Improving probability learning based local search for graph coloring

Yangming Zhou [a,b], Béatrice Duval [b], Jin-Kao Hao [b,c,*]

[a] *Glorious Sun School of Business and Management, Donghua University, No.1882, Yan'an Road (West), Shanghai, China*

[b] *LERIA, Université d'Angers, 2 boulevard Lavoisier, 49045 Angers, France*

[c] *Institut Universitaire de France, 1 rue Descartes, 75231 Paris, France*

Applied Soft Computing
https://doi.org/10.1016/j.asoc.2018.01.027

Abstract

This paper presents an improved probability learning based local search algorithm for the well-known graph coloring problem. The algorithm iterates through three distinct phases: a starting coloring generation phase based on a probability matrix, a heuristic coloring improvement phase and a learning based probability updating phase. The method maintains a dynamically updated probability matrix which specifies the chance for a vertex to belong to each color group. To explore the specific feature of the graph coloring problem where color groups are interchangeable and to avoid the difficulty posed by symmetric solutions, a group matching procedure is used to find the group-to-group correspondence between a starting coloring and its improved coloring. Additionally, by considering the optimization phase as a black box, we adopt the popular tabu search coloring procedure for the coloring improvement phase. We show extensive computational results on the well-known DIMACS benchmark instances and comparisons with state-of-the-art coloring algorithms.

Key words: Graph coloring; grouping problems; learning-based optimization; tabu search; heuristics.

* Corresponding author.
Email addresses: zhou.yangming@yahoo.com (Yangming Zhou), beatrice.duval@univ-angers.fr (Béatrice Duval), jin-kao.hao@univ-angers.fr (Jin-Kao Hao).

1 Introduction

Given an undirected graph $G = (V, E)$ with vertex set V and edge set E, a (legal) k-coloring of G is a function $g : V \to \{1, 2, \ldots, k\}$ such that $g(u) \neq g(v)$ for all edges $(u, v) \in E$. In other words, two adjacent vertices (i.e., linked by an edge) must be colored with two different colors. Given a vertex u, $g(u)$ is called the color or the color group of u. The *graph coloring problem* (GCP) involves finding a k-coloring of G with k minimum. The minimum number of colors required to color G is known as the *chromatic number* of G and denoted by $\chi(G)$ or χ.

Let g_i denote the group of vertices receiving color i. Then a k-coloring S can also be considered as a partition of V into k color groups (called independent sets) $S = \{g_1, \ldots, g_k\}$ such that no two adjacent vertices belong to the same color group. It is important to note that GCP has the following property concerning symmetric solutions. Let $\pi : \{1, 2, \ldots, k\} \to \{1, 2, \ldots, k\}$ be an arbitrary permutation of the set of colors and let $S = \{g_1, \ldots, g_k\}$ be a k-coloring. Then $S' = \{g_{\pi(1)}, \ldots, g_{\pi(k)}\}$ is also a k-coloring which is strictly equivalent to S. In other words, S and S' are two symmetric solutions representing exactly the same k-coloring. This property implies that the names of the color groups in a coloring are irrelevant and interchangeable.

GCP is a very popular NP-hard combinatorial optimization problem in graph theory [18] and has attracted much attention in the literature. GCP arises naturally in a wide variety of real-world applications, such as register allocation [8], timetabling [5,9], frequency assignment [17,21,23], and scheduling [29,54]. It was one of the three target problems of several International Competitions including the well-known Second DIMACS Implementation Challenge on Maximum Clique, Graph Coloring, and Satisfiability.

In recent years, research on combining learning techniques and heuristic algorithms has received increasing attention from the research community [48]. For example, Boyan and Moore [3] proposed the STAGE algorithm to learn an evaluation function which predicts the outcome of a local search algorithm. Hutter et al. [27] used machine learning techniques (random forests and approximate Gaussian process) to build a prediction model of the algorithm's runtime as a function of problem-specific instance features. Wang and Tang [47] combined clustering technique and statistical learning within a memetic algorithm for a multi-objective flowshop scheduling problem. Zhou et al. [52] proposed an opposition-based memetic algorithm for solving the maximum diversity problem, which integrates the concept of opposition-based learning into the memetic search framework.

Recently, Zhou et al. [51] introduced a general-purpose local search method-

ology called Reinforcement learning based Local Search (RLS) for solving *grouping problems*. RLS employs a dynamically updated probability matrix to generate starting solutions for a descent-based local search procedure. However, as indicated in Section 3, applying the original RLS per se to the graph coloring problem leads to two important limitations. First, as a specific grouping problem, GCP has the particularity that the color groups of a solution are interchangeable and the naming of the groups is irrelevant. Second, the descent-based search proposed in the initial RLS is quite limited, compared to a number of powerful graph coloring algorithms in the literature [13,15,34] (See Section 2). These two observations constitute the main motivations of this work. We aim to propose an enhanced probability learning based local search algorithm for graph coloring, by reinforcing the original RLS, which remains an innovative and appealing approach due to its learning feature and simplicity.

We summarize the main contributions of this work as follows.

- The proposed probability learning based local search algorithm (PLSCOL) proposed in this work considers the specific feature of the graph coloring problem and brings two enhancements to the RLS approach. First, to avoid the difficulty posed by symmetric solutions, PLSCOL uses a group matching procedure to identify the group-to-group relation between the starting solution and its improved solution regardless of the numbering of the groups in both solutions (Section 4.2). Second, to ensure an effective coloring improvement, we adopt a popular coloring algorithm based on tabu search (Section 4.3) to replace the (basic) descent-based local search of the RLS method. By following the general RLS approach (Section 3), PLSCOL uses a learning based probability matrix to generate starting solutions for the tabu coloring algorithm.
- We assess the performance of the proposed PLSCOL approach on the benchmark instances of the well-known second DIMACS Implementation Challenge. The computational results indicate that PLSCOL performs remarkably well compared to state-of-the-art local search coloring algorithms like [1,7,26]. PLSCOL also proves to be competitive even when it is compared to more complex and sophisticated hybrid algorithms [14,16,30,32,35,38,44,45], which reported most of the current best-known results for the most difficult DIMACS benchmark instances.
- The availability of the code of our PLSCOL algorithm (see Section 5.2) contributes favorably to future research on GCP and related problems. Specifically, the code can be used to perform comparative studies or solve other problems that can be formulated as GCP. This can also help to promote more research on learning based optimization methods, which constitutes a promising, yet immature domain.

The rest of the paper is organized as follows. In the next section, we conduct

a brief review of some representative heuristic algorithms for GCP. In Section 3, we review the general framework of the RLS method for grouping problems. Section 4 presents our improved PLSCOL algorithm for GCP. Section 5 shows extensive computational results and comparisons on the DIMACS challenge benchmark instances. In Section 6, we analyze and discuss several important features of the proposed alorithm. Finally, conclusions and future work are provided in Section 7.

2 Literature review on heuristic approaches for GCP

In this section, we provide a brief review of the existing heuristic algorithms for GCP. Some of the best-performing and most representative algorithms are used as reference algorithms in our comparative study. As indicated in [13], due to significant structure differences of these instances, none of the existing GCP approaches can be considered as the most effective method for all DIMACS benchmark instances.

Given the NP-hardness of GCP, exact algorithms are usually effective only for solving small graphs or graphs of specific classes. In fact, some graphs with as few as 150 vertices cannot be solved optimally by any exact algorithm [33,53]. To deal with large and difficult graphs, heuristic algorithms are preferred to solve the problem approximately. Comprehensive reviews of the graph coloring algorithms can be found in [13,15,34]. In what follows, we focus on some representative heuristic-based coloring algorithms.

- `Constructive approaches` generally construct the color groups by iteratively adding a vertex at one time to a color group until a complete coloring is reached. At each iteration, there are two steps: the next vertex to be colored is chosen at first, and then this vertex is assigned to a color group. DSATUR [4] and RLF [29] are two well-known greedy algorithms which employ refined rules to dynamically determine the next vertex to color. These greedy heuristic algorithms are usually fast, but they tend to need much more colors than the chromatic number to color a graph. Consequently, they are often used as initialization procedures in hybrid algorithms.
- `Local search approaches` start from an initial solution and try to improve the coloring by performing local changes. In particular, tabu search is known as one of the most popular local search method for GCP [25]. It is often used as a subroutine in hybrid algorithms, such as the hybrid evolutionary algorithm [12,14,30,35,38]. However, local search algorithms are often substantially limited by the fact that they do not exploit enough global information (e.g., solution symmetry), and do not compete well with hybrid population-based algorithms. A thorough survey of local search algorithms for graph coloring can be found in [15].

