Reinforcement learning based local search for grouping problems: A case study on graph coloring

Yangming Zhou, Jin-Kao Hao, Béatrice Duval

Reinforcement learning based local search for grouping problems: A case study on graph coloring

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

[a] *LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers, France*
[b] *Institut Universitaire de France, Paris, France*

Expert Systems with Application, doi: dx.doi.org/10.1016/j.eswa.2016.07.0xx

Abstract

Grouping problems aim to partition a set of items into multiple mutually disjoint subsets according to some specific criterion and constraints. Grouping problems cover a large class of computational problems including clustering and classification that frequently appear in expert and intelligent systems as well as many real applications. This paper focuses on developing a general-purpose solution approach for grouping problems, i.e., reinforcement learning based local search (RLS), which combines reinforcement learning techniques with local search. This paper makes the following contributions: we show that (1) reinforcement learning can help obtain useful information from discovered local optimum solutions; (2) the learned information can be advantageously used to guide the search algorithm towards promising regions. To the best of our knowledge, this is the first attempt to propose a formal model that combines reinforcement learning and local search for solving grouping problems. The proposed approach is verified on a well-known representative grouping problem (graph coloring). The generality of the approach makes it applicable to other grouping problems.

Keywords: Grouping problems; Reinforcement learning; Heuristics; Learning-based optimization.

1. Introduction

Grouping problems aim to partition a set of items into a collection of mutually disjoint subsets according to some specific criterion and constraints. Grouping problems naturally arise in numerous domains. Well-known grouping problems include, for instance, the graph coloring problem (GCP) (Garey & Johnson,

[*] Corresponding author.
Email addresses: zhou.yangming@yahoo.com (Yangming Zhou), hao@info.univ-angers.fr (Jin-Kao Hao), bd@info.univ-angers.fr (Béatrice Duval)

1979; Galinier & Hao, 1999; Lewis, 2009; Elhag & Özcan, 2015) and its variants like selective graph coloring (Demange, Monnot, Pop & Ries, 2012, 2014), partition graph coloring (Fidanova & Pop, 2016; Pop, Hu & Raidl, 2013), sum coloring (Jin, Hamiez & Hao, 2012) and bandwidth coloring (Lai, Hao, Lü & Glover, 2016), timetabling (Lewis & Paechter, 2007; Elhag & Özcan, 2015), bin packing (Falkenauer, 1998; Quiroz-Castellanos, Cruz-Reyes, Torres-Jiménez et al., 2015), scheduling (Kashan, Kashan & Karimiyan, 2013) and clustering (Agustn-Blas, Salcedo-Sanz, Jiménez-Fernández, et al., 2012). Formally, given a set V of n distinct items, the task of a grouping problem is to partition the items of set V into k different groups g_i $(i = 1, \ldots, k)$ (k can be fixed or variable), such that $\cup_{i=1}^{k} g_i = V$ and $g_i \cap g_j = \emptyset, i \neq j$ while taking into account some specific constraints and optimization objective. For instance, the graph coloring problem is to partition the vertices of a given graph into a minimum number of k color classes such that adjacent vertices must be put into different color classes.

According to whether the number of groups k is fixed in advance, grouping problems can be divided into constant grouping problems or variable grouping problems (Kashan, Kashan & Karimiyan, 2013). In some contexts, the number of groups k is a fixed value of the problem, such as identical or non-identical parallel-machines scheduling problem, while in other settings, k is variable and the goal is to find a feasible grouping with a minimum number of groups, such as bin packing and graph coloring. Grouping problems can also be classified according to the types of the groups. A grouping problem with identical groups means that all groups have similar characteristics, thus naming of the groups is irrelevant. Aforementioned examples such as identical parallel-machines scheduling, bin-packing and graph coloring belong to this category. Another category of grouping problems have non-identical groups where the groups are of different characteristics. Hence, swapping items between two groups will result in a new grouping, such as the non-identical parallel-machines scheduling problem.

Many grouping problems, including the examples mentioned above are NP-hard, thus computationally challenging. Consequently, exponential times are expected for any algorithm to solve such a problem exactly. On the other hand, heuristic and meta-heuristic methods are often employed to find satisfactory sub-optimal solutions in acceptable computing time, but without ensuring the optimality of the attained solutions. A number of heuristic approaches for grouping problems, in particular based on genetic algorithms, have been proposed in the literature with varying degrees of success (Falkenauer, 1998; Galinier & Hao, 1999; Quiroz-Castellanos, Cruz-Reyes, Torres-Jiménez et al., 2015). These approaches are rather complex since they are population-based and often hybridized with other search methods like local optimization.

