A Memetic Algorithm for Graph Coloring

Zhipeng Lü and Jin-Kao Hao

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, Cedex 01, France

Abstract

Given an undirected graph $G = (V, E)$ with a set V of vertices and a set E of edges, the graph coloring problem consists of partitioning all vertices into k independent sets and the number of used colors k is minimized. This paper presents a memetic algorithm (denoted by MACOL) for solving the problem of graph coloring. The proposed MACOL algorithm integrates several distinguished features such as an adaptive multi-parent crossover (AMPaX) operator and a distance-and-quality based replacement criterion for pool updating. The proposed algorithm is evaluated on the DIMACS challenge benchmarks and computational results show that the proposed MACOL algorithm achieves highly competitive results, compared with 11 state-of-the-art algorithms. The influence of some ingredients of MACOL on its performance is also analyzed.

keywords: graph coloring, memetic algorithm, crossover operator, pool updating

1 Introduction

Given an undirected graph $G = (V, E)$ with a set V of vertices and a set E of edges, a legal k-coloring of G corresponds to a partition of V into k independent sets where an independent set is a subset of non-adjacent vertices of G. Graph coloring aims at finding the smallest k for a given graph G (its *chromatic number* $\chi(G)$) such that G has a legal k-coloring.

Besides its theoretical significance as a canonical NP-Hard problem [19], graph coloring arises naturally in a variety of real-world applications, such as register allocation [7], timetabling [4], frequency assignment [37], crew scheduling [17], printed circuit testing [18], satellite range scheduling [38] as well as manufacturing [20]. Due to its NP-Hardness, heuristics and metaheuristics are

Email addresses: zhipeng.lui@gmail.com, lu@info.univ-angers.fr (Zhipeng Lü), hao@info.univ-angers.fr (Jin-Kao Hao).

preferred to handle large and hard coloring problems. We briefly review below some of the most representative algorithms.

The first heuristic approaches to solving the graph coloring problem were based on greedy construction, which color the vertices of the graph one by one guided by a predefined greedy function. These algorithms are very fast by nature but their quality is generally unsatisfactory. The best known algorithms in this class are the *largest saturation degree* heuristic (DSATUR) [3] and the *recursive largest first* heuristic (RLF) [28]. In recent decades, these greedy constructive heuristics are often used to generate initial solutions for advanced metaheuristic algorithms.

On the other hand, local search based metaheuristic algorithms have been widely used to tackle the graph coloring problem in recent decades. One representative example is the so-called *Tabucol* algorithm which is the first application of Tabu Search to graph coloring [23]. *Tabucol* has been latter improved by several researchers and used as subcomponent of more elaborate coloring algorithms (see for examples [8,11,14]). Other local search metaheuristic methods include Simulated Annealing [5,25], Iterated Local Search [6], Reactive Partial Tabu Search [2], GRASP [27], Variable Neighborhood Search [1], Variable Space Search [24] and *Clustering*-Guided Tabu Search [35]. Interested readers are referred to [15] for a comprehensive survey of the local search approaches for graph coloring presented in recent years.

In parallel, researchers have also proposed other different approaches for solving the graph coloring problem, especially for tackling some large random graphs. One of the most recent and very promising approaches is based upon hybridization that embeds a local search algorithm into the framework of an evolutionary algorithm in order to achieve a better tradeoff between intensification and diversification (see for examples [9,14,16,29,34]).

Another approach for dealing with large graphs is to first extract several large independent sets and then solve the residual graph. This approach is particularly useful for very large graphs (see for example [12]) and can be employed as a pre-processing step for other search algorithms.

This paper presents MACOL, a hybrid metaheuristic algorithm integrating a tabu search procedure with an evolutionary algorithm for solving the graph coloring problem. In this algorithm, we highlight the importance of the diversity of individuals and the balance between intensification and diversification. To achieve this goal, we devise an adaptive multi-parent crossover operator, which is an extension of the *greedy partition crossover* (GPX) presented in [14]. Furthermore, we highlight the diversity of the population by defining a new replacement criterion for the population updating. This replacement criterion takes into account both the quality and the diversity among individuals.

Experiments are presented on the set of DIMACS challenge benchmark graphs, showing the proposed algorithm achieves very competitive results matching many and improving some previous best solutions. Furthermore, we carefully analyze the influence of some critical components of the proposed hybrid algorithm, showing the essential factors of the proposed algorithm and shedding light on the possible ways to further improve its performance.

The rest of this paper is organized as follows. Section 2 describes the general framework and each main component of the proposed MACOL algorithm. In Section 3 computational results are presented and compared with 11 state-of-the-art algorithms in the literature. Section 4 investigates several essential parts of the proposed memetic algorithm, followed by some conclusions.

2 Memetic Algorithm

The graph coloring problem can be solved from the point view of constraint satisfaction by solving a series of k-coloring problems [15]. We starts from an initial number of k colors ($k \leq |V|$) and solve the k-coloring problem. As soon as the k-coloring problem is solved, we decrease k by setting k to k-1 and solve again the k-coloring problem. This process is repeated until no legal k-coloring can be found. Therefore, we will only consider the k-coloring problem in the rest of the paper, i.e., our MACOL algorithm aims to find a legal k-coloring from an initially conflicting (or illegal) k-coloring. It should be clear that a smaller k for a given graph leads to a harder k-coloring problem. The solution approach just described solves thus a series of k-coloring problems of increasing difficulty.

2.1 Main Scheme

Generally speaking, our MACOL algorithm can be considered as a memetic algorithm [31]. It is composed of four main components: initial population generator, tabu search procedure, multi-parent crossover operator and population updating rule. From an initial population of illegal k-coloring individuals, tabu search is used to improve each individual of the population by reducing color conflicts of adjacent vertices. The crossover operator is then used to generate new k-colorings which are also further optimized by tabu search. The population updating rule decides whether such a new offspring k-coloring should be inserted into the population and if yes, which existing individual is replaced. The general algorithm architecture is described in Algorithm 1. In the following subsections, the four components of our memetic algorithm are described in details.

Algorithm 1 Pseudo-code of the Memetic Algorithm for k-Coloring

Input: Graph G, number of colors k, population size p
Output: The best k-coloring S^* found so far
$P = \{S_1, \ldots, S_p\} \leftarrow$ Initial_Population( ) /* Section 2.3 */
for $i = \{1, \ldots, p\}$ **do**
  $S_i \leftarrow$ Tabu_Search(S_i) /* Section 2.4 */
end for
$S^* = arg\ min\{f(S_i), i = 1, \ldots, p\}$
repeat
  Randomly choose m individuals $\{S_{i1}, \ldots, S_{im}\}$ from P $(2 \leq m \leq p)$
  $S_0 \leftarrow$ Adaptive_Multi-Parent_Crossover($S_{i1}, \ldots, S_{im}$) /* Section 2.5 */
  $S_0 \leftarrow$ Tabu_Search(S_0) /* Section 2.4 */
  if $f(S_0) < f(S^*)$ **then**
    $S^* = S_0$
  end if
  $\{S_1, \ldots, S_p\} \leftarrow$ Pool_Updating($S_0, S_1, \ldots, S_p$) /* Section 2.6 */
until Stop condition met

2.2 Search Space and Evaluation Function

When designing a search algorithm for solving a specific problem, one has to define the search space. For the graph coloring problem, the authors of [15] identify four different strategies for defining the search space: *legal strategy*, *penalty strategy*, *k-fixed partial legal strategy*, *k-fixed penalty strategy*.