- `Population-based hybrid approaches` work with multiple solutions that can be manipulated by some selection and recombination operators. To maintain the population diversity which is critical to avoid a premature convergence, population-based algorithms usually integrate dedicated diversity preservation mechanisms which require the computation of a suitable distance metric between solutions [22]. For example, hybrid algorithms [30,35,38,44] are among the most effective approaches for graph coloring, which have reported the best solutions on most of the difficult DIMACS instances. Nevertheless, population-based hybrid algorithms have the disadvantage of being more complex in design and more sophisticated in implementation. Moreover, their success greatly depends on the use of a meaningful recombination operator, an effective local optimization procedure and a mechanism for maintaining population diversity.
- "`Reduce and solve`" `approaches` combines a *preprocessing phase* and a *coloring phase*. The preprocessing phase identifies and extracts some vertices (typically independent sets) from the original graph to obtain a reduced graph, while the subsequent coloring phase determines a proper coloring for the reduced graph. Empirical results showed that "reduce-and-solve" approaches achieve a remarkable performance on some large graphs [24,49]. "Reduce and solve" approaches are designed for solving large graphs and are less suitable for small and medium-scale graphs. Moreover, their success depends on the vertices extraction procedure and the underlying coloring algorithm.
- `Other approaches` which cannot be classified into the previous categories include for instance a method that encodes GCP as a boolean satisfiability problem [2], a modified cuckoo algorithm [31], a grouping hyper-heuristic algorithm [11] and a multi-agent based distributed algorithm [41].

3 The RLS method for grouping problems [51]

The RLS method proposed in [51] is a general framework designed for solving grouping problems. Generally, a grouping problem aims to group a set of given items into a fixed or variable number of groups while respecting some specific requirements. Typical examples of grouping problems include bin-packing, data clustering and graph coloring. The basic idea of RLS is to iterate through a group selection phase (to generate a starting solution S according to a *probability matrix* P that indicates for each item its chance to belong to each group), a descent-based local search (DB-LS) phase (to obtain an improved solution S' from S), and a probability learning phase [1]. During the probability learning phase, RLS compares the starting solution S and the

[1] Instead of the term "reinforcement learning" initially used in [51], we adopt in this work the more explicit and appropriate term "probability learning".

improved solution S' to check whether each item moved from its original group to a new group in S' or stayed in its original group of S. Then RLS adjusts the probability matrix accordingly by the following rule. If an item stayed in its original group, the selected group (called correct group) is rewarded for the item; if the item moved to a new group in S', the discarded group (called incorrect group) is penalized and the new group (called expected group) for the item is compensated.

Algorithm 1 General RLS framework for grouping problem

1: **Input**: instance G and number of available groups k;
2: **Output**: the best solution found so far;
3: $P \leftarrow P_0$ /* Initialise the probability matrix */
4: **while** stopping condition not reached **do**
5: $S \leftarrow GroupSelection(P)$; /* generate a starting solution */
6: $S' \leftarrow DB\text{-}LS(S)$; /* find a local optimal solution */
/* Probability learning phase */
7: $P \leftarrow ProbabilityUpdating(S, S', P)$; /* learn a new probability matrix */
8: $P \leftarrow ProbabilitySmoothing(P)$; /* conduct a probability smoothing */
9: **end while**

Algorithm 1 shows the general RLS framework. To apply RLS to a grouping problem, three procedures need to be specified. The *GroupSelection* procedure (line 5) generates a starting solution S. The *DB-LS* procedure (line 6) aims to obtain a local optimum S' from the starting solution S. During the probability learning phase (lines 7-8), *ProbabilityUpdating* updates the probability matrix P by comparing the starting solution S and its improved solution S', while *ProbabilitySmoothing* erases old decisions that become useless and may even mislead the search. The modifications of the probability matrix rely on information about group changes of the items. Using the updated probability matrix, a new starting solution is built for the next round of the *DB-LS* procedure (this is also called a generation). A schematic diagram of the whole RLS process is provided in Figure 1.

The probability matrix P of size $n \times k$ (n and k being the number of items and the number of available groups respectively) defines the chance for an item to select a given group. In other words, element p_{ij} denotes the probability that the item v_i should select the group g_j. Therefore, the i-th row of the probability matrix P (denoted as p_i) defines the probability vector of the i-th item with respect to each available group. Initially, all items uniformly select each group, i.e., $p_{ij} = 1/k, \forall i \in \{1, 2, ..., n\}, \forall j \in \{1, 2, ..., k\}$. This matrix evolves at each generation of the RLS method.

At generation t, each item v_i selects one suitable group g_j according to a hybrid group selection strategy which combines greedy selection and random selection. Specifically, with a small noise probability ω, item v_i selects its group at random; with probability $(1-\omega)$, item v_i selects the group g_m such that $m =$

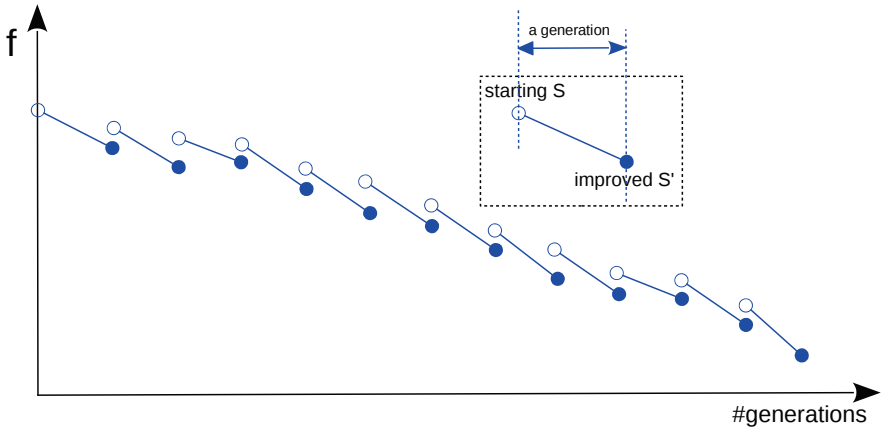

Fig. 1. A schematic diagram of the RLS framework introduced in [51].

$\arg\max_{j\in\{1,\ldots,k\}}\{p_{ij}\}$, i.e., the group with the maximum associated probability for v_i. As shown in Section 4.3.3 of [51], this hybrid selection strategy has the advantage of being flexible by switching back and forth between greediness and randomness, and allows the algorithm to occasionally move away from being too greedy. This hybrid selection strategy is motivated by the following consideration. Probability based selection is a greedy strategy (item i selects group j with probability p_{ij} maximum). However, since the probability values are learned progressively and convey heuristic information, applying such a strategy all the time could be too greedy and misguide the search to wrong directions. So we occasionally apply a random selection (item i selects group j with equal probability) to alleviate the greedy selection.

Once a new starting solution S is obtained from the group selection phase, the $DB-LS$ procedure is invoked. $DB-LS$ iteratively makes transitions from the current solution to a neighboring solution according to a given neighborhood relation such that each transition leads to a better solution [37]. This iterative local search improvement process stops when no improved solution exists in the neighborhood in which case the current solution corresponds to a local optimum S' with respect to the neighborhood.

After the local search phase, the probability learning procedure is applied to update the probability matrix. Specifically, for each item v_i, if it stayed in its original group (say g_u), we reward the selected group g_u and update its probability vector p_i with the award factor α, as shown in Eq. (1) (t is the current generation number).

$$p_{ij}(t+1) = \begin{cases} \alpha + (1-\alpha)p_{ij}(t) & j = u \\ (1-\alpha)p_{ij}(t) & \text{otherwise.} \end{cases} \tag{1}$$

If the item v_i moved from its original group g_u of solution S to a new group

$g_v(v \neq u)$ of the improved solution S', we penalize the former group g_u, compensate the new group g_v, and update its probability vector p_i with the penalization factor β and the compensation factor γ, as shown in Eq. (2).

$$p_{ij}(t+1) = \begin{cases} (1-\gamma)(1-\beta)p_{ij}(t) & j = u \\ \gamma + (1-\gamma)\frac{\beta}{k-1} + (1-\gamma)(1-\beta)p_{ij}(t) & j = v \\ (1-\gamma)\frac{\beta}{k-1} + (1-\gamma)(1-\beta)p_{ij}(t) & \text{otherwise.} \end{cases} \quad (2)$$

Our probability updating scheme is inspired by learning automata (LA) [36], which is a kind of policy iteration method where the optimal policy is directly determined in the space of candidate policies [42]. In LA, an action probability vector is maintained and updated according to a specific probability learning technique or reinforcement scheme. Well-known reinforcement schemes include linear reward-penalty and linear reward-inaction and aim to increase the probability of selecting an action in the event of success and decrease it in the case of failure [48].

Unlike these general schemes, our probability updating scheme not only rewards the correct group (by factor α) and penalizes the incorrect group (by factor β), but also compensates the expected group (by factor γ). With the help of learning scheme (1) and (2), the probability of correct groups will increase, and other probabilities will decrease.

Additionally, a probability smoothing technique is employed to reduce the group probabilities periodically. Once the probability of a group in a probability vector achieves a given threshold (i.e., p_0), it is reduced by multiplying a smoothing coefficient (i.e., $\rho < 1$) to forget some earlier decisions. Specifically, we suppose $p_{im} > p_0, 1 \leqslant m \leqslant k$, then we have $p'_{im} = \rho * p_{im}$, and the probabilities of selecting other groups are kept constant, i.e., $p'_{ij} = p_{ij}, j \neq m$ and $1 \leqslant j \leqslant k$. To make sure that the probability vector p_i has the sum value of one after probability smoothing, we scale all k probabilities $p'_{ij}(1 \leqslant j \leqslant k)$ by dividing them with a coefficient $1-(1-\rho)*p_{im}$. More details can be found in [51].