In this work, we are interested in investigating a general-purpose local search methodology for grouping problems which employs machine learning techniques to process information collected from the search process with the purpose of improving the performance of heuristic algorithms. Indeed, previous work has demonstrated that machine learning can contribute to improve optimization methods (Baluja, Barto, Boese, et al., 2000; Battiti & Brunato, 2014; Hafiz &

Abdennour, 2016). The studies in these areas have pursued different objectives, illustrated as follows.

- Algorithm selection and analysis. For instance, Hutter et al. used machine learning techniques such as random forests and approximate Gaussian process to model algorithm's runtime as a function of problem-specific instance features. This model can predict algorithm runtime for the propositional satisfiability problem, traveling salesperson problem and mixed integer programming problem (Hutter, Xu, Hoos & Leyton-Brown, 2014).

- Learning generative models of solutions. For example, Ceberio, Mendiburu & Lozano (2013) introduced the Plackett-Luce probability model to the framework of estimation of distribution algorithms and applied it to solve the linear order problem and the flow-shop scheduling problem.

- Learning evaluation functions. For instance, Boyan & Moore (2001) proposed the STAGE algorithm to learn an evaluation function which predicts the outcome of a local search algorithm as a function of state features along its search trajectories. The learned evaluation function is used to bias future search trajectories towards better solutions.

- Understanding the search space. For example, Porumbel, Hao & Kuntz (2010a) used multidimensional scaling techniques to explore the spatial distribution of the local optimal solutions visited by tabu search, thus improving local search algorithms for the graph coloring problem. For the same problem, Hamiez & Hao (1993) used the results of an analysis of legal k-colorings to help finding solutions with fewer colors.

In this paper, we present the reinforcement learning based local search (RLS) approach for grouping problems, which combines reinforcement learning techniques with a descent-based local search procedure (Section 3). Our proposed RLS approach belongs to the above-mentioned category of learning generative models of solutions. For a grouping problem with its k groups, we associate to an item a probability vector with respect to each possible group and determine the group of the item according to the probability vector (Sections 3.1 and 3.2). Once all items are assigned to their groups, a grouping solution is generated. Then, the descent-based local search procedure is invoked to improve this solution until a local optimum is attained (Section 3.3). At this point, the probability vector of each item is updated by comparing the item's groups in the starting solution and in the attained local optimum solution (Section 3.4). If an item does not change its group, then we reward the selected group of the item, otherwise we penalize the original group and compensate the new group (i.e., expected group). There are two key issues that need to be considered, i.e., how do we select a suitable group for each item according to the probability vector, and how do we smooth the probabilities to avoid potential search traps. To handle these issues, we design two strategies: a hybrid group selection strategy that uses a noise probability to switch between random selection and greedy selection (Section 3.2); and a probability smoothing mechanism to forget old decisions (Section 3.5).

To evaluate the viability of the proposed RLS method, we use the well-known graph coloring problem as a case study (Section 4). GCP is one representative grouping problem which has been object of intensive studies in the past decades (Section 4.1). GCP and its variants like bandwidth coloring have numerous applications including those arising in expert and intelligent decision systems including school timetabling (Ahmed, Özcan & Kheiri, 2012), frequency assignment in mobile networks (Lai & Hao, 2015) and structural analysis of complex networks (Xin, Xie & Yang, 2013). Popular problems like Sudoku and geographical maps of countries are two other application examples of GCP. For our experimental assessment of the proposed method, we test our approach on the popular DIMACS and COLOR02 benchmarks which cover both randomly generated graphs and graphs from real applications (Section 4.2). The experimental results demonstrate that the proposed approach, despite its simplicity, achieves competitive performances on most tested instances compared to many existing algorithms (Section 4.4). With an analysis of three important issues of RLS (Section 4.3), we show the effectiveness of combining reinforcement learning and descent-based local search. We also assess the contribution of the probability smoothing technique to the performance of RLS. We draw useful conclusions based on the computational outcomes (Section 5).