In this paper, we adapt the *k-fixed penalty strategy* which is also used by many coloring algorithms. For a given graph $G = (V, E)$, the number k of colors is fixed and the search space contains all possible (legal and illegal) k-colorings. A k-coloring will be represented by $S = \{V_1, \ldots, V_k\}$ such that V_i is the set of vertices receiving color i. Thus, if for all $\{u, v\} \in E$, $u \in V_i$ and $v \in V_j$, $i \neq j$, then S is a legal k-coloring. Otherwise, S is an illegal (or conflicting) k-coloring. The optimization objective is then to minimize the number of conflicting edges (referred to *conflict number* hereafter) and find a legal k-coloring in the search space.

Given a k-coloring $S = \{V_1, \ldots, V_k\}$, the evaluation function f counts the conflict number induced by S such that

$$f(S) = \sum_{\{u,v\} \in E} \delta_{uv} \tag{1}$$

where

$$\delta_{uv} = \begin{cases} 1, \text{ if } u \in V_i,\ v \in V_j \text{ and } i = j, \\ 0, \text{ otherwise.} \end{cases} \tag{2}$$

Accordingly, a coloring S with $f(S) = 0$ corresponds to a legal k-coloring.

Relevant examples of using other search space strategies such as *legal strategy*, *penalty strategy*, *k-fixed partial legal strategy* can be found in [2,25,29,30].

2.3 Initial Population

The individuals of initial population are generated with a randomized version of the DANGER coloring heuristic proposed in [21]. In the original DANGER heuristic, the choice of the next vertex to be colored is based upon a measure called "dynamic vertex dangers measure" (see [21] for more details). The color assignment uses another heuristic measure; the chosen vertex is assigned the color that is least likely to be required by neighboring vertices. In our implementation, the choices of the next vertex and its color are made in a probabilistic way according to the above heuristic measures. Thus a vertex (or color) with a higher score will have more chances to be selected.

Moreover, we take advantage of the initialization step to build a diversified population. For each new k-coloring, we observe its *distance* to the previous generated ones. If the new k-coloring is too close to one of the previous ones, then it is discarded and another new k-coloring is generated. The concept of the individual *distance* will be discussed in details in Section 2.6.

Let us mention that we also used at the beginning of this study a pure random method to generate the initial k-colorings of the population. We did not observe real difference for the final coloring results. Nevertheless, since the quality of the solutions generated using the above heuristic procedure is better than using the pure random procedure, this helps the memetic algorithm to save some computational efforts during the first generations of its search. Notice also that other popular greedy heuristics are often employed to generate initial solutions, including *largest saturation degree* heuristic (DSATUR) [3] and the *recursive largest first* (RLF) heuristic [28].

2.4 Tabu Search Algorithm

In this paper, we employ a simple Tabu Search algorithm as our local search procedure. A neighborhood of a given k-coloring is obtained by moving a *conflicting* vertex u from its original color class V_i to another color class V_j ($i \neq j$) (denoted by (u, i, j)), called "critical one-move" neighborhood. Therefore, for a k-coloring S with cost $f(S)$, the size of this neighborhood is bounded by $O(f(S) \times k)$. More details can be found in a similar TS in [14].

In order to evaluate the neighborhood efficiently, we employ an incremental evaluation technique used in [11]. The main idea consists of maintaining a special data structure to record the *move values* for each neighborhood move. Each time a move is carried out, only the move values affected by this move are updated accordingly.

For the tabu list, once move (u, i, j) is performed, vertex u is forbidden to move back to color class V_i for the next l iterations. Here, the tabu tenure l is dynamically determined by $l = \mu \cdot f(S) + r(10)$ where $r(10)$ takes a random number in $\{1, \ldots, 10\}$ [14]. In our case, μ is set to 1 (while in [14] $\mu = 0.6$). The stop condition of our tabu search is just the maximal number of iterations with which the best k-coloring has not been improved. We call this number *depth of tabu search*, denoted by α.

2.5 Adaptive Multi-Parent Crossover (AMPaX)

In a genetic algorithm, crossover is usually the main operator to generate new individuals to be added into the population, thus it is also believed to be the most important factor in a hybrid algorithm. In the literature, there are two kinds of crossover operators for graph coloring problem: assignment crossover as in [12] and partition crossover as in [9,14,16,22,34], the latter one sharing ideas of grouping genetic algorithm [10].

The assignment crossover is believed to be less powerful than the partition crossover for graph coloring since it cannot transmit "*good properties*" of the parent individuals to the offspring. In contrast, the general idea of the partition crossover is to consider a configuration of coloring as a partition of vertices into color classes and crossover operator is used to transmit color classes from one generation to another, which is much more meaningful and relevant for the coloring problem.

In this paper, we propose a new Adaptive Multi-Parent crossover (AMPaX) operator which can be considered as an extended version of the GPX crossover presented in [14]. There are two main differences between AMPaX and GPX. First, AMPaX uses two or more parents to produce an offspring while GPX are based on exactly two parents. Second, at each step of the crossover operation, AMPaX adaptively choose a parent and a color class to transmit to the offspring while these choices are done in a successive way in GPX.

Another relevant crossover operator in the literature is the "GHZ recombination operator" proposed in [16]. Both the AMPaX and GHZ operators utilize multi sources for creating solutions. However, one finds several differences between them. Notice first that in our case, the population is composed of complete k-coloring solutions, while in [16] the population contains indepen-

dent sets. From this, our AMPaX operator combines, in a particular way (see Section 2.5.2), several k-coloring solutions chosen from the population, while the GHZ operator builds an offspring by considering all the independent sets in the population. In addition, at each recombination step, the GHZ operator transmits one independent set to the offspring, while our AMPaX operator retains a set of vertices, which may or may not be an independent set.

Finally, let us mention that AMPaX also shares similarities with the recombination operators described in [22,34] for graph coloring and the recombination operator of [38] designed for a satellite range scheduling problem.

2.5.1 General Idea

A legal k-coloring is a collection of k independent sets. With this point of view, if we could maximize the size of the independent sets by a crossover operator as far as possible, it will in turn help to push those left vertices into independent sets. In other words, the more vertices are transmitted from parent individuals to the offspring within k steps, the less vertices are left unassigned. In this way, the obtained offspring individual has more possibility to become a legal coloring. Both GPX and AMPaX crossovers are based on this idea.

Given two parent k-colorings S_1 and S_2, the GPX crossover in [14] buids step by step the k classes $V_{01}, \ldots, V_{0k}$ of the offspring k-coloring S_0. At the first step, the GPX crossover operator builds the class V_{01} by choosing the class having the maximum number of vertices in parent S_1. Similarly, the second class V_{02} of S_0 is built by considering the second parent S_2. Then, two parents S_1 and S_2 are successively considered to build the remaining color classes $V_{03}, \ldots, V_{0k}$ of S_0. Once k color classes are built, each left uncolored vertex is assigned a random color.

AMPaX uses m ($m \geq 2$) parents to generate an offspring. At each step, AMPaX builds one color class by considering the color classes of all parents with a global view. With this strategy, AMPaX has an advantage over the GPX crossover, i.e., to build a color class, AMPaX has more choices, thus giving more chance of finding larger color class in the parent individuals. Consequently, it could transmit more vertices from parents to offspring within k steps.

