase 2 is able to improve on the incumbent solution up to the solution reported in Table 1, and in three instances, namely DSJC1000.9, r1000.5, and latin_square_10, the solution is better than that found by EA alone. In particular, the complete MMT algorithm solved for the first time, to proven optimality, instance r1000.5. In synthesis, algorithm MMT, on the complete set of the 42 considered instances, three times improves on the best-known solution in the literature, 35 times finds the best-known solution in the literature, and in only four instances (DSJC1000.5, DSJC1000.9, latin_square_10, and flat300_28_0) finds a worse solution.

4.3. Comparison with Heuristics in Optimization Version

Since algorithm MMT works in optimization version, a comparison of its behavior with those of algorithms *Impasse* (Morgenstern 1996), HCA (Galinier and Hao 1999), and PC-RPC (Blöchliger and Zufferey 2008) considered in §4.1 is far from trivial. For the problem in optimization version, we report in Table 4 the computational results of:

- MIPS-CLR (*minimal-state processing search algorithm for the graph "CoLoRing" problem*), the algorithm proposed in Funabiki and Higashino (2000). This algorithm works in optimization version but requires as input the target value k_{init} for the coloring together with some other parameters whose values are tuned for hard instances. If the algorithm is not able to solve the problem with k_{init} colors, it dynamically modifies this target value. Since MIPS-CLR uses random

Table 3 Performance of the Algorithm MMT

		Initialization			Phase 1		Phase 2	Overall	
Instance	χ^*	*LB*	*UB*	Time	TL_{EA}	k_{avg}	k_{avg}	k_{best}	Time
DSJC125.1	5	3	6	0.1	20	5.00		5	21
DSJC125.5	17	8	20	0.1	20	17.00		17	122
DSJC125.9	44	28	50	0.1	20	44.00		44	121
DSJC250.1	8	3	10	0.1	20	8.00		8	21
DSJC250.5	28	9	34	0.2	80	28.00		28	117
DSJC250.9	72	34	84	0.3	80	72.50	72.00	72	89
DSJC500.1	12	4	14	0.1	200	12.25		12	210
DSJC500.5	48	9	61	0.4	200	48.25		48	388
DSJC500.9	127	42	150	1.2	200	128.75	127.75	127	433
DSJC1000.1	20	4	24	0.2	200	20.25		20	260
DSJC1000.5	83	11	108	2.0	3,000	84.25		84	8,407
DSJC1000.9	224	47	283	8.7	800	233.75	226.00	225	3,234
					3,000	229.75	225.50	225	9,476
DSJR500.1	**12**	11	13	0.1	20	**12.00**		**12**	25
DSJR500.1C	85	67	91	1.0	60	86.00	85.00	85	88
DSJR500.5	**122**	111	127	0.7	100	**122.00**		**122**	163
le450_15a	**15**	15	17	0.1	20	**15.00**		**15**	0
le450_15b	**15**	15	16	0.1	20	**15.00**		**15**	0
le450_15c	**15**	15	23	0.1	20	**15.00**		**15**	3
le450_15d	**15**	15	24	0.1	20	**15.00**		**15**	4
le450_25c	26	25	28	0.1	4,000	**25.00**		**25**	1,321
le450_25d	26	25	28	0.2	4,000	**25.00**		**25**	436
le450_5a	**5**	5	8	0.1	20	**5.00**		**5**	0
le450_5b	**5**	5	7	0.1	20	**5.00**		**5**	0
le450_5d	**5**	5	7	0.1	20	**5.00**		**5**	0
r125.1	**5**	5	5	0.1				**5**	0
r125.1c	**46**	44	46	0.1	20	**46.00**		**46**	21
r125.5	**36**	33	38	0.1	20	**36.00**		**36**	21
r250.1	**8**	8	8	0.1				**8**	0
r250.1c	**64**	55	66	0.3	20	**64.00**		**64**	21
r250.5	**65**	61	68	0.2	50	**65.00**		**65**	64
r1000.1	**20**	17	20	0.2	20	**20.00**		**20**	37
r1000.1c	98	68	105	4.0	200	98.25	98.00	98	518
r1000.5	237	225	247	4.3	400	238.75	**234.00**	**234**	753
school1	**14**	14	22	0.1	20	**14.00**		**14**	0
school1_nsh	**14**	13	25	0.1	20	**14.00**		**14**	21
latin_square_10	99	90	125	2.8	2,000	103.50	102.00	101	5,156
					3,000	103.50	101.75	101	6,346
flat300_20_0	**20**	9	38	0.2	20	**20.00**		**20**	21
flat300_26_0	**26**	9	39	0.2	20	**26.00**		**26**	36
flat300_28_0	**28**	9	39	0.2	150	31.00		31	212
flat1000_50_0	**50**	11	106	1.9	150	78.50	**50.00**	**50**	1,417
flat1000_60_0	**60**	11	105	1.9	300	74.50	**60.00**	**60**	3,645
flat1000_76_0	83	10	104	1.9	1,000	84.50		84	3,709
					3,000	83.50		83	7,325