2. Reinforcement learning and heuristic search

In this section, we briefly introduce the principles of reinforcement learning (RL) and provide a review of some representative examples of using reinforcement learning to solve combinatorial optimization problems.

2.1. Reinforcement learning

Reinforcement learning is a learning pattern, which aims to learn optimal actions from a finite set of available actions through continuously interacting with an unknown environment. In contrast to supervised learning techniques, reinforcement learning does not need an experienced agent to show the correct way, but adjusts its future actions based on the obtained feedback signal from the environment (Gosavi, 2009).

There are three key elements in a RL agent, i.e., states, actions and rewards. At each instant a RL agent observes the current state, and takes an action from the set of its available actions for the current state. Once an action is performed, the RL agent changes to a new state, based on transition probabilities. Correspondingly, a feedback signal is returned to the RL agent to inform it about the quality of its performed action.

2.2. Reinforcement learning for heuristic search

There are a number of studies in the literature where reinforcement learning techniques are put at the service of heuristic algorithms for solving combinatorial problems. Reinforcement learning techniques in these studies have been explored at three different levels.

Heuristic level where RL is directly used as a heuristic to solve optimization problems. In this case, RL techniques are used to learn and directly assign values to the variables. For example, Miagkikh & Punch (1999) proposed to solve combinatorial optimization problems based on a population of RL agents. Pairs of variable and value are considered as the RL states, and the branching strategies as the actions. Each RL agent is assigned a specific area of the search space where it has to learn and find good local solutions. Another example is presented in (Torkestani & Meybodi, 2011) where reinforcement learning was used to update the states of cellular automata applied to the graph coloring problem.

Meta-heuristic level where RL is integrated into a meta-heuristic. There are two types of these algorithms. Firstly, RL is used to learn properties of good initial solutions or an evaluation function that guides a meta-heuristic toward high quality solutions. For example, RL is employed to learn a new evaluation function over multiple search trajectories of the same problem instance and alternates between using the learned and the original evaluation function (Boyan & Moore, 2001). Secondly, RL learns the best neighborhoods or heuristics to build or change a solution during the search, so that a good solution can be obtained at the end. For instance, Xu, Stern & Samulowitz (2009) proposed a formulation of constraint satisfaction problems as a RL task. A number of different variable ordering heuristics are available, and RL learns which one to use, and when to use it.

Hyper-heuristic level where RL is used as a component of a hyper-heuristic. Specifically, RL is integrated into selection mechanisms and acceptance mechanisms in order to select a suitable low-level heuristic and determine when to accept a move respectively. For example, Burke, Kendall & Soubeiga (2003) presented a hyper-heuristic in which the selection of low-level heuristics makes use of basic reinforcement learning principles combined with a tabu search mechanism. The algorithm increases or decreases the rank of the low-level heuristics when the objective function value is improving or deteriorating. Two other examples can be found in (Guo, Goncalves & Hsu, 2013; Sghir, Hao, Jaafar & Ghédira, 2015) where RL is used to schedule several search operators under the genetic and multi-agent based optimization frameworks.

Both meta-heuristic level and hyper-heuristic level approaches attempt to replace the random component of an algorithm with a RL component to obtain an informed decision mechanism. In our case, RLS uses reinforcement learning techniques to generate promising initial solutions for descent-based local search.

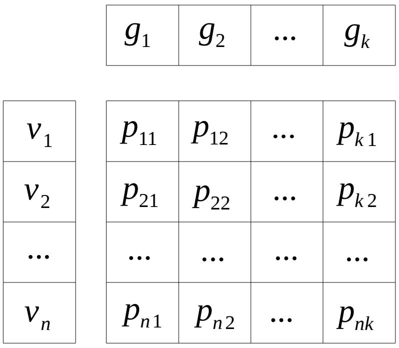
3. Reinforcement learning based local search for grouping problems

Grouping problems aim to partition a set of items into k disjoint groups according to some imperative constraints and an optimization criterion. For our RLS approach, we suppose that the number of groups k is given in advance. Note that such an assumption is not necessarily restrictive. In fact, to handle a grouping problem with variable k, one can repetitively run RLS with different k values. We will illustrate this approach on the graph coloring problem in Section 4.

3.1. Main scheme

By combining reinforcement learning techniques with a solution improvement procedure, our proposed RLS approach is composed of four keys components: a descent-based local search procedure, a group selection strategy, a probability updating mechanism, and a probability smoothing technique.