2.5.2 The Adaptive Multi-Parent Crossover Procedure

The AMPaX operator builds one by one the color classes of the offspring. After one color class has been built, all the vertices in this color class are removed from all parents. This process is repeated until all k color classes are built. At the end of these k steps, some vertices may remain unassigned. These vertices

are handled in the same way as in the GPX crossover: they are randomly assigned to a color class.

Given m chosen parents $\{S_1, \ldots, S_m\}$ $(m \geq 2)$, each k-coloring S_i can be represented as $S_i = \{V_{i1}, \ldots, V_{ik}\}$ where V_{ij} is the set of vertices with color j. Then we produce an offspring $S_0 = \{V_{01}, \ldots, V_{0k}\}$ using these m parents individuals as follows.

We attempt to build each color class with a global view, i.e., each time we choose the color class with the maximal cardinality in all m parent individuals. It is aimed to transmit as more vertices as possible at each step such that the number of unassigned vertices after k transmitting steps is as small as possible. After one color class of S_0 is built, all vertices in this color class are removed from all the m parent individuals.

Furthermore, in order to diversify the offspring individual and avoid focusing on a single parent, at each step we forbid the current parent to be reconsidered within a few number of steps, denoted by q. In this paper, we set q to be $\lfloor m/2 \rfloor$. In other words, we build the ith color class V_{0i} by choosing the color class with the maximal cardinality among the parent individuals which are not used in building the previous q color classes $V_{0i-1}, \ldots, V_{0i-q}$. In principle, this idea is similar to the spirit of tabu search. Our Adaptive Multi-Parent crossover is presented in Algorithm 2.

Algorithm 2 Pseudo-code of the Adaptive Multi-Parent Crossover Operator

1: **Input**: m parent individuals $\{S_1, \ldots, S_m\}$ $(m \geq 2)$
2: **Output**: An offspring individual $S_0 = \{V_{01}, \ldots, V_{0k}\}$
3: Set forbidden length for each parent: $q(S_i) = 0$, $i = 1, \ldots, m$
4: **for** $i = 1, \ldots, k$ **do**
5: Build the ith color class V_{0i} of S_0 as follows:
6: Indicate the non-forbidden color classes: $\widehat{V} = \{V_{uv} | , q(S_u) = 0\}$
7: Find out the maximal cardinality class: $V_{u^*v^*} = arg\ max\{|V_{uv}||, V_{uv} \in \widehat{V}\}$
8: Let V_{0i} be the set of vertices in S_{u^*} with color v^*: $V_{0i} = V_{u^*v^*}$
9: Remove all the vertices in V_{0i} from m parent individuals:
$V_{uv} = V_{uv} \backslash V_{0i}$, $u = 1, \ldots, m, v = 1, \ldots, k$
10: Update forbidden length: $q(S_u) = q(S_u) - 1$ $(q(S_u) > 0)$ and $q(S_{u^*}) = \lfloor m/2 \rfloor$
11: **end for**
12: Randomly choose a color class for each unassigned vertex.

Now we consider the time complexity of the AMPaX crossover operator. It mainly consists of three phases: finding out the maximal cardinality color class (line 7), removing vertices from all m parents (line 9) and updating forbidden length (line 10). For the phase of finding out the maximal cardinality color class (line 7), its time complexity is $O(\frac{m}{2} \times k^2)$ since it repeats k times and each time it has at most $\frac{m}{2} \times k$ operations. For the removing vertices phase (line 9), its time complexity is $O(m \times n)$ since at most n vertices are removed and each vertex is removed from m parents. For the forbidden length updating

phase (line 10), it is obvious that its time complexity is $O(m \times k)$. Therefore, the total time complexity of the AMPaX crossover is $O(m \times (k^2 + n))$. Given that m is a parameter and the time complexity of the GPX crossover in [14] is $O(k^2 + n)$, we can conclude that the time complexity of our AMPaX crossover is just a constant times of the GPX crossover.

Figure 1 illustrates how our AMPaX crossover works. In this example, there are 4 parent individuals $S_1, \ldots, S_4$ with k=3 colors and 9 vertices $u_1 \ldots, u_9$. The number of forbidden steps q for each parent is indicated below. At the first step, color class $V_{22} = \{u_2, u_4, u_8, u_9\}$ is chosen to become the first class V_{01} of the offspring since it has the maximal cardinality. Then, vertices u_2, u_4, u_8, u_9 are removed from all parents. Similarly, V_{02} and V_{03} can be built by choosing color class V_{41} and V_{32} respectively, as shown in the illustration. After three steps, there is no vertex left unassigned. Thus, a complete k-coloring offspring is constructed.

2.6 Pool Updating

In the literature, various pool updating strategies have been proposed. In [16], the k independent sets transformed from an offspring solution replace k random color classes in the pool. In [29], if the offspring is *similar* to one of the parents or to a solution in the pool, this offspring will not be inserted into the population. Instead, a completely new solution is added to avoid premature convergence. According to the definition, two solutions are *similar* if they have the same score and the same number of uncolored vertices. In [22,34],

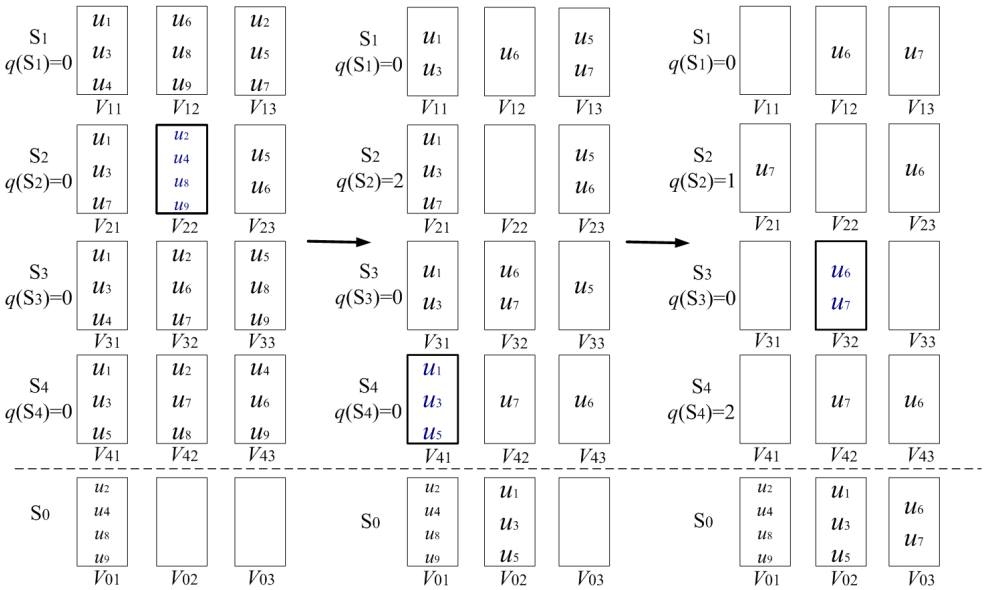

Fig. 1. Adaptive Multi-Parent crossover illustration

the pool updating is conditioned by both the solution quality and the distance between the offspring and other solutions in the population. We adopt a pool updating strategy, simultaneously taking into account the solution quality and the diversity of solutions.