4 Improving probability learning based local search for GCP

Following many coloring algorithms [13–15,25,30,32,38], we approximate GCP by solving a series of k-coloring problems. For a given graph and a fixed number of k colors, we try to find a legal k-coloring. If a legal k-coloring is found, we set $k = k - 1$ and try to solve the new k-coloring problem. We repeat this process until no legal k-coloring can be found, in which case we return $k + 1$

(for which a legal coloring has been found) as an approximation (upper bound) of the chromatic number of the given graph.

4.1 Main scheme

We first define the search space and the evaluation function used by the PLSCOL algorithm. For a given graph $G = (V, E)$ with k available colors, the search space Ω contains all possible (both legal or illegal) k-colorings (candidate solutions). A candidate solution in Ω can be represented by $S = \{g_1, g_2, \ldots, g_k\}$ such that g_i is the group of vertices receiving color i. The evaluation function $f(S)$ is used to count the conflicting edges induced by S.

$$f(S) = \sum_{\{u,v\} \in E} \delta(u, v) \tag{3}$$

where $\delta(u, v) = 1$, if $u \in g_i, v \in g_j$ and $i = j$, and otherwise $\delta(u, v) = 0$. Thus, a candidate solution S is a legal k-coloring when $f(S) = 0$. The objective of PLSCOL is to minimize f, i.e., the number of conflicting edges to find a legal k-coloring in the search space.

Algorithm 2 presents the PLSCOL algorithm. Initially, by setting the probability $p_{ij} = 1/k, i \in \{1, \ldots, n\}, j \in \{1, \ldots, k\}$, each group is selected uniformly by each item (lines 3-7). A starting solution S is produced by the hybrid selection strategy based on the current probability matrix P (line 9). The tabu search procedure is used to improve the starting solution S to a new solution S' (line 10, see Section 4.3). The *GroupMatching* procedure is then applied to find a maximum weight matching between the starting solution S and its improved solution S' (line 14, see Section 4.2), followed by the *ProbabilityUpdating* procedure (to update the probability matrix P according to the matching results), and the *ProbabilitySmoothing* procedure (to forget some earlier decisions).

PLSCOL uses probability learning to generate starting solutions for tabu search. Compared to the original probability learning based local search approach RLS [51], PLSCOL introduces two improvements. Considering the specific feature of GCP where color groups are interchangeable, we introduce a group matching procedure to establish a group-to-group correspondence between a starting solution and its improved solution (Section 4.2). Also, to seek a legal coloring, we replace the basic descent-based local search of RLS by tabu search based coloring algorithm (Section 4.3). Finally, we notice that probability learning has also been used to learn evaluation function of a local search algorithm [3], select search operators of a genetic algorithm [20] and a multi-agent optimization approach [40].

Algorithm 2 Pseudo-code of PLSCOL for k-coloring

Input: instance G = (V, E) and number of available colors k;
Output: the best k-coloring S^* found so far;
/* Initialise the probability matrix $P_{n\times k}$ */
for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, k$ **do**
    $p_{ij} = 1/k$
  end for
end for
while stopping condition not reached **do**
  $S \leftarrow GroupSelection(P)$; /* generate a starting solution */
  $S' \leftarrow TabuSearch(S)$; /* find a local optimal solution */
  if $f(S') < f(S^*)$ **then**
    $S^* = S'$
  end if
  $feedback \leftarrow GroupMatching(S, S')$ /* make a maximum matching */
  $P \leftarrow ProbabilityUpdating(feedback, P)$; /* learn a new probability matrix */
  $P \leftarrow ProbabilitySmoothing(P)$; /* conduct a probability smoothing */
end while

4.2 Group matching

As a particular grouping problem, the graph coloring problem is characterized by the fact that the numberings of the groups in a solution are irrelevant. For instance, for a graph with five vertices {a,b,c,d,e}, suppose that we are given a solution (coloring) that assigns a color to {a,b,c} and another color to {d,e}. It should be clear that what really characterizes this coloring is not the naming/numbering of the colors used, but is the fact that some vertices belong to the same group and some other vertices cannot belong to the same group due to the coloring constraints. As such, we can name {a,b,c} by 'group 1' and {d,e} by 'group 2' or reversely name {a,b,c} by 'group 2' and {d,e} by 'group 1', these two different group namings represent the same coloring. This symmetric feature of colorings is known to represent a real difficulty for many coloring algorithms [14,38].

In the original RLS approach, the learning phase was based on a direct comparison of the groups between the starting solution and its improved solution, by checking for each vertice its starting group number and the new group number. This approach has the advantage of easy implementation and remains meaningful within the RLS method. Indeed, RLS does not assume any particular feature (e.g., symmetry of solutions in the case of GCP) of the given grouping problem. Moreover, when a strict descent-based local search is used to obtain an improved solution (i.e., local optimum), only a small portion of the vertices change their groups. It suffices to compare the groups of the two solutions to identify the vertices that changed their group. However, the

situation is considerably different when tabu search (or another optimization algorithm) is used to obtain an improved coloring, since many vertices can change their groups during the search process. This difficulty is further accentuated for GCP due to its symmetric feature of colorings, making it irrelevant to compare the group numbers between the starting and improved solutions.

To illustrate this point, consider the example of Fig. 2. From the same starting 8-coloring S with $f(S) = 568$, the descent-based local search ends with an improved (but always illegal) 8-coloring S'_{ds} with $f(S') = 10$, while the tabu search procedure successfully obtains an improved 8-coloring S'_{ts} with $f(S') = 0$. In this figure, the new groups in the two improved solutions are marked by red color and underlined.

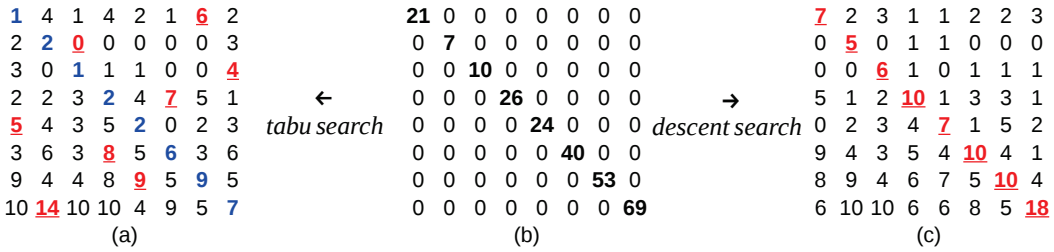

Fig. 2. Distribution of vertices of solutions on instance DSJC250.1. The value on i-th row and j-th column represents the number of vertices whose color have changed from color i to color j. (a) an improved solution S'_{ts} (with $f(S'_{ts}) = 0$) obtained by tabu search, (b) a starting solution S (with $f(S) = 568$), and (c) an improved solution S'_{ds} with $f(S'_{ds}) = 10$ obtained by descent-based local search.

By comparing the distributions of vertices of the two improved solutions from descent-based search and tabu search, we observe a clear one-to-one correspondence between the starting solution S and its improved solution S'_{ds} obtained by descent-based search, i.e., $g_1 \leftrightarrow g'_1, g_2 \leftrightarrow g'_2, g_3 \leftrightarrow g'_3, g_4 \leftrightarrow g'_4, g_5 \leftrightarrow g'_5, g_6 \leftrightarrow g'_6, g_7 \leftrightarrow g'_7, g_8 \leftrightarrow g'_8$. However, it is difficult to find such a relationship between S and S'_{ts} obtained by tabu search. Indeed, from the same starting solution, much more changes have been made by tabu search. For example, there are 21 vertices colored by color 1 in the starting solution S; after improving to S'_{ds} by descent-based search, 7 vertices keep their original color, the remaining 2, 3, 1, 1, 2, 2, 3 vertices respectively changed to new colors 2,3,4,5,6,7,8. When S is improved to S'_{ts} by tabu search, only 1 vertex of these 21 vertices keeps its original color, the remaining 4, 1, 4, 2, 1, 6, 2 vertices have moved to color groups 2,3,4,5,6,7,8.

Now if we examine the groups of vertices in S and S'_{ts} regardless of the numbering of the groups, we can identify the following group-to-group correspondence between S and S'_{ts}: $g_1 \leftrightarrow g'_7, g_2 \leftrightarrow g'_3, g_3 \leftrightarrow g'_8, g_4 \leftrightarrow g'_6, g_5 \leftrightarrow g'_1, g_6 \leftrightarrow g'_4, g_7 \leftrightarrow g'_5, g_8 \leftrightarrow g'_2$. Indeed, this correspondence can be achieved by finding a maximum weight matching between the two compared solutions as follows.