numbers, five runs were performed for each instance. The first four columns in Table 4 under MIPS-CLR correspond to the results obtained by giving the target value k_{init} as external input. We report the value of k_{init}, the best value of k found over the five runs, the average value of k, and the average running time approximately scaled with respect to the benchmark problem (we run the benchmark problem on a machine similar to the one used in Funabiki and Higashino 2000, which spent 17 seconds of user time to solve the benchmark problem). In the latter four columns, we report the results obtained by setting the target value k_{init} equal to the cardinality of a maximal clique in the graph, computed during the initialization of the algorithm. The solution quality is slightly worse than that of the previous initialization of the target value k_{init}, whereas the computing time is larger when k_{init} is far from the final solution value.

- MMT, whose performance is summarized reporting the best value of k over four runs, the average

Table 4 Comparison of MMT with the Most Effective Heuristics in Optimization Version

		MIPS-CLR								MMT		
Instance	χ^*	k_{init}	k_{best}	k_{avg}	Time	k_{init}	k_{best}	k_{avg}	Time	k_{best}	k_{avg}	Time
DSJC125.1	5	5	5	5.0	0	4.0	5	5.0	1	5	5.00	21
DSJC125.5	17	17	17	17.0	1	9.6	17	17.2	15	17	17.00	122
DSJC125.9	44	44	44	44.0	0	32.8	44	44.0	22	44	44.00	121
DSJC250.1	8	7	8	8.0	5	4.0	8	8.0	30	8	8.00	21
DSJC250.5	28	28	28	28.4	14	11.0	28	28.6	80	28	28.00	117
DSJC250.9	72	72	72	72.4	31	41.6	72	72.4	148	72	72.00	89
DSJC500.1	12	10	12	12.4	84	5.0	12	12.8	137	12	12.25	210
DSJC500.5	48	47	49	49.4	349	12.2	49	50.0	454	48	48.25	388
DSJC500.9	127	125	127	127.8	480	53.2	128	128.8	999	127	127.75	433
DSJC1000.1	20	20	21	21.0	90	5.2	21	21.0	776	20	20.25	260
DSJC1000.5	83	82	88	89.0	4,658	14.2	89	89.6	2,634	84	84.25	8,407
DSJC1000.9	224	228	228	229.6	1,565	62.0	228	229.8	7,087	225	226.00	3,234
DSJR500.1	**12**	12	**12**	**12.0**	0	12.0	**12**	**12.0**	0	**12**	**12.00**	25
DSJR500.1C	85	85	85	85.0	6					85	85.00	88
DSJR500.5	**122**	122	**122**	123.4	276	122.0	**122**	123.4	276	**122**	**122.00**	163
le450_15a	**15**	15	**15**	**15.0**	1	15.0	**15**	**15.0**	1	**15**	**15.00**	0
le450_15b	**15**	15	**15**	**15.0**	1	15.0	**15**	**15.0**	1	**15**	**15.00**	0
le450_15c	**15**	15	**15**	15.2	11	15.0	**15**	15.2	11	**15**	**15.00**	3
le450_15d	**15**	15	**15**	**15.0**	5	15.0	**15**	**15.0**	5	**15**	**15.00**	4
le450_25c	26	26	26	26.0	7					**25**	**25.00**	1,321
le450_25d	26	26	26	26.4	1					**25**	**25.00**	436
le450_5a	**5**	5	**5**	**5.0**	1	5.0	**5**	**5.0**	1	**5**	**5.00**	0
le450_5b	**5**	5	**5**	**5.0**	2	5.0	**5**	**5.0**	2	**5**	**5.00**	0
le450_5d	**5**	5	**5**	**5.0**	3	5.0	**5**	**5.0**	3	**5**	**5.00**	0
r125.1	**5**	5	**5**	**5.0**	0	5.0	**5**	**5.0**	0	**5**	**5.00**	0
r125.1c	**46**	46	**46**	**46.0**	0	46.0	**46**	**46.0**	0	**46**	**46.00**	20
r125.5	**36**	36	**36**	**36.0**	0	36.0	**36**	**36.0**	0	**36**	**36.00**	21
r250.1	**8**	8	**8**	**8.0**	0	8.0	**8**	**8.0**	0	**8**	**8.00**	0
r250.1c	**64**	64	**64**	**64.0**	2					**64**	**64.00**	21
r250.5	**65**	65	**65**	65.8	16	65.0	**65**	65.8	16	**65**	**65.00**	64
r1000.1	**20**	20	**20**	**20.0**	0	20.0	**20**	**20.0**	0	**20**	**20.00**	37
r1000.1c	98	98	98	98.8	557					98	98.00	518
r1000.5	237	234	237	238.6	1,345					**234**	**234.00**	753
school1	**14**	14	**14**	**14.0**	0	14.0	**14**	**14.0**	0	**14**	**14.00**	0
school1_nsh	**14**	14	**14**	**14.0**	1	14.0	**14**	**14.0**	1	**14**	**14.00**	21
latin_square_10	99	90	99	100.2	938	90.0	99	100.2	938	101	102.00	5,156
flat300_20_0	**20**	20	**20**	**20.0**	2	10.2	**20**	**20.0**	114	**20**	**20.00**	21
flat300_26_0	**26**	26	**26**	**26.0**	1	11.0	**26**	**26.0**	104	**26**	**26.00**	36
flat300_28_0	**28**	30	31	31.0	133	11.4	31	31.2	1,355	31	31.00	212
flat1000_50_0	**50**	50	**50**	**50.0**	14	13.0	**50**	**50.0**	2,351	**50**	**50.00**	1,417
flat1000_60_0	**60**	60	**60**	**60.0**	59	13.6	**60**	**60.0**	2,436	**60**	**60.00**	3,645
flat1000_76_0	83	84	87	87.8	2,499	13.6	87	88.2	7,087	83	83.50	7,325