When an offspring individual S_0 is obtained by the AMPaX crossover, we improve S_0 by the tabu search algorithm and then decide whether the offspring should be inserted into the population, replacing the *worst* parent. For this purpose, we define a distance-and-quality goodness score to judge whether the offspring S_0 should be inserted into the population. The main idea is that if the quality of S_0 is not good enough or if S_0 is *similar* to one of the population individuals, this offspring should not be inserted into the pool. For two k-colorings, we say that they are *similar* if one k-coloring can be transformed into another using a few number of *one-move* (Section 2.4) steps. To make things clear, we give the following definitions.

Definition 1. Distance Between two k-colorings: Given two k-colorings $S_i = \{V_{i1}, \ldots, V_{ik}\}$ and $S_j = \{V_{j1}, \ldots, V_{jk}\}$, the distance between S_i and S_j is defined as the least number of *one-move* steps for transforming S_i to S_j (or S_j to S_i), denoted by d_{ij}.

It is easy to observe that we can calculate d_{ij} by solving a bipartite matching problem [32]. Assume that $share_{uv}$ is the number of vertices shared by the uth color class V_{iu} of S_i and the vth color class V_{jv} of S_j, then the maximal total number of sharing vertices in the same class for S_i and S_j (denoted by mc_{ij}) is defined as a perfect matching where the sum of the numbers of the shared vertices in the matching has a maximal value. Therefore, d_{ij} is just equal to $n - mc_{ij}$, where n is the total number of vertices. In this paper, we use an exact algorithm for solving the matching problem, the augmenting path algorithm implemented in [36].

Definition 2. Distance Between one k-coloring and a Population: Given a population $P = \{S_1, \ldots, S_p\}$ and the distance d_{ij} between any two k-colorings S_i and S_j $(i, j = 1, \ldots, p, i \neq j)$, the distance between a k-coloring S_i $(i = 1, \ldots, p)$ and the population P is defined as the minimum distance between S_i and any other k-coloring in P, denoted by $D_{i,P}$:

$$D_{i,P} = min\{d_{ij} | S_j \in P, j \neq i\} \tag{3}$$

Definition 3. Goodness Score of a k-coloring for a Population: Given a population $P = \{S_1, \ldots, S_p\}$ and the distance d_{ij} between any two k-colorings S_i and S_j $(i, j = 1, \ldots, p, i \neq j)$, the goodness score of k-coloring S_i for population P is defined as:

$$h_{i,P} = f(S_i) + e^{\beta / D_{i,P}} \tag{4}$$

where $f(S_i)$ is the conflict number of S_i and β is a parameter. Since the maximum value of $D_{i,P}$ is n, we set $\beta = \lambda n$ and $\lambda = 0.08$ in this paper.

It is clear that the smaller the goodness score $h_{i,P}$, the better k-coloring S_i. This choice can be justified as follows. On the one hand, at the beginning of the search, the quality of the solutions in the population is relatively poor (i.e., $f(S_i)$ is large) and the solutions are also very diversified (i.e., $D_{i,P}$ is large). Therefore, we want the solution quality to be the dominant part of this function. One easily observes that in this case the second part of the goodness score becomes trivial such that the quality $f(S_i)$ dominates this scoring function.

On the other hand, as the algorithm progresses, both $f(S_i)$ and $D_{i,P}$ values in the current population become smaller and smaller. In this situation, since all the solutions in the population have quite good quality, we have to enlarge the importance of the diversity of the solutions to avoid a premature convergence of the population. If a solution S_i is too close to at least one of the solutions in the population (i.e., $D_{i,P}$ is small), the second part of the function in Eq. (4) should become large enough such that the goodness score of this k-coloring turns to be quite bad, thus being likely to be discarded. We observe that the function in Eq. (4) fulfills these requirements.

At this stage, let us notice that other goodness score functions are possible. For instance, the following one has also been experimented: $h_{i,P} = f(S_i) + \beta / D_{i,P}$. In practice, we observe that this non-exponential function leads to slightly worse results in terms of quality of the solutions than the one in Eq. (4) because of the above analyzed reasons.

Given an offspring S_0 optimized by tabu search and a population $P = \{S_1, \ldots, S_p\}$, we use the following rule to decide whether S_0 should be inserted into the population. First of all, S_0 is temporarily inserted into the population P, i.e., $P' = P \cup \{S_0\}$. Then, the goodness score for each k-coloring $S_i \in P'$ is calculated according to Eq. 4 and the worst k-coloring (with the largest value of goodness score) is identified, denoted by S_w. If S_w is not the offspring k-coloring S_0, then S_0 will be inserted into the population and replace the worst k-coloring S_w, i.e., $P = P \cup \{S_0\} \backslash \{S_w\}$. Otherwise, the second worst k-coloring S_{sw} in the population is replaced by S_0 with a probability $P_r = 0.2$, i.e., $P = P \cup \{S_0\} \backslash \{S_{sw}\}$. The Pseudo-code of our pool updating strategy is presented in Algorithm 3.

Our pool updating strategy mainly consists of three phases: calculate $D_{i,P'}$ for each solution S_i (line 5); identify the worst k-coloring (line 8) and update the pool (lines 9 to 15). One easily observes that both the second and the last phases can be fulfilled within $O(p)$ time. For the first phase, in practice we use a $p \times p$ matrix to record all the one-to-one distances between any two solutions

Algorithm 3 Pseudo-code of the Pool Updating Rule

Input: Population $P = \{S_1, \ldots, S_p\}$ and offspring k-coloring S_0
Output: Updated Population $P = \{S_1, \ldots, S_p\}$
Tentatively add S_0 to Population P: $P' = P \cup \{S_0\}$
for $i = 0, \ldots, p$ **do**
    Calculate the distance between S_i and P' ($D_{i,P'}$) according to Eq. 3
    Calculate the goodness score of S_i ($h_{i,P'}$) according to Eq. 4
end for
Identify the k-coloring with the largest value of the goodness score: $S_w = arg\ max\{h_{i,P'} | i = 0, \ldots, p\}$
if $S_w \neq S_0$ **then**
    Replace S_w with S_0: $P = P \cup \{S_0\} \backslash \{S_w\}$
else
    if $rand(0,1) < 0.2$ **then**
        Identify the second worst k-coloring: $S_{sw} = arg\ max\{h_{i,P'} | i = 0, \ldots, p, S_i \neq S_w\}$
        Replace the second worst k-coloring S_{sw} with S_0: $P = P \cup \{S_0\} \backslash \{S_{sw}\}$
    end if
end if

Table 1
Settings of important parameters

Parameters	Section	Description	Values
p	2.1	size of population	20
α	2.4	depth of TS	100,000
m	2.5	number of parents for crossover	r[2,. . . ,6]
P_r	2.6	probability for accepting worse offspring	0.2
λ	2.6	parameter for goodness score function	0.08

during the initialization of the population, together with the smallest distance to other solutions ($D_{i,P}$) for each S_i. Therefore, we need only to calculate the distance between the newly generated offspring S_0 and other solutions in the population. This can be achieved in $O(pk^3)$, since the time complexity of the augmenting path algorithm for calculating the distance between two solutions is $O(k^3)$ where k is the number of used colors and we repeat this process for p times. The process for updating all $D_{i,P'}$ requires $O(p)$. Therefore, the time complexity of our pool updating is $O(pk^3)$. Given the fact that both p and k are small numbers which will not increase with the size of the graph (p is equal to 20 and the maximum value of k is 280 in our experiments), we can say that the time complexity of our pool updating is relatively low.

3 Experimental Results

In this section, we report intensive experimental results of our MACOL algorithm on the well-known DIMACS coloring benchmarks. We compare the results with 11 other state-of-the-art coloring algorithms from the literature.