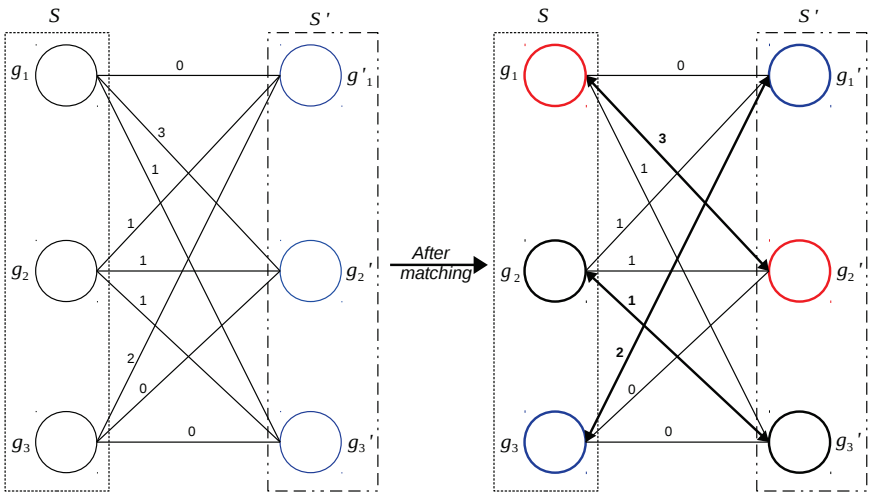

Fig. 3. (a) A complete bipartite graph with the weights between two groups $\omega_{g_i,g'_j} = |g_i \cap g'_j|$ and (b) The corresponding maximum weight complete matching with the maximum weight of 6).

Specifically, to identify such a relationship between a starting solution S and its improved solution S', we sequentially match each group of S with each group of S'. Then we build a complete bipartite graph $G = (V_1, V_2, E)$, where V_1 and V_2 represent respectively the k groups of solution S and S'. Each edge $(g_i, g'_j) \in E$ is associated with a weight $w_{g_i,g'_j} = |g_i \cap g'_j|$, which is defined as the number of common vertices in group g_i of S and g'_j of S'. Based on the bipartite graph, we can find a maximum weight matching with the well-known Hungarian algorithm [28] in $O(k^3)$. Fig. 3 shows an illustrative example of matching two solutions $S = \{(v_1, v_3, v_7, v_8), (v_2, v_4, v_5), (v_6, v_9)\}$ and $S' = \{(v_5, v_6, v_9), (v_2, v_3, v_7, v_8), (v_1, v_4)\}$. The group matching procedure finds the following one-to-one group relation between these two solutions: $g_1 \leftrightarrow g'_2$, $g_2 \leftrightarrow g'_3$ and $g_3 \leftrightarrow g'_1$.

Finally, note that the group matching procedure just requires a starting solution S and an improved solution S'. This procedure is thus generally applicable in combination with other graph coloring algorithms rather than TabuCol. This possibility provides interesting perspectives which are worthy of further investigations.

4.3 Tabu search coloring algorithm

The original RLS method uses a descent-based local search ($DB - LS$) procedure to improve each starting solution built by the hybrid group selection strategy. Starting from an arbitrary k-coloring solution, $DB - LS$ iteratively

improves the current solution, by changing the color of a *conflicting vertex* (i.e., which has the same color as at least one neighbor vertex). At each iteration, if the color change leads to a better solution (i.e., with a reduced number of conflicting edges), this solution becomes the new current solution. This process is repeated until the current solution cannot be further improved. This descent procedure is fast, but it is unable to escape the first local optimum encountered which may be of poor quality. The hybrid group selection strategy of RLS helps to introduce some degree of diversification. However, RLS can still be trapped into poor local optima. To overcome this problem, we replace $DB - LS$ by an improved version of the original tabu search coloring procedure of [25]. As described in [10,14] and summarized in the next paragraph, our improved tabu coloring procedure (TabuCol) integrates three enhancements (see more details below). First, instead of considering a *sample* of neighboring colorings, it considers all neighboring colorings induced by the set of conflicting vertices. Second, it adopts a dynamic tabu tenure which is defined as a function of the number of conflicting vertices and a random number. Third, TabuCol employs dedicated data structures to ensure a fast and incremental evaluation of neighboring solutions.

Given a conflicting k-coloring S, the "one-move" neighborhood $N(S)$ consists of all solutions produced by moving a conflicting vertex v_i from its original group g_u to a new group g_v ($u \neq v$). TabuCol picks at each iteration a best neighbor $\hat{S} \in N(S)$ according to the evaluation function f given by Eq. (3) such that either $\hat{S}$ is a best solution not forbidden by the tabu list or is better than the best solution found so far S^* (i.e., $f(\hat{S}) < f(S^*)$). Ties are broken at random. Once a move is made, e.g., a conflicting vertex v_i is moved from group g_u to group g_v, the vertex v_i is forbidden to go back to group g_u in the following l iterations (l is called tabu tenure). The tabu tenure l is dynamically determined by $l = \mu \times f(S) + \texttt{Random(A)}$, where `Random(A)` returns a random integer in $\{0, \ldots, A-1\}$ [10,14]. In our algorithm, we set $\mu = 1.2$ and $A = 10$. The TabuCol process stops when the number of iterations without improving S^* reaches a predefined value I_{max}, which is set to be 10^5.

For an efficient implementation of the TabuCol procedure, we use an incremental technique [12,14] to maintain and update the move gains $\Delta f = \gamma_{i,u} - \gamma_{i,v}$ for each possible candidate move (i.e., displacing vertex v_i from group g_v to group g_u).

5 Experimental results

5.1 Benchmark instances

This section is dedicated to an extensive experimental evaluation of the PLSCOL algorithm using the well-known DIMACS challenge benchmark instances [2]. These instances have been widely used in the literature for assessing the performances of graph coloring algorithms. They belong to the following six types:

- *Standard random graphs* denoted as "`DSJCn.x`", where n is the number of vertices and 0.x is the density. They are generated in such a way that the probability of an edge being present between two given vertices equals the density.
- *Geometric random graphs* named as "`DSJRn.x` " and "`Rn.x`". They are produced by choosing randomly n points in the unit square, which define the vertices of the graph, by joining two vertices with an edge, if the two related points at a distance less than x from each other. Graphs with letter c denotes the complement of a geometric random graph.
- *Leighton graphs* named as "`len_`χ`x`", are of density below 0.25, where n is the number of vertices, χ is the chromatic number, and $x \in \{a, b, c, d\}$ is a letter to indicate different graphs with the similar parameter settings.
- *"Quasi-random" flat graphs* denoted as "`flatn_`χ`_`δ", where n is the number of vertices, χ is the chromatic number, and δ is a flatness parameter giving the maximal allowed difference between the degrees of two vertices.
- *Scheduling graphs* include two scheduling graphs `school1` and `school1_nsh`.
- *Latin square graph* represents a latin square graph `latin_square_10`.

These instances can be roughly divided into two categories: easy graphs and difficult graphs according to the classification in [13,16]. Let k^* be $\chi(G)$ (if known) or the smallest number of colors for which a legal coloring has been found by at least one coloring algorithm. Then easy graphs G are those that can be colored with k^* colors by a basic coloring algorithm like DSATUR [4] (thus by numerous algorithms). Otherwise, if a k^*-coloring can only be achieved by a few advanced coloring algorithms, the graph will be qualified as difficult. Since easy graphs do not represent any challenge for PLSCOL (and many reference algorithms), we mainly focus our tests on difficult graphs.

[2] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/

5.2 Experimental settings

The PLSCOL algorithm [3] was implemented in C++, and complied using GNU g++ on an Intel E5-2760 with 2.8 GHz and 2GB RAM under Linux operating system. For our experiments, each instance was solved 10 times independently. Each execution was terminated when the maximum allowable run time of 5 CPU hour is reached or a legal k-coloring is found. We used the same parameters setting to solve all tested instances except for the penalization factor β. Parameter β is sensitive to the structures of GCP instances, and it mainly affects the run time to find the best k-coloring (see Section 6.3 for an analysis of this parameter). Accordingly, we adopted values for β from $[0.05, 0.45]$ to obtain the reported results on the set of 20 difficult instances. Table 1 gives the description and setting of each parameter.

Table 1
Parameter setting of PLSCOL

Parameter	Description	Value
ω	noise probability	0.200
α	reward factor for correct group	0.100
β	penalization factor for incorrect group	[0.05, 0.45]
γ	compensation factor for expected group	0.300
ρ	smoothing coefficient	0.500
p_0	smoothing threshold	0.995

5.3 Comparison with the original RLS approach

We first assess the performance of the proposed PLSCOL algorithm with respect to the original RLS approach [51]. This experiment aims to demonstrate the usefulness of the two enhancements, i.e., the probability learning scheme using group matching and the tabu coloring procedure. Both algorithms were run 10 times on each instance, each run being given a CPU time of 5 hours.

The comparative results between PLSCOL and RLS are summarized in Table 2. Columns 1-2 indicate the instance name, its chromatic number χ when it is known or the best-known result reported in the literature (k^*). For each algorithm, we report the smallest number of colors (k) used by the algorithm, the rate of successful runs out of 10 runs ($\#succ$), the average number of moves ($\#iter$) and the average run time in seconds ($time(s)$). Better results (with a smaller k) between the two algorithms are indicated in bold. When both algorithms achieve the same result in terms of number of colors used, we underline the better results in terms of number of iterations.

[3] The code of the PLSCOL algorithm is available upon request.

Table 2
Comparative results of PLSCOL and RLS on the difficult DIMACS graphs.