value of k, and the average computing time up to the time limit or to proven optimality (so as to allow a comparison with the timings of MIPS-CLR).

Again, values in Table 4 are in bold when corresponding to a provable optimal value.

Since when we use MMT we do not take advantage of any information from the literature (such as the expected value of the coloring), we think that a fair time comparison with MIPS-CLR should be done with the values obtained by setting the target value k_{init} equal to the cardinality of a maximal clique, i.e., with the values reported in the last four columns of MIPS-CLR. MMT always finds solutions that are better or equal to those found by MIPS-CLR, considering both the best value or the average solution value with the exception of instance latin_square_10, for which MIPS-CLR finds the best-known solution. Table 4 also shows that the running times of the algorithms on the common instances are practically the same.

To compare with a single index the performance of the different algorithms considered in this paper, we compute the average ratio between the solution found by each algorithm and the best-known solution value χ^* from the literature. This ratio always refers to the best results reported for the corresponding instance in the associated paper (i.e., k for *Impasse*

Table 5 Average Gap on the Common Subset of Instances

	Instance set from Morgenstern (1996)	Instance set from Galinier and Hao (1999)	Instance set from Blöchliger and Zufferey (2008)	Instance set from Funabiki and Higashino (2000)
Impasse	1.0114			
HCA		1.0163		
PC-RPC			1.0086	
MIPS-CLR				1.0072
EA	1.0041	1.0086	1.0017	1.0018
MMT	1.0046	1.0095	1.0029	1.0013

by Morgenstern 1996, HCA by Galinier and Hao 1999, PC-RPC by Blöchliger and Zufferey 2008, and EA and k_{best} for MIPS-CLR by Funabiki and Higashino 2000 and MMT) and χ^*. Since Morgenstern (1996), Galinier and Hao (1999), and Blöchliger and Zufferey (2008) did not consider the entire set of instances, in Table 5 we compare our results with those of the other algorithms on the common subset of instances.

Table 5 confirms that the proposed approaches EA and MMT outperform the other algorithms with respect to the solution quality, with comparable computing times.

5. Conclusions

In this paper, we presented the two-phase metaheuristic algorithm MMT for the vertex coloring problem. The first phase of MMT is based on an evolutionary algorithm which combines an effective tabu search with a diversification procedure based on a specialized crossover operator; the second phase is a postoptimization phase based on the set covering formulation of the problem.

Extensive computational experiments performed on 42 instances from the well-known DIMACS benchmark graph instances show that the evolutionary algorithm is very effective and has a robust behavior on all the considered instances, still requiring the experimental setup of a couple of input parameters. This evolutionary algorithm is subsequently combined, in the overall MMT algorithm, with a procedure which performs the dynamic setup of the parameters during the computation. A postoptimization phase is finally performed. The overall MMT algorithm approaches the problem in optimization version, requiring no input parameter with the exception of the time limit of the first phase. Computational experiments show that the quality of the solutions found by MMT (with the dynamic setup of the parameters and the postoptimization phase) is slightly better than that of EA (with the ad hoc setup of the parameters), leading the proposed approach to be the state-of-the-art heuristic algorithm for the problem. More in detail, three instances from the literature (namely, le450_25c, le450_25d, and r1000.5) were solved for the first time to proven optimality, the best-known solution for instance flat1000_76_0 was improved, and the best-known solution values were still found on 35 of the remaining 38 instances.