3.1 Problem Instances and Experimental Protocol

The DIMACS graphs constitute the recognized standard benchmarks in the literature for evaluating the performance of graph coloring algorithms [26]. Particularly, this set of instances include 12 random graphs (DSJC125.x, DSJC250.x, DSJC500.x and DSJC1000.x, x = 1, 5 and 9), 6 flat graphs (flat300_x_0, x=20, 26 and 28; flat1000_x_0, x = 50, 60 and 76), 8 Leighton graphs (le450_15x, le450_25x, x = a, b, c and d), 12 random geometric graphs (R125.x, R250.x, DSJR500.x and R1000.x, x = 1, 1c and 5), 2 huge random graphs (C2000.5 and C4000.5), 2 class scheduling graphs (school1 and school1.nsh) and 1 latin square graph (latin_square_10).

These instances can be classified into two categories [1]: *Easy* graphs and *difficult* graphs. Those instances that can be solved very easily by most modern coloring heuristics are called *easy* graphs, while for *difficult* graphs not every algorithm can reach their chromatic number or the best known results. In this section, we only report our computational results on the set of *difficult* graphs and the results on the *easy* instances are given in the Appendix.

Our MACOL algorithm is programmed in C and compiled using GNU GCC on a PC with 3.4GHz CPU and 2G RAM. To obtain our computational results, each instance is solved 20 times independently with different random seeds (few huge instances are solved 5 times). Each run is stopped when the processing time reaches its timeout limit or a legal k-coloring is found. The timeout limit is set to be 5 CPU hours except for two large graphs *C2000.5* and *C4000.5*, for which a time limit of 120 hours is allowed. All the computational results were obtained without special tuning of the parameters, i.e., all the parameters used in our algorithm are fixed (constant) during the problem solving for *all* the instances considered here. Table 1 gives the descriptions and settings of the important parameters used in our MACOL algorithm, in which $r[2, \ldots, 6]$ denotes a random number from 2 to 6. Notice that fine-tuning these parameters might lead to improved results.

3.2 Computational Results

Our experiment aims to evaluate the MACOL algorithm on the *difficult* DIMACS challenge instances, by comparing its performance with the best known results in the literature. Table 2 summarizes the computational statistics of our MACOL algorithm. Columns 2-4 give the features of the tested instance: the number of vertices (n), the number of edges (n_e) and the density of the

[1] The classification is from: http://www.info.univ-angers.fr/pub/porumbel/graphs/index.html

Table 2
Computational results of MACOL algorithm on the *difficult* DIMACS challenge benchmarks within 5 CPU hours, except for graphs *C2000.5* and *C4000.5*

						MACOL			
Instances	n	n_e	$dens$	k^*	$references$	k	$\#hit$	$iter$	$time(m)$
DSJC250.5	250	15,668	0.50	28	[14,16,24,29,30,34]	28	20/20	2.6×10^6	<1
DSJC500.1	500	12,458	0.10	12	[2,16,24,29,30,34]	12	20/20	5.3×10^7	<1
DSJC500.5	500	62,624	0.50	48	[2,14,16,24,29,34]	48	20/20	2.1×10^6	22
DSJC500.9	500	112,437	0.90	126	[2,16,24,30,34]	126	20/20	1.9×10^8	95
DSJC1000.1	1000	49,629	0.10	20	[2,14,16,24,29,34]	20	16/20	3.5×10^7	108
DSJC1000.5	1000	249,826	0.50	83	[14,29,34]	83	20/20	2.2×10^8	47
DSJC1000.9	1000	449,449	0.90	224[a]	[14,16,24,34]	**223**	18/20	4.5×10^8	150
						224	20/20	7.9×10^7	45
DSJR500.1c	500	121,275	0.97	85	[12,24,29,30]	85	20/20	4.5×10^6	5
DSJR500.5	500	58,862	0.47	122	[29,33]	122	11/20	2.5×10^7	115
R250.5	250	14,849	0.48	65	[2,29,30]	65	20/20	2.7×10^5	4
R1000.1c	1000	485,090	0.97	98	[2,29,30,34]	98	20/20	7.5×10^5	8
R1000.5	1000	238,267	0.48	**234**	[29,33]	245	13/20	1.2×10^9	276
						246	16/20	5.8×10^8	152
						247	20/20	2.3×10^8	76
le450_15c	450	16,680	0.17	15	[2,12,16,24,29,30]	15	20/20	2.0×10^6	3
le450_15d	450	16,750	0.17	15	[2,12,16,24,29,30]	15	20/20	1.8×10^6	5
le450_25c	450	17,343	0.17	25	[2,29,30,34]	25	20/20	1.3×10^7	15
le450_25d	450	17,425	0.17	25	[2,29,30,34]	25	20/20	2.1×10^7	10
flat300_26_0	300	21,633	0.48	26	[12,29,30]	26	20/20	2.6×10^6	4
flat300_28_0	300	21,695	0.48	**28**	[2,24]	29	15/20	1.7×10^7	128
						30	20/20	4.6×10^6	13
flat1000_50_0	1000	245,000	0.49	50	[2,16,24,29,30]	50	20/20	3.2×10^5	5
flat1000_60_0	1000	245,830	0.49	60	[2,16,24,29,30]	60	20/20	6.3×10^5	9
flat1000_76_0	1000	246,708	0.49	82	[29,34]	82	20/20	7.2×10^7	68
						83	20/20	2.7×10^7	27
C2000.5	2000	999,836	0.50	151	[34]	**148**	1/5	8.2×10^8	2156
						149	3/5	6.6×10^8	1875
						150	5/5	3.9×10^8	1385
						151	5/5	3.2×10^8	867
C4000.5	4000	4,000,268	0.50	280	[12]	**272**	3/5	1.2×10^9	7165
						273	4/5	9.3×10^8	5274
						274	4/5	6.5×10^8	3786
						275	5/5	4.8×10^8	2943
						280	5/5	2.3×10^8	1546
latin_sqr_10	900	307,350	0.76	**98**	[30]	99	5/20	6.7×10^7	158
						100	17/20	1.3×10^7	35

Note *a*: Very recently a 223-coloring was independently reported in [35] for graph *DSJC1000.9*.

graph ($dens$). Columns 5 and 6 present the best known results k^* reported in the literature and the corresponding references. In columns 7-9, the computational statistics of our MACOL algorithm are given, including the number of colors obtained (k) [2], the success rate ($\#hit$) and the average number of iterations for reaching the given k ($iter$). If there exist multiple hits on the k-coloring with k, the values $iter$ listed in Table 2 are the average over these multiple hits. Additionally, for indicative purpose, last column shows the average computation time in minutes for reaching the given k. Finally, for six very difficult graphs, Table 2 shows two or more k values which are generally close to our best k.

When comparing with the best known results reported in the literature (col-

[2] Our best results are available at: http://www.info.univ-angers.fr/pub/hao/BestColoring.html

umn 5 in Table 2), one observes that the results obtained by our MACOL algorithm are quite competitive with respect to these previous best results.

For the 6 random graphs (*DSJC*) and the 4 Leighton graphs (*le450*), MACOL reaches the previous best known results easily. In particular, for the graph *DSJC1000.9* we get a *223*-coloring which was only reported very recently in [35].