		PLSCOL				RLS			
Instance	χ/k^*	k	#succ	#iter	time(s)	k	#succ	#iter	time(s)
DSJC250.5	?/28	**28**	10/10	4.0×10^5	4	29	10/10	2.8×10^6	91
DSJC500.1	?/12	**12**	07/10	7.5×10^6	43	13	10/10	5.6×10^5	17
DSJC500.5	?/47	**48**	03/10	7.9×10^7	1786	50	05/10	1.9×10^7	1714
DSJC500.9	?/126	**126**	10/10	2.4×10^7	747	129	03/10	5.0×10^8	9859
DSJC1000.1	?/20	**20**	01/10	2.9×10^8	3694	21	10/10	1.4×10^7	1223
DSJC1000.5	?/83	**87**	10/10	2.7×10^7	1419	94	07/10	3.8×10^8	9455
DSJC1000.9	?/222	**223**	05/10	3.1×10^8	12094	233	03/10	2.5×10^8	12602
DSJR500.1c	?/85	**85**	10/10	3.2×10^7	386	**85**	01/10	4.6×10^6	700
DSJR500.5	?/122	**126**	08/10	7.3×10^7	1860	**126**	01/10	1.8×10^8	3428
le450_15c	15/15	**15**	07/10	2.8×10^8	1718	**15**	07/10	9.5×10^6	308
le450_15d	15/15	**15**	03/10	2.8×10^8	2499	**15**	04/10	5.8×10^8	211
le450_25c	25/25	**25**	10/10	2.0×10^8	1296	26	07/10	4.7×10^6	181
le450_25d	25/25	**25**	10/10	2.2×10^8	1704	26	04/10	1.1×10^7	438
flat300_26_0	26/26	**26**	10/10	4.9×10^6	195	**26**	09/10	9.4×10^6	450
flat300_28_0	28/28	**30**	10/10	1.5×10^7	233	32	09/10	4.4×10^6	173
flat1000_76_0	76/81	**86**	01/10	1.1×10^8	5301	89	01/10	4.5×10^7	11609
R250.5	?/65	**66**	10/10	1.1×10^8	705	**66**	04/10	7.0×10^8	6603
R1000.1c	?/98	**98**	10/10	9.1×10^6	256	100	02/10	5.3×10^8	16139
R1000.5	?/234	**254**	04/10	3.7×10^7	7818	261	02/10	3.7×10^7	9015
latin_square_10	?/97	**99**	08/10	6.7×10^7	2005	**99**	02/10	3.3×10^7	12947

Table 2 indicates that PLSCOL significantly outperforms RLS, achieving better solutions for 13 out of 20 instances and equal solutions on the remaining instances. Among the 7 instances where both algorithms reach the same results, PLSCOL performs better in terms of success rate and average run time in 5 cases. This study shows that the group matching procedure and the tabu coloring procedure of PLSCOL significantly boosts the performance of the original RLS approach for solving the graph coloring problem.

5.4 Comparison with other state-of-the-art algorithms

This section shows a comparison of PLSCOL with 10 state-of-the-art coloring algorithms in the literature. For this comparison, we focus on the criterion of solution quality in terms of the smallest number of colors used to find a legal coloring. In fact, despite the extremely vast literature on graph coloring, there is no uniform experimental condition to assess a coloring algorithm. This is mainly because the difficult DIMACS instances are really challenging. Indeed, the best-known solutions for these instances can be found only by few most powerful algorithms which were implemented with different programming lan-

guages and executed under various computing platforms and different stopping conditions (maximum allowed generations, maximum allowed fitness evaluations, maximum allowed iterations or still maximum allowed cut off time). In most cases, a run time of several hours or even several days was typically applied (see e.g., [30,32,35,38,44,45]). For this reason, it would be quite difficult to compare CPU times of the compared algorithms. Following the common practice of reporting comparative results in the coloring literature, we use the best solution (i.e., the smallest number of used colors) for this comparative study. Since the experimental conditions of the compared algorithms are not equivalent, the comparison is just intended to show the relative performance of the proposed algorithm, which should be interpreted with caution.

5.4.1 Comparative results on difficult instances

This comparative study is based on two state-of-the-art algorithms, including three local search algorithms and seven hybrid population-based algorithms. For indicative purposes, we show the CPU frequency of the computer and the stopping condition used by each reference algorithm. As one can observe, the majority of the reference algorithms allowed large run time limits of at least of 5 hours (the cut off limit for our experiments).

(1) Iterated local search algorithm (IGrAl) [7] (a 2.8 GHz Pentium 4 processor and a cut off time of 1 hour).
(2) Variable space search algorithm (VSS) [26] (a 2.0 GHz Pentium 4 processor and a cut off time of 10 hours).
(3) Local search algorithm using partial solutions (Partial) [1] (a 2.0 GHz Pentium 4 and a time limit of 10 hours together with a limit of $2 * 10^9$ iterations without improvement).
(4) Hybrid evolutionary algorithm (HEA) [14] (the processor used is not available for this oldest algorithm and the results were obtained with different parameter settings).
(5) Adaptive memory algorithm (AMA) [16] (the processor applied is not available and the results were obtained with different parameter settings).
(6) Two-phase evolutionary algorithm (MMT) [32] (a 2.4 GHz Pentium processor and a cut off time of 6000 or 40000 seconds).
(7) Evolutionary algorithm with diversity guarantee (Evo-Div) [38] (a 2.8 GHz Xeon processor and a cut off time of 12 hours).
(8) Memetic algorithm (MACOL or MA) [30] (a 3.4 GHz processor and a cut off time of 5 hours).
(9) Distributed quantum annealing algorithm (QACOL or QA) [44,45] (a 3.0 GHz Intel processor with 12 cores and a cut off time of 5 hours).
(10) The newest parallel memetic algorithm (HEAD) [35] (a 3.1 GHz Intel Xeon processor with 4 cores used to run in parallel the search processes with a cut off time of at least 3 hours).

Table 3 presents the comparative results of PLSCOL (from Table 2) with these state-of-the-art algorithms. In this table, column 2 recalls the chromatic number or best-known value χ/k^* in the literature. Column 3 shows the best results of PLSCOL (k_{best}). The following columns are the best results obtained by the reference algorithms, which are extracted from the literature.

As we observe from Table 3, PLSCOL competes very favorably with the three reference local search algorithms except for instance `flat300_28_0`. The best result for this instance was reached only by one algorithm [1]. To the best of our knowledge, for difficult instances `le450_25c` and `le450_25d`, PLSCOL is the first local search algorithm which achieves the optimal 25-coloring within 1 hour. A more detailed comparison on these two instances is shown in Table 4, including two additional advanced tabu search algorithms (TS-Div and TS-Int with long run times of 50 hours [39]). Although these three local search algorithms can also obtain the optimal 25-coloring, they need much more run times and more iterations with a lower success rate. It is important to mention that no other local search method was able to find the optimal solution of `le450_25c` and `le450_25d` more efficiently than PLSCOL.

When comparing with the seven complex population-based hybrid algorithms, we observe that PLSCOL also remains competitive. In order to facilitate comparisons, we divide these seven reference algorithms into two groups. PLSCOL achieves at least three better solutions compared to the first three algorithms (HEA, AMA and MMT). For example, PLSCOL saves at least one color for `DSJC1000.9`, `flat300_28_0` and `latin_square_10`. Compared to the last four algorithms (Evo-Div, MACOL, QACOL and HEA), PLSCOL competes favorably for some instances. For example, though PLSCOL and the last four algorithms reach the same 126-coloring for `DSJC500.9`, PLSCOL uses less time and iterations to achieve this result (even if timing information was not shown in the table). The same observation applies for other instances, such as `DSJR500.1c` and `R1000.1c`. Moreover, compared to Evo-Div and QACOL, PLSCOL also saves one color and finds the 30-coloring for instance `flat300_28_0` even if MACOL achieves a better performance for this instance.

On the other hand, PLSCOL performs worse than the population-based algorithms for several instances, such as `DSJC1000.5`, `flat1000_76_0` and `R1000.5`. However, this is not a real surprise given that the coloring procedure of PLSCOL (i.e., TabuCol) is extremely simple compared to these highly complex population-based algorithms which integrate various search strategies (solution recombination, advanced population management, local optimization). In this sense, the results achieved by PLSCOL are remarkable and demonstrates that its probability learning scheme greatly boosts the performance of the rather simple tabu coloring algorithm.