The main deficiency of our approach is that when optimality is not proven, the computation continues up to the time limit, even if the problem is already solved or the diversity of the population of the evolutionary algorithm is low and it is unlikely that new improved solutions will be found. Thus, future work should deal with the development of a procedure to stop the computation when the probability of improving the best incumbent solution becomes too low. Future work should also continue with the search for improved lower bounds for the problem.

Acknowledgments

The authors would like to thank Luca Agostini for his help in implementing the code and carrying out preliminary computational results. All experiments have been performed in the Lab.O.R. (Laboratory of Operations Research) of the University of Bologna. The authors also thank two anonymous referees for helpful comments and Massimiliano Caramia and Paolo Dell'Olmo for stimulating discussions. This research was partially supported by MIUR, Italy.

References

Blöchliger, I., N. Zufferey. 2008. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Comput. Oper. Res.* **35** 960–975.

Bollobàs, B., A. Thomason. 1985. Random graphs of small order. *Ann. Discrete Math.* **28** 47–97.

Brèlaz, D. 1979. New methods to color the vertices of a graph. *Comm. ACM* **22** 251–256.

Caprara, A., M. Fischetti, P. Toth. 1999. A heuristic method for the set covering problem. *Oper. Res.* **47** 730–743.

Caramia, M., P. Dell'Olmo. 1999. A fast and simple local search for graph coloring. J. S. Vitter, C. D. Zaroliagis, eds. *Proc. 3rd Workshop on Algorithm Engrg. WAE'99, Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 319–313.

Chow, F. C., J. L. Hennessy. 1990. The priority-based coloring approach to register allocation. *ACM Trans. Programming Languages Systems* **12** 501–536.

Culberson, J. C., F. Luo. 1996. Exploring the *k*-colorable landscape with iterated greedy. D. S. Johnson, M. A. Trick, eds. *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challenge, 1993*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI, 245–284.

Davis, L. 1998. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.

de Werra, D. 1985. An introduction to timetabling. *Eur. J. Oper. Res.* **19** 151–162.

Diaz, I. M., P. Zabala. 2002. A branch and cut algorithm for graph coloring. D. S. Johnson, A. Mehrotra, M. Trick, eds. *Proc. Computational Sympos. Graph Coloring and Its Generalization, Ithaca, NY*, 55–62.

Dongarra, J. J. 2006. Performance of various computers using standard linear equations software (linpack benchmark report). Technical Report CS-89-85, Computer Science Department, University of Tennessee, Knoxville.

Dorne, R., J. K. Hao. 1998. Tabu search for graph coloring, t-coloring and set t-colorings. 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, 77–92.

Fleurent, C., J. A. Ferland. 1996. Object-oriented implementation of heuristic search methods for graph coloring. D. S. Johnson, M. A. Trick, eds. *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challenge, 1993*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI, 619–652.

Funabiki, N., T. Higashino. 2000. A minimal-state processing search algorithm for graph coloring problems. *IEICE Trans. Fundamentals* **E83-A** 1420–1430.

Galinier, P., J. K. Hao. 1999. Hybrid evolutionary algorithms for graph coloring. *J. Combin. Optim.* **3** 379–397.

Gamst, A. 1986. Some lower bounds for a class of frequency assignment problems. *IEEE Trans. Vehicular Tech.* **35** 8–14.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco.

Glover, F., M. Laguna. 1997. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

Hertz, A., D. de Werra. 1987. Using tabu search techniques for graph coloring. *Computing* **39** 345–351.

Johnson, D. S., M. A. Trick. 1996. *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challange, 1993*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI.

Johnson, D. S., C. R. Aragon, L. A. McGeoch, C. Schevon. 1991. Optimization by simulated annealing: An experimental evaluation; Part II, graph coloring and number partitioning. *Oper. Res.* **39** 378–406.

Leighton, F. T. 1979. A graph coloring algorithm for large scheduling problems. *J. Res. National Bureau Standards* **84** 489–503.

Mehrotra, A., M. A. Trick. 1996. A column generation approach for graph coloring. *INFORMS J. Comput.* **8** 344–354.

Monaci, M., P. Toth. 2006. A set-covering based heuristic approach for bin-packing problems. *INFORMS J. Comput.* **18** 71–85.

Morgenstern, C. A. 1996. Distributed coloration neighborhood search. D. S. Johnson, M. A. Trick, eds. *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challange, 1993*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI, 335–358.

Sager, T. J., S. Lin. 1991. A pruning procedure for exact graph coloring. *ORSA J. Comput.* **3** 226–230.

Sewell, E. C. 1996. An improved algorithm for exact graph coloring. D. S. Johnson, M. A. Trick, eds. *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challange, 1993*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI, 359–373.

Woo, T. K., S. Y. W. Su, R. Newman Wolfe. 2002. Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm. *IEEE Trans. Comm.* **39** 1794–1801.