For the 6 geometric random graphs (*R* and *DSJR*), we obtain the previous best known results except for graph *R1000.5*. The result for *R1000.5* is far behind the best known result. This instance is also the only one for which we obtain worse results than several algorithms. The anti-performance remains to be analyzed.

For the two large random graphs *C2000.5* and *C4000.5*, the previous best known results are k^*=153 and k^*=280 respectively which were obtained by extracting several large stable sets and then solving the residual graph [12]. Without this technique, the results of [12] are even worse. For the graph *C2000.5*, a *151*-coloring is recently obtained by [34]. It is noteworthy to notice that MACOL significantly improves these previous results and obtains a *148*- and *272*-coloring respectively.

For the 5 flat graphs (*flat*), except the graph *flat300_28_0* we can obtain the previous best known results very easily. In the literature, there are only two algorithms that can reach *28*-colorings for this instance. For the last graph *latin_square_10*, we got a *99*-coloring which require one more color than the previous best known coloring. Note that only one algorithm can get *98*-coloring [30].

To summarize, for 3 large instances *DSJC1000.9*, *C2000.5* and *C4000.5*, our MACOL algorithm improves the previous best known results with colorings using respectively 1, 3 and 8 less colors. However, we have achieved worse results than the previous best known results for 3 instances (*R1000.5*, *flat300_28_0* and *latin_square_10*), for which only few algorithms can reach that results. For all the remaining 18 instances, our MACOL algorithm has no difficulty to obtain the best known results. To the best of our knowledge, our MACOL algorithm is the only algorithm that reaches such a performance, as shown in Section 3.3.

Table 2 shows the average iteration number for reaching the given number of colors k (column 9). Note that this indicator is machine independent and can be directly compared with other algorithms. Now we turn our attention to the CPU time. For the majority of the tested graphs, the CPU time for reaching the smallest k-coloring is within 2.5 hours (150 minutes) although a maximum of 5 CPU hours is allocated for each instance. However, for few very hard graphs, the computation time is very long, i.e., several hours to

Table 3
Comparison with the state-of-the-art algorithms in terms of the best results obtained

			Local Search Algorithms				Hybrid Algorithms						
Instances	k^*	k_{best}	[24]	[8]	[6]	[2]	[12]	[30]	[14]	[16]	[29]	[13]	[34]
DSJC250.5	28	**28**	—	**28**	**28**	—	29	**28**	**28**	**28**	**28**	**28**	**28**
DSJC500.1	12	**12**	**12**	13	**12**	**12**	—	—	—	**12**	**12**	**12**	**12**
DSJC500.5	48	**48**	**48**	50	49	**48**	49	49	**48**	**48**	**48**	49	**48**
DSJC500.9	126	**126**	**126**	127	**126**	127	—	—	—	**126**	127	127	**126**
DSJC1000.1	20	**20**	**20**	21	—	**20**	—	—	**20**	**20**	**20**	21	**20**
DSJC1000.5	83	**83**	86	90	89	89	84	89	**83**	84	**83**	88	**83**
DSJC1000.9	224	**223**	224	226	—	226	—	—	224	224	224	228	224
DSJR500.1c	85	**85**	**85**	—	—	**85**	**85**	**85**	—	86	**85**	**85**	—
DSJR500.5	122	**122**	125	—	124	125	130	123	—	127	**122**	**122**	124
R250.5	65	**65**	—	66	—	66	69	**65**	—	—	**65**	**65**	—
R1000.1c	98	**98**	—	**98**	—	**98**	99	**98**	—	—	**98**	**98**	**98**
R1000.5	234	245	—	242	—	248	268	241	—	—	**234**	237	245
le450_15c	15	**15**	**15**	—	**15**	**15**	16	**15**	**15**	**15**	**15**	**15**	—
le450_15d	15	**15**	**15**	—	**15**	**15**	16	**15**	—	**15**	**15**	**15**	—
le450_25c	25	**25**	**25**	—	26	**25**	—	—	26	26	**25**	26	**25**
le450_25d	25	**25**	**25**	—	26	**25**	—	—	—	26	**25**	26	**25**
flat300_26_0	26	**26**	—	**26**	**26**	—	**26**	**26**	—	**26**	**26**	**26**	—
flat300_28_0	28	29	**28**	31	31	**28**	33	31	31	31	31	31	31
flat300_50_0	50	**50**	**50**	**50**	—	**50**	84	**50**	—	**50**	**50**	**50**	—
flat300_60_0	60	**60**	**60**	**60**	—	**60**	84	**60**	—	**60**	**60**	**60**	—
flat300_76_0	82	**82**	85	89	—	87	84	89	83	84	**82**	87	**82**
C2000.5	153	**148**	—	—	—	—	153	165	—	—	—	162	151
C4000.5	280	**272**	—	—	—	—	280	—	—	—	—	301	—
latin_sqr_10	98	99	—	—	99	—	106	**98**	—	104	101	99	—
better	3	—	4	9	6	7	16	6	4	9	4	11	4
equal	18	—	11	5	7	11	2	9	5	10	17	11	11
worse	3	—	1	1	0	1	0	2	0	0	1	2	0
total	24	24	16	15	13	19	18	17	9	19	22	24	15

several days. In order to accept more comparisons with other algorithms under the same stop condition, we also report in Table 2 the results on larger k ($k+i, i \geq 1$). However, for the two largest graphs *C2000.5* and *C4000.5*, our MACOL algorithm need much longer CPU time than for other instances to get a legal k-coloring, since the search space of these two instances are extremely huge.

3.3 Comparison with Other Algorithms

Table 2 assesses the performance of MACOL with respect to the best known results. In this section, we compare the results of our MACOL algorithm with the most effective heuristic algorithms in the literature. Table 3 gives the computational comparison of our MACOL algorithm with 11 state-of-the-art algorithms, which cover the best known results for all the tested instances. Columns 2 and 3 recall the previous best known k^* and the best results found by MACOL (k_{best}). Columns 4 to 14 present the best results obtained by these reference algorithms in the literature, including 4 local search algorithms [2,6,9,24] and 7 hybrid algorithms [12–14,16,29,30,34]. Notice that five of them ([2,16,24,29,34]) were published very recently in 2008 and 2009.

Furthermore, the last four rows show the summary of the comparison between our MACOL algorithm and these reference algorithms. The rows *better*, *equal*, *worse* respectively denotes the number of instances for which our MACOL algorithm gets better, equal and worse results than the corresponding reference algorithm. The row *total* indicates the total number of instances for which the reference algorithm has reported the results. For example, 16 graphs are used in [24] and we get respectively better, equal and worse results than those in [24] for 4, 11 and 1 of them.

If we compare the results of our MACOL algorithm with the 4 local search algorithms, one easily observes that MACOL dominates these local search algorithms (see last four rows of Table 3). For each of these four algorithms, our MACOL algorithm obtains worse results for at most one instance while better results for at least 4 instances.

When comparing with the results of the 7 hybrid algorithms, one observes that our MACOL algorithm is also very competitive. Only two algorithms ([13] and [30]) get better results than MACOL for more than one (actually two) instance. At the same time, our MACOL algorithm reaches better results than the algorithms in [13] and [30] for respectively 11 and 9 instances. Moreover, other five algorithms can only obtain better results than MACOL for at most one instance, while our MACOL gets better results than these algorithms for at least 4 instances.

4 Analysis of the AMPaX Crossover and Pool Updating Strategy

We turn now our attention to analyze the two most important features of the proposed MACOL algorithm, i.e., the adaptive multi-parent crossover operator and the goodness score function for pool updating.