Table 3. Comparative results of PLSCOL and 10 state-of-the-art algorithms on the difficult DIMACS graphs.

		local search algorithms				population-based algorithms						
		PLSCOL	IGrAl	VSS	Partial	HEA	AMA	MMT	Evo-Div	MA	QA	HEAD
Instance	χ/k^*	k_{best}	2008	2008	2008	1999	2008	2008	2010	2010	2011	2015
DSJC250.5	?/28	**28**	29	*	*	*	**28**	**28**	*	**28**	**28**	**28**
DSJC500.1	?/12	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
DSJC500.5	?/47	48	50	48	48	48	48	48	48	48	48	**47**
DSJC500.9	?/126	**126**	129	**126**	127	**126**	**126**	127	**126**	**126**	**126**	**126**
DSJC1000.1	?/20	**20**	22	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
DSJC1000.5	?/82	87	94	86	89	83	84	84	83	83	83	**82**
DSJC1000.9	?/222	223	239	224	226	224	224	225	223	223	**222**	**222**
DSJR500.1c	?/85	**85**	**85**	**85**	**85**	*	86	**85**	**85**	**85**	**85**	**85**
DSJR500.5	?/122	126	126	125	125	*	127	**122**	**122**	**122**	**122**	*
le450_15c	15/15	**15**	16	**15**	**15**	**15**	**15**	**15**	*	**15**	**15**	*
le450_15d	15/15	**15**	16	**15**	**15**	**15**	**15**	**15**	*	**15**	**15**	*
le450_25c	25/25	**25**	27	**25**	25	26	26	**25**	**25**	**25**	**25**	**25**
le450_25d	25/25	**25**	27	**25**	25	26	26	**25**	**25**	**25**	**25**	**25**
flat300_26_0	26/26	**26**	*	*	*	*	**26**	**26**	*	**26**	*	*
flat300_28_0	28/28	30	*	**28**	**28**	31	31	31	31	29	31	31
flat1000_76_0	76/81	86	*	85	87	83	84	83	82	82	82	**81**
R250.5	?/65	66	*	*	66	*	*	**65**	**65**	**65**	**65**	**65**
R1000.1c	?/98	**98**	*	*	*	*	*	**98**	**98**	**98**	**98**	**98**
R1000.5	?/234	254	*	*	248	*	*	**234**	238	245	238	245
latin_square_10	?/97	99	100	*	*	*	104	101	100	99	**98**	*

Table 4
Comparative results of PLSCOL and other local search algorithms to find optimal 25-coloring on instance le450_25c and le450_25d.

	le450_25c			le450_25d		
Instance	#succ	#iter	time(h)	#succ	#iter	time(h)
PLSCOL	10/10	2.0×10^8	< 1	10/10	2.2×10^8	< 1
VSS	9/10	1.6×10^9	5	6/10	2.2×10^9	6
Partial	2/5	2.0×10^9	10	3/5	2.0×10^9	10
TS-Div	4/10	7.7×10^8	11	2/10	1.2×10^9	19
TS-Int	10/10	3.4×10^9	10	10/10	6.5×10^9	25

5.4.2 Comparative results on easy instances

Finally, we show that for easy instances, PLSCOL can attain the best-known solution more quickly. We illustrate this by comparing PLSCOL with MACOL [30] which is one of the most powerful population-based coloring algorithms in the literature on 19 easy DIMACS instances (see Table 5). The results of MACOL were obtained on a PC with 3.4 GHz CPU and 2G RAM (which is slightly faster than our PC with 2.8 GHz and 2G RAM). Table 5 indicates that both PLSCOL and MACOL can easily find the best-known results k^* with a 100% success rate. However, PLSCOL uses less time (at most 1 minute) to find its best results while MACOL needs 1 to 5 minutes to achieve the same results. Moreoevr, PLSCOL needs much less iterations (as underlined) compared to MACOL on all instances except `DSJC125.5`, `R125.5` and `R250.1c` for which PLSCOL still requires a shorter run time.

6 Analysis and discussion

In this section, we investigate the usefulness of the main components of PLSCOL. We first study the benefit of probability learning. We have conducted a similar experiment in [51], but as the tabu search algorithm of PLSCOL is more efficient than the descent search of RLS, it is interesting to evaluate the impact of the learning component on the results. Then we study the benefit of the group matching procedure. We also analyze the impact of the penalization factor β over the performance of PLSCOL.

6.1 Benefit of the probability learning scheme

We compared the performance of the PLSCOL algorithm with a variant (denoted by PLSCOL_0) where the probability learning component is disabled. At

Table 5
Comparative results of PLSCOL and MACOL on easy DIMACS graphs.

		MACOL				PLSCOL			
Instance	χ/k^*	k	#succ	#iter	time(m)	k	#succ	#iter	time(m)
DSJC125.1	?/5	**5**	10/10	1.4×10^5	1	**5**	10/10	$\underline{4.8 \times 10^3}$	< 1
DSJC125.5	?/17	**17**	10/10	$\underline{4.8 \times 10^4}$	3	**17**	10/10	6.3×10^4	< 1
DSJC125.9	?/44	**44**	10/10	2.4×10^6	4	**44**	10/10	$\underline{3.0 \times 10^3}$	< 1
DSJC250.1	?/8	**8**	10/10	6.9×10^5	2	**8**	10/10	$\underline{6.4 \times 10^5}$	< 1
DSJC250.9	?/72	**72**	10/10	5.5×10^6	3	**72**	10/10	$\underline{2.6 \times 10^5}$	< 1
R125.1	?/5	**5**	10/10	3.7×10^5	2	**5**	10/10	$\underline{3.6 \times 10^1}$	< 1
R125.1c	?/46	**46**	10/10	2.8×10^6	5	**46**	10/10	$\underline{1.6 \times 10^6}$	< 1
R125.5	?/36	**36**	10/10	$\underline{3.2 \times 10^4}$	1	**36**	10/10	8.0×10^5	< 1
R250.1	?/8	**8**	10/10	1.5×10^6	5	**8**	10/10	$\underline{1.7 \times 10^3}$	< 1
R250.1c	?/64	**64**	10/10	$\underline{2.8 \times 10^6}$	4	**64**	10/10	8.9×10^6	1
DSJR500.1	?/12	**12**	10/10	3.3×10^5	4	**12**	10/10	$\underline{1.6 \times 10^3}$	< 1
R1000.1	?/20	**20**	10/10	2.9×10^5	2	**20**	10/10	$\underline{1.9 \times 10^3}$	< 1
le450_15a	15/15	**15**	10/10	2.7×10^5	2	**15**	10/10	$\underline{1.3 \times 10^5}$	< 1
le450_15b	15/15	**15**	10/10	3.5×10^5	2	**15**	10/10	$\underline{7.4 \times 10^4}$	< 1
le450_25a	25/25	**25**	10/10	1.8×10^5	4	**25**	10/10	$\underline{4.4 \times 10^2}$	< 1
le450_25b	25/25	**25**	10/10	2.8×10^6	3	**25**	10/10	$\underline{3.5 \times 10^2}$	< 1
school1	?/14	**14**	10/10	8.8×10^5	6	**14**	10/10	$\underline{9.3 \times 10^2}$	< 1
school1_nsh	?/14	**14**	10/10	7.3×10^5	1	**14**	10/10	$\underline{5.6 \times 10^3}$	< 1
flat300_20_0	20/20	**20**	10/10	1.7×10^6	4	**20**	10/10	$\underline{1.6 \times 10^3}$	< 1

each generation, instead of generating a new initial solution with the probability matrix, PLSCOL$_0$ generates a new solution at random (i.e., line 9 of Algorithm 2 is replaced by the random generation procedure) and then uses the tabu search procedure to improve the initial solution. In other words, PLSCOL$_0$ repetitively restarts the TabuCol procedure. We ran PLSCOL$_0$ under the same stopping condition as before – PLSCOL$_0$ stops if a legal k-coloring is found or the maximum allowed run time of 5 CPU hours is reached. Each instance was solved 10 times.

Table 6 summarizes the computational statistics of PLSCOL and PLSCOL$_0$. Columns 1-2 indicates the name of each instance, its chromatic number χ when it is known or the best-known result reported in the literature (k^*). For each algorithm, we report the following information: the smallest number of colors (k) used by the algorithm, the frequency of successful runs out of 10 runs ($\#succ$), the average number of generations ($\#gen$), the average number of tabu search iterations (or moves) ($\#iter$) and the average run time in seconds. The lowest k values between the two algorithms are in bold (a smaller value indicates a better performance in terms of solution quality). When the two algorithms achieve the same result in terms of number of colors used, we underline the smallest number of iterations iterations (which indicates a better performance in terms of computational efficiency).

Table 6
Comparative results of PLSCOL and PLSCOL$_0$ on the difficult DIMACS graphs.

		PLSCOL					PLSCOL$_0$				
Instance	χ/k^*	k	#succ	#gen	#iter	time(s)	k	#succ	#gen	#iter	time(s)
DSJC250.5	?/28	**28**	10/10	3	4.0×10^5	4	**28**	10/10	58	1.1×10^7	102
DSJC500.1	?/12	**12**	07/10	69	7.5×10^6	43	**12**	10/10	8936	1.9×10^9	12808
DSJC500.5	?/47	**48**	03/10	761	7.9×10^7	1786	49	06/10	1122	2.5×10^8	5543
DSJC500.9	?/126	**126**	10/10	187	2.4×10^7	747	127	10/10	362	9.2×10^7	2704
DSJC1000.1	?/20	**20**	01/10	369	2.9×10^8	3694	21	10/10	2	3.7×10^5	4
DSJC1000.5	?/83	**87**	10/10	203	2.7×10^7	1419	89	02/10	492	1.4×10^8	7031
DSJC1000.9	?/222	**223**	05/10	2886	3.1×10^8	12094	229	05/10	270	1.0×10^8	9237
DSJR500.1c	?/85	**85**	10/10	317	3.2×10^7	386	**85**	10/10	554	8.3×10^7	1825
DSJR500.5	?/122	**126**	08/10	464	7.3×10^7	1860	127	01/10	2,701	4.3×10^8	8592
le450_15c	15/15	**15**	07/10	2883	2.8×10^8	1718	**15**	10/10	155	2.1×10^7	238
le450_15d	15/15	**15**	03/10	2787	2.8×10^8	2499	**15**	10/10	766	1.1×10^8	1314
le450_25c	25/25	**25**	10/10	1968	2.0×10^8	1296	26	10/10	1	8.1×10^4	1
le450_25d	25/25	**25**	10/10	2110	2.2×10^8	1704	26	10/10	1	1.1×10^5	2
flat300_26_0	26/26	**26**	10/10	49	4.9×10^6	195	**26**	10/10	31	5.1×10^6	254
flat300_28_0	28/28	**30**	10/10	147	1.5×10^7	233	31	10/10	95	1.9×10^7	242
flat1000_76_0	76/81	**86**	01/10	908	1.1×10^8	5301	89	02/10	339	9.1×10^7	3709
R250.5	?/65	**66**	10/10	865	1.1×10^8	705	**66**	01/10	1793	2.3×10^8	2038
R1000.1c	?/98	**98**	10/10	88	9.1×10^6	256	**98**	10/10	110	2.0×10^7	702
R1000.5	?/234	**254**	04/10	189	3.7×10^7	7818	260	10/10	1	3.1×10^5	124
latin_square_10	?/97	**99**	08/10	666	6.7×10^7	2005	103	10/10	444	9.7×10^7	7769

From Table 6, it can be seen that for the 20 instances, PLSCOL obtains a better solution for 12 instances and an equal solution for the 8 remaining instances. With the help of the learning scheme, the improvement of PLSCOL over PLSCOL$_0$ is quite significant for several very difficult graphs [4] like `DSJC1000.9` (-6 colors), at `flat1000_76_0` (-3 colors), `R1000.5` (-6 colors) and `latin_square_10` (-4 colors). Finally, we observe that for the 8 instances where both algorithms achieve an equal result in terms of colors used, PLSCOL requires less iterations and shorter computing times than PLSCOL$_0$ in 6 cases. This study demonstrates the effectiveness of the probability learning scheme used in our PLSCOL algorithm.

6.2 Benefit of group matching procedure

We now investigate the usefulness of the group matching procedure (Section 4.2), which is used to provide feedback information to the probability learning

[4] For these instances, even gaining one color could be difficult, since when k is close to χ, finding a legal coloring is usually much harder for k than for $k+1$.

component. For this study, we replace the group matching procedure by the group comparison procedure of the original RLS approach described in Section 3. For this purpose, we create PLSCOL$_1$, which is a PLSCOL variant where we replace line 14 of Algorithm 2 with the group comparison procedure of RLS. We ran PLSCOL$_1$ under the same stopping condition as before – PLSCOL$_1$ stops if a legal k-coloring is found or the maximum allowed run time of 5 CPU hours is reached. Each tested instance was solved 10 times.

Table 7
Comparative results of PLSCOL and PLSCOL$_1$ on the difficult DIMACS graphs.

		PLSCOL					PLSCOL$_1$				
Instance	χ/k^*	k	#succ	#gen	#iter	time(s)	k	#succ	#gen	#iter	time(s)
DSJC250.5	?/28	**28**	10/10	3	$\underline{4.0 \times 10^5}$	4	**28**	10/10	103	6.7×10^7	1026
DSJC500.1	?/12	**12**	07/10	69	7.5×10^6	43	**12**	10/10	5	$\underline{5.5 \times 10^6}$	59
DSJC500.5	?/47	**48**	03/10	761	7.9×10^7	1786	50	10/10	15	6.2×10^6	154
DSJC500.9	?/126	**126**	10/10	187	$\underline{2.4 \times 10^7}$	747	**126**	06/10	931	2.7×10^8	9314
DSJC1000.1	?/20	**20**	01/10	369	2.9×10^8	3694	21	10/10	0	8.9×10^4	1
DSJC1000.5	?/83	**87**	10/10	203	2.7×10^7	1419	89	09/10	407	6.9×10^7	4070
DSJC1000.9	?/222	**223**	05/10	2886	3.1×10^8	12094	224	02/10	826	1.0×10^8	8265
DSJR500.1c	?/85	**85**	10/10	317	$\underline{3.2 \times 10^7}$	386	**85**	10/10	60	5.5×10^7	600
DSJR500.5	?/122	**126**	08/10	464	$\underline{7.3 \times 10^7}$	1860	**126**	07/10	753	3.9×10^8	7534
le450_15c	15/15	**15**	07/10	2883	$\underline{2.8 \times 10^8}$	1718	**15**	02/10	874	1.5×10^9	8744
le450_15d	15/15	**15**	03/10	2787	2.8×10^8	2499	16	10/10	0	3.3×10^5	3
le450_25c	25/25	**25**	10/10	1968	2.0×10^8	1296	26	10/10	0	1.2×10^5	1
le450_25d	25/25	**25**	10/10	2110	2.2×10^8	1704	26	10/10	1	1.3×10^5	2
flat300_26_0	26/26	**26**	10/10	49	$\underline{4.9 \times 10^6}$	195	**26**	07/10	603	1.4×10^8	6034
flat300_28_0	28/28	**30**	10/10	147	1.5×10^7	233	31	10/10	410	2.4×10^8	4101
flat1000_76_0	76/81	**86**	01/10	908	1.1×10^8	5301	87	01/10	321	3.9×10^7	3212
R250.5	?/65	**66**	10/10	865	$\underline{1.1 \times 10^8}$	705	**66**	10/10	151	2.0×10^8	1516
R1000.1c	?/98	**98**	10/10	88	9.1×10^6	256	**98**	10/10	18	$\underline{3.8 \times 10^6}$	181
R1000.5	?/234	**254**	04/10	189	3.7×10^7	7818	255	05/10	241	1.6×10^7	2425
latin_square_10	?/97	**99**	08/10	666	6.7×10^7	2005	100	06/10	648	2.4×10^8	6480

Table 7 displays the comparative results between PLSCOL and PLSCOL$_1$, based on the same indicators adopted in Table 6. From this table, we observe that PLSCOL significantly outperforms PLSCOL$_1$, achieving better objective values for 11 out of 20 instances and equal results for the remaining instances. Moreover, for the 9 instances where both algorithms achieve the same best objective values, PLSCOL needs less iterations and shorter computing times than PLSCOL$_1$ for 7 instances (underlined entries in Table 7). These observations confirm the usefulness of the group matching procedure for probability learning within the PLSCOL algorithm.

6.3 Effect of the penalization factor β

The penalization factor β is used to determine the new probability vector when an item selects an incorrect group. Preliminary results suggest that the performance of PLSCOL is more sensitive to β than the other two parameters α and γ. In the following, we report a detailed analysis of β on six selected instances. To avoid the influence caused by random numbers, we set the random seed to 1 in the experiments. For each parameter setting and each instance, we conduct an independent experiment with a maximum allowed time limit of 5 CPU hours. We set other parameters to their default values (as shown in Table 1) and only vary the parameter β.

Table 8
Effect of penalization factor β on the run time (s) of PLSCOL.

β	DSJC250.5	DSJC500.9	DSJR500.1c	R1000.1c	DSJC1000.9	flat1000_76_0
0.45	13.10	2055.03	**128.40**	88.50	*	*
0.40	17.65	367.57	139.73	229.88	5884.32	7474.21
0.35	13.10	1096.23	471.30	**34.25**	**1158.67**	3154.03
0.30	17.57	2467.98	242.40	2644.97	*6133.87*	*7519.52*
0.25	13.28	903.89	*6120.46*	*8383.67*	7122.16	1035.84
0.20	13.26	*10149.23*	1412.54	164.01	3008.86	*
0.15	*17.66*	8765.75	1519.88	55.96	4015.04	7163.64
0.10	**11.48**	**316.26**	1232.88	113.23	2815.88	1244.57
0.05	17.14	*	5753.18	181.73	4349.16	**995.73**
Δ_{max}	6.18	9832.97	5992.06	8349.42	4975.20	6523.79

Table 8 shows the impact of β on the performance of PLSCOL with nine different values for β, $\beta \in \{0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45\}$. In this table, we report the run time (the best values are in bold and the worst values are in italic) needed to find the given best k-coloring for each parameter value and each instance. "$*$" indicates that PLSCOL cannot find a k-coloring with this parameter setting. At the last row of the table, we also give the maximum run time difference of PLSCOL under two different β values. Table 8 indicates that PLSCOL can find a legal coloring with the given best k in the majority of cases except for `DSJC500.9` with $\beta = 0.05$, `DSJC1000.9` with $\beta = 0.45$ and `flat1000_76_0` with $\beta = 0.45$ and $\beta = 0.20$ respectively. This parameter mainly affects the run time of PLSCOL to find the given k-coloring. Taking the instance `DSJC500.9` as an example, with $\beta = 0.10$, PLSCOL finds the best 126-coloring within 316.26 seconds while it needs much more time (10149.23 seconds) to find the 126-coloring when $\beta = 0.20$. The maximum difference of run time with different β values is 9832.97 seconds. We also observe that the optimal setting of β depends to a large extent on the structures of the instances which greatly vary for different graphs. Although the run time required by PLSCOL to find a legal coloring is sensitive to the

β values, PLSCOL can successfully find the solutions with many β values.