4.1 Multi-Parent Crossover versus 2-Parent Crossover

As indicated in Section 2.5, the AMPaX crossover operator is an extended version of the previous 2-parent GPX crossover in [14]. In order to be sure this extension is meaningful, we carried out additional experiments to compare the performance of the AMPaX crossover operator with different number of parents.

Keeping other ingredients unchanged in our MACOL algorithm, we tested the performance of different m-parent crossover operators (with m=2, m=4 and m=$r[2, \ldots, 6]$ respectively). Notice that m=2 corresponds to the initial GPX

operator of [14]. We observe the average number of iterations for reaching different k with different m. For the purpose of illustration, we choose the large instance *DSJC1000.5* as our test bed.

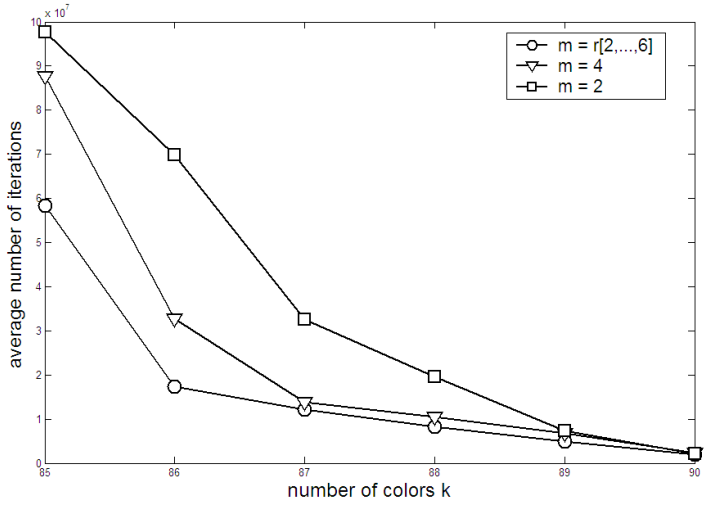

Fig. 2. Comparison between multi-parent crossover and 2-parent crossover

Figure 2 shows the average number of iterations for $k = 85, \ldots, 90$. For each pair (k and m), the problem is solved 10 times. From Figure 2, one easily observes that the average number of iterations for multi-parent crossover (m=4 and m=r[2,...,6]) is much smaller than that for 2-parent crossover (m=2) for different k. We performed a 95% confidence t-test to assess this difference and found that it is statistically significant for all $k < 89$. These results highlight thus the importance of the multi-parent crossover strategy.

When one compares the fixed multi-parent crossover (m=4) and the random multi-parent crossover (m=$r[2, \ldots, 6]$), one finds that when k is large ($k \geq 87$) (i.e. when the problem is not too hard) both strategies performs similarly. However, when k is small ($k < 87$) (i.e., when the problem is harder) the random strategy performs always better than the fixed strategy. This can be explained by the fact that the random strategy provides the search with more diversifications than the fixed strategy and thus helps to find the feasible colorings more easily, especially when the problem becomes difficult (k is small). Let us mention that the same experiments have also been carried out on several other instances shown in Tables 2 and 4, leading to similar observation.

4.2 Goodness Score Function for Pool Updating

In order to evaluate the distance-and-quality based goodness score function (denoted by *DisQual*) for pool updating, we compare *DisQual* with previous

goodness score functions proposed in the literature. The traditional goodness score function for pool updating is only based on the quality f: one is to replace the worst k-coloring in the population with the new offspring (denoted by *PoolWorst*), the other is to replace the worst parent (denoted by *ParentWorst*).

Once again we kept other ingredients unchanged in our MACOL algorithm and observe the average number of iterations for reaching a given legal k-coloring with these three different pool updating strategies. Figure 3 shows the average number of iterations for $k = 85, \ldots, 90$ on *DSJC1000.5*. The experiment is also based on 10 independent runs.

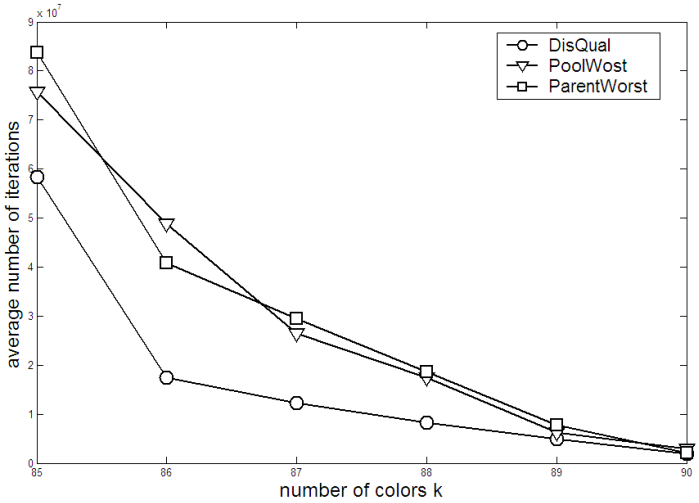

Fig. 3. Comparison between different goodness score functions

One easily finds that MACOL with our new goodness score function converges more quickly towards a legal k-coloring than with the tradition pool updating strategies *PoolWorst* and *ParentWorst*, which implies that considering the diversity of the population as well as the quality for pool updating is essential for our algorithm. On the other hand, *PoolWost* and *ParentWost* strategies perform similarly. Once again, this observation was also confirmed with other graphs of Tables 2 and 4.

5 Conclusion

In this paper, we have presented MACOL, a memetic algorithm for the graph coloring problem. The proposed algorithm integrates a number of original features. First, we have proposed an Adaptive Multi-Parent crossover operator. Second, based on the definition of the distance between two k-colorings, we introduced the distance between a k-coloring and a population. Third, we

proposed a new goodness score function considering both solution quality and diversity of individuals. These strategies provide the algorithm with a good tradeoff between intensification and diversification.

We have shown that this hybrid heuristic obtains highly competitive results on a large number of DIMACS challenge benchmark graphs. Compared with a number of reference algorithms, our MACOL algorithm is quite effective and has a robust behavior on all the considered instances.

Additionally, we investigated several essential parts of our proposed algorithm. We first carried out experiments to demonstrate that the multi-parent crossover is more powerful than the traditional 2-parent crossover in this context. Moreover, we showed that the proposed distance-and-quality based goodness score function for pool updating is quite useful.

The success of the MACOL algorithm on graph coloring problem reminds us that it is essential to introduce meaningful diversification mechanisms and highlight the tradeoff between intensification and diversification in designing heuristic search algorithms. Following this spirit, we hope to design even more robust and effective heuristic algorithms for solving graph coloring problem and other optimization problems.

Acknowledgment

We are grateful for comments by the referees that have significantly improved the paper. This work was partially supported by "*Angers Loire Métropole*" and the Region of "*Pays de la Loire*" within the MILES and RADAPOP Projects.

Appendix A:

In this appendix, we report our computational results on the *easy* set of the DIMACS challenge benchmark graphs with the time limit of 2 CPU hours, as shown in Table 4. The symbols in this table are the same as those in Table 2.

References

[1] C. Avanthay, A. Hertz, N. Zufferey, A variable neighborhood search for graph coloring, European Journal of Operational Research 151(2) (2003) 379–388.