7 Conclusion

In this paper, we presented an enhanced probability learning based local search approach for the well-known graph coloring problem, which improves the general RLS approach of [51]. The first enhancement improves the probability learning scheme of the original RLS by using a group matching procedure to find the group-to-group relation between a starting solution and its improved solution. This matching procedure copes with the difficulty raised by symmetric solutions of GCP. The second enhancement concerns the coloring algorithm based on tabu search, which is both more powerful than the descent-based algorithm used in RLS, and still remains simple compared to more complex hybrid algorithms.

Experimental evaluations on popular DIMACS challenge benchmark graphs showed that PLSCOL competes favorably with all existing local search coloring algorithms. Compared with the most effective hybrid evolutionary algorithms which are much more sophisticated in their design, PLSCOL remains competitive in spite of the simplicity of its underlying coloring algorithm.

As future work, we could explore the following possibilities, First, the optimization component of PLSCOL can be considered as a black box. As such, it would be interesting to investigate whether using a more powerful coloring algorithm instead of the TabuCol algorithm leads to still better results. Second, PLSCOL with the group matching procedure is able to avoid the difficulty related to symmetric solutions of grouping problems with interchangeable groups. It would be interesting to adapt this approach to other grouping problems with this feature (e.g., identical parallel-machine scheduling, data clustering and bin-packing). Third, this work shows that learning techniques can help coloring algorithms to find better solutions. More investigation in this direction is thus desirable to make additional innovations.

Finally, reinforcement learning techniques [6,42,43,46,50] could be promising to be used in combination with heuristic search algorithms. For instance, they can be employed to improve local search by predicting sequential searching actions. More generally, reinforcement learning can help a search algorithm to take right search decisions and to make the search more informed and adaptive [48]. Reinforcement learning should be considered as a valuable tool for optimization that deserves more research effort and that could lead to new powerful optimization methods for solving hard combinatorial problems.

Acknowledgement

We are grateful to the anonymous referees for their helpful comments and suggestions, which helped us to improve the presentation of the work. The first author was supported by the China Scholarship Council (2014-2017), which is also acknowledged.

References

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

[2] N. Bouhmala, O. Granmo, Solving graph coloring problems using learning automata, in: Evolutionary Computation in Combinatorial Optimization (EvoCOP 2008), Lecture Notes in Computer Science 4972 (2008) 277–288.

[3] J. Boyan, A. W. Moore, Learning evaluation functions to improve optimization by local search, Journal of Machine Learning Research 1 (2001) 77–112.

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

[5] E. Burke, D. Elliman, R. Weare, A university timetabling system based on graph colouring and constraint manipulation, Journal of Research on Computing in Education 27 (1) (1994) 1–18.

[6] L. Busoniu, R. Babuska, B. D. Schutter, D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, Inc., 2010.

[7] M. Caramia, P. Dell'Olmo, Coloring graphs by iterated local search traversing feasible and infeasible solutions, Discrete Applied Mathematics 156 (2) (2008) 201–217.

[8] G. J. Chaitin, Register allocation & spilling via graph coloring, ACM Sigplan Notices 17 (6) (1982) 98–101.

[9] D. de Werra, An introduction to timetabling, European Journal of Operational Research 19 (2) (1985) 151–162.

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

[11] A. Elhag, E. Özcan, A grouping hyper-heuristic framework: Application on graph colouring, Expert Systems with Applications 42 (13) (2015) 5491–5507.

[12] C. Fleurent, J. A. Ferland, Genetic and hybrid algorithms for graph coloring, Annals of Operations Research 63 (3) (1996) 437–461.

[13] P. Galinier, J. Hamiez, J. K. Hao, D. C. Porumbel, Recent advances in graph vertex coloring, in: Handbook of Optimization - From Classical to Modern Approach, 2013, pp. 505–528.

[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 & Operation Research 33 (2006) 2547–2562.

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

[17] A. Gamst, Some lower bounds for a class of frequency assignment problems, IEEE Transactions on Vehicular Technology 35 (1) (1986) 8–14.

[18] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[19] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[20] Y. Guo, G. Goncalves, T. Hsu, A multi-agent based self-adaptive genetic algorithm for the long-term car pooling problem, Journal of Mathematical Modelling and Algorithms in Operations Research 12(1) (2013) 45–66.

[21] W. K. Hale, Frequency assignment: Theory and applications, Proceedings of the IEEE 68 (12) (1980) 1497–1514.

[22] J. K. Hao, Memetic Algorithms in Discrete Optimization. In F. Neri, C. Cotta, P. Moscato (Eds.) Handbook of Memetic Algorithms. Studies in Computational Intelligence 379, Chapter 6, pages 73-94, 2012.

[23] J. K. Hao, R. Dorne, P. Galinier, Tabu search for frequency assignment in mobile radio networks, Journal of Heuristics 4 (1) (1998) 47–62.

[24] J. K. Hao, Q. Wu, Improving the extraction and expansion method for large graph coloring, Discrete Applied Mathematics 160 (16-17) (2012) 2397–2407.

[25] A. Hertz, D. de Werra, Using tabu search techniques for graph coloring, Computing 39 (4) (1987) 345–351.

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

[27] F. Hutter, L. Xu, H. H. Hoos, K. Leyton-Brown, Algorithm runtime prediction: Methods & evaluation, Artificial Intelligence 206 (2014) 79 – 111.

[28] H. W. Kuhn, The hungarian method for the assignment problem, Naval Research Logistics Quarterly 2 (1-2) (1955) 83–97.

[29] 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.

[30] Z. Lü, J. K. Hao, A memetic algorithm for graph coloring, European Journal of Operational Research 203 (1) (2010) 241–250.

[31] S. Mahmoudi, S. Lotfi, Modified cuckoo optimization algorithm (MCOA) to solve graph coloring problem, Applied Soft Computing 33 (2015) 48–64.

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

[33] E. Malaguti, M. Monaci, P. Toth, An exact approach for the vertex coloring problem, Discrete Optimization 8 (2) (2011) 174–190.

[34] E. Malaguti, P. Toth, A survey on vertex coloring problems, International Transactions in Operational Research 17 (1) (2010) 1–34.

[35] L. Moalic, A. Gondran, The new memetic algorithm HEAD for graph coloring: An easy way for managing diversity, in: Evolutionary Computation in Combinatorial Optimization (EvoCOP 2015), Lecture Notes in Computer Science 9026 (2015) 173–183.

[36] K. S. Narendra, M. A. L. Thathachar, Learning Automata: An Introduction, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[37] C. H. Papadimitriou, K. Steiglitz, Combinatorial optimization: Algorithms and complexity, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[38] D. C. Porumbel, J. K. Hao, P. Kuntz, An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring, Computers & Operation Research 37 (10) (2010) 1822–1832.

[39] D. C. Porumbel, J. K. Hao, P. Kuntz, A search space cartography for guiding graph coloring heuristics, Computers & Operations Research 37 (4) (2010) 769–778.

[40] I. Sghir, J. K. Hao, I. B. Jaafar, K. Ghédira, A multi-agent based optimization method applied to the quadratic assignment problem, Expert Systems with Applications 42(23) (2015) 9252–9263.

[41] I. Sghir, J. K. Hao, I. B. Jaafar, K. Ghédira, A distributed hybrid algorithm for the graph coloring problem, Lecture Notes in Computer Science 9554 (2016) 205–218.

[42] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, Vol. 1, MIT Press Cambridge, 1998.

[43] C. Szepesvári, Algorithms for Reinforcement Learning, Morgan & Claypool Publishers, 2010.

[44] O. Titiloye, A. Crispin, Quantum annealing of the graph coloring problem, Discrete Optimization 8 (2) (2011) 376–384.

[45] O. Titiloye, A. Crispin, Parameter tuning patterns for random graph coloring with quantum annealing, Plos One 7(11) (2012) e50060.

[46] F. Y. Wang, H. Zhang, D. Liu, Adaptive dynamic programming: An introduction, IEEE Computational Intelligence Magazine, 4(2) (2009) 39–47.

[47] X. Wang, L. Tang, A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem, Computers & Operations Research 79 (2017) 60–77.

[48] T. Wauters, K. Verbeeck, P. De Causmaecker, G. V. Berghe, Boosting metaheuristic search using reinforcement learning, in: Hybrid Metaheuristics, Springer, 2013, pp. 433–452.

[49] Q. Wu, J. K. Hao, An extraction and expansion approach for graph coloring, Asian-Pacific Journal of Operational Research 30 (5), 1350018.

[50] X. Xu, L. Zuo, Z. Huang, Reinforcement learning algorithms with function approximation: Recent advances and applications, Information Sciences, 261 (2014) 1–31.

[51] Y. Zhou, J. K. Hao, B. Duval, Reinforcement learning based local search for grouping problems: A case study on graph coloring, Expert Systems with Applications 64 (2016) 412–422.

[52] Y. Zhou, J. K. Hao, B. Duval, Opposition-based memetic search for the maximum diversity problem, IEEE Transactions on Evolutionary Computation 21 (5) (2017) 731–745.

[53] Z. Zhou, C. M. Li, C. Huang, R. Xu, An exact algorithm with learning for the graph coloring problem, Computers & Operations Research 51 (2014) 282–301.

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