Table 4
Computational results of MACOL algorithm on the *easy* DIMACS benchmarks

					MACOL			
Instances	n	n_e	$dens$	k^*	k	$\#hit$	$iter$	$time(m)$
DSJC125.1	125	736	0.09	5	5	10/10	1.4×10^5	1
DSJC125.5	125	3,891	0.50	17	17	10/10	4.8×10^4	3
DSJC125.9	125	6,961	0.89	44	44	10/10	2.4×10^6	4
DSJC250.1	250	3,218	0.10	8	8	10/10	6.9×10^5	2
DSJC250.9	250	27,897	0.90	72	72	10/10	5.5×10^6	3
R125.1	125	209	0.03	5	5	10/10	3.7×10^5	2
R125.1c	125	7,501	0.97	46	46	10/10	2.8×10^6	5
R125.5	125	3,838	0.50	36	36	10/10	3.2×10^4	1
R250.1	250	867	0.03	8	8	10/10	1.5×10^6	5
R250.1c	250	30,227	0.97	64	64	10/10	2.8×10^6	4
DSJR500.1	500	3,555	0.03	12	12	10/10	3.3×10^5	4
R1000.1	1000	14,348	0.03	20	20	10/10	2.9×10^5	2
le450_15a	450	8,168	0.08	15	15	10/10	2.7×10^5	2
le450_15b	450	8,169	0.08	15	15	10/10	3.5×10^5	2
le450_25a	450	8,260	0.08	25	25	10/10	1.8×10^5	4
le450_25b	450	8,263	0.08	25	25	10/10	2.8×10^6	3
school1	385	19,095	0.26	14	14	10/10	8.8×10^5	6
school1_nsh	352	14,612	0.24	14	14	10/10	7.3×10^5	4
flat300_20_0	300	21,375	0.48	20	20	10/10	1.7×10^6	1

[2] I. Blöchliger, N. Zufferey, A graph coloring heuristic using partial solutions and a reactive tabu scheme, Computers and Operations Research 35(3) (2008) 960-975.

[3] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM 22(4) (1979) 251–256.

[4] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyper heuristic for timetabling problems, European Journal of Operational Research 176 (2007) 177–192.

[5] M. Chams, A. Hertz, D. de Werra, Some experiments with simulated annealing for coloring graphs, European Journal of Operational Research 32 (1987) 260-266.

[6] M. Chiarandini, T. Stützle, An application of iterated local search to graph coloring. In: D.S. Johnson, A. Mehrotra, M. Trick, editors, Proc. of the Computational Symposium on Graph Coloring and its Generalizations, Ithaca, New York, USA, 112–125, 2002.

[7] D. de Werra, C. Eisenbeis, S. Lelait, B. Marmol, On a graph-theoretical model for cyclic register allocation, Discrete Applied Mathematics 93(2-3) (1999) 191–203.

[8] R. Dorne, J.K. Hao, Tabu search for graph coloring, T-colorings and set T-colorings. Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, 77-92, 1998.

[9] R. Dorne, J.K. Hao, A new genetic local search algorithm for graph coloring, Lecture Notes in Computer Science 1498 (1998) 745–754.

[10] E. Falkenauer, A hybrid grouping genetic algorithm for bin packing, Journal of Heuristics 2(1) (1996) 5-30.

[11] C. Fleurent, J.A. Ferland, Genetic and hybrid algorithms for graph coloring, Annals of Operations Research 63 (1996) 437-461.

[12] C. Fleurent, J.A. Ferland, Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability, In: [26], (1996) 619-652.

[13] N. Funabiki, T. Higashino, A minimal-state processing search algorithm for graph coloring problems, IEICE Transaction Fundamentals E83-A (2000) 1420-1430.

[14] P. Galinier, J.K. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization 3(4) (1999) 379-397.

[15] P. Galinier, A. Hertz, A survey of local search methods for graph coloring, Computers and Operations Research 33(9) (2006) 2547–2562.

[16] P. Galinier, A. Hertz, N. Zufferey, An adaptive memory algorithm for the K-colouring problem, Discrete Applied Mathematics 156(2) (2008) 267–279.

[17] M. Gamache, A. Hertz, J.O. Ouellet, A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding, Computers and Operations Research 34(8) (2007) 2384–2395.

[18] M.R. Garey, D.S. Johnson, H.C. So, An application of graph coloring to printed circuit testing, IEEE Transactions on Circuits and Systems 23 (1976) 591–599.

[19] M.R Garey, D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness. W.H. Freeman and Company, San Francisco, 1979.

[20] C. Glass, Bag rationalisation for a food manufacturer, Journal of the Operational Research Society 53 (2002) 544–551.

[21] F. Glover, M. Parker, J. Ryan, Coloring by tabu branch and bound, In: [26], (1996) 285–307.

[22] J.P. Hamiez, J.K. Hao, Scatter search for graph coloring, Lecture Notes in Computer Science 2310 (2002) 168–179, Springer-Verlag.

[23] A. Hertz, D. de Werra, Using tabu search techniques for graph coloring, Computing 39 (1987) 345-351.

[24] A. Hertz, M. Plumettaz, N. Zufferey, Variable space search for graph coloring, Discrete Applied Mathematics 156(13) (2008) 2551–2560.

[25] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. Operations Research 39(3) (1991) 378-406.

[26] D.S. Johnson, M. Trick, Cliques, coloring, and satisfiability: second DIMACS implementation challenge, American Mathematical Society, Volume 26, 1996.

[27] M. Laguna, R. Martí, A GRASP for coloring sparse graphs, Computational optimization and applications 19(2) (2001) 165–178.

[28] F.T. Leighton, A graph coloring algorithm for large scheduling problems, Journal of Research of the National Bureau of Standards 84(6) (1979) 489–506.

[29] E. Malaguti, M. Monaci, P. Toth, A metaheuristic approach for the vertex coloring problem, INFORMS Journal on Computing 20(2) (2008) 302–316.

[30] C. Morgenstern, Distributed coloration neighborhood search, In: [26], (1996) 335-357.

[31] P. Moscato, Memetic algorithms: a short introduction. in: New Ideas in Optimization, Mcgraw-Hill Ltd., Maidenhead, UK. 219–234, 1999.

[32] C.H. Papadimitriou, K. Steiglitz, Combinatorial optimization: algorithms and complexity, Prentice-Hall, Inc., 1982.

[33] S. Prestwich, Coloration neighbourhood search with forward checking, Annals of Mathematics and Artificial Intelligence 34(4) (2002) 327–340.

[34] D.C. Porumbel, J.K. Hao, P. Kuntz, Diversity control and multi-parent recombination for evolutionary graph coloring algorithms, C. Cotta, P. Cowling (Eds.): EvoCOP 2009, Lecture Notes in Computer Science 5482 (2009) 121-132, Springer-Verlag.

[35] D.C. Porumbel, J.K. Hao, P. Kuntz, A search space cartography for guiding graph coloring heuristics, Accepted by Computers and Operations Research, July, 2009.

[36] J.C. Setubal, Sequential and parallel experimental results with bipartite matching algorithms, Technical Report EC-96-09, Institute of Computing, University of Campinas, Brasil 1996.

[37] D.H. Smith, S. Hurley, S.U. Thiel, Improving heuristics for the frequency assignment problem, European Journal of Operational Research 107(1) (1998) 76–86.

[38] N. Zufferey, P. Amstutz, P. Giaccari, Graph colouring approaches for a satellite range scheduling problem, Journal of Scheduling 11(4) (2008) 263–277.