We define a probability matrix P of size $n \times k$ (n is the number of items and k is the number of groups, see Figure 1 for an example). An element p_{ij} denotes the probability that the i-th item v_i selects the j-th group g_j as its group. Therefore, the i-th row of the probability matrix defines the probability vector of the i-th item and is denoted by p_i. At the beginning, all the probability values in the probability matrix are set as $1/k$. It means that all items select a group from the available k groups with equal probability.



	g_1	g_2	...	g_k
v_1	p_{11}	p_{12}	...	p_{k1}
v_2	p_{21}	p_{22}	...	p_{k2}
...	...	...	...	...
v_n	p_{n1}	p_{n2}	...	p_{nk}

Figure 1: Probability matrix P

At instant t, each item v_i, $i \in \{1, 2, ..., n\}$ selects one suitable group g_j, $j \in \{1, 2, ..., k\}$ by applying a group selection strategy (Section 3.2) based on its probability vector $p_i(t)$. Once all the items are assigned to their groups, a grouping solution S_t is obtained. Then, this solution is improved by a descent-based local search procedure to attain a local optimum denoted by $\hat{S}_t$ (Section 3.3). By comparing the solution S_t and the improved solution $\hat{S}_t$, we update the probability vector of each item based on the following rules (Section 3.4):

(a) If the item stays in its original group, then we reward the selected group.

(b) If the item is moved to a new group, then we penalize the selected group and compensate its new group (i.e., expected group).

Next, we apply a probability smoothing technique to smooth each item's probability vector (Section 3.5). Hereafter, RLS iteratively runs until a predefined stopping condition is reached (e.g., a legal solution is found or the

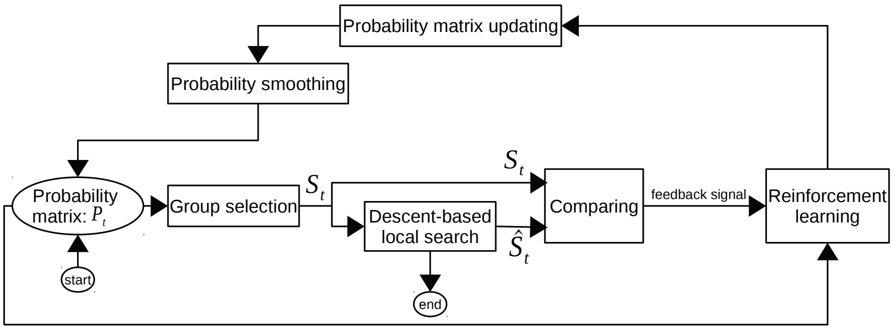


Figure 2: A schematic diagram of RLS for grouping problems. From a starting solution generated according to the probability matrix, RLS iteratively runs until it meets its stopping condition (see Sections 3.2 to 3.5 for more details)

number of iterations without improvement attains a given maximum value). The schematic diagram of RLS for grouping problems is depicted in Figure 2 while its algorithmic pseudo-code is provided in Algorithm 1. In the following subsections, the four key components of our RLS approach are presented.

Algorithm 1 Pseudo-code of our RLS for grouping problems.

```
Input:
    $G$: a grouping problem instance;
    $k$: the number of available groups;
Output: the best solution $S^*$ found so far;
for all $v_i, i = 1, 2, ..., n$ do
    $P_0 = [p_{ij} = 1/k]_{j=1,2,...,k}$;
end for
repeat
    $S_t \leftarrow groupSelecting(P_{t-1}, \omega)$;                              /* Section 3.2 */
    $\hat{S}_t \leftarrow DB - LS(S_t)$;                                          /* Section 3.3 */
    $P_t \leftarrow probabilityUpdating(P_{t-1}, S_t, \hat{S}_t, \alpha, \beta, \gamma)$;   /* Section 3.4 */
    $P_t \leftarrow probabilitySmoothing(P_t, p_0, \rho)$;                         /* Section 3.5 */
until Stopping condition met
```

3.2. Group selection

At each iteration of RLS, each item v_i needs to select a group g_j from the k available groups according to its probability vector p_i. We consider four possible group selection strategies:

- Random selection: the item selects its group at random (regardless of its probability vector). As this selection strategy does not use any useful information collected from the search history, it is expected that this strategy would not perform well.

- Greedy selection: the item always selects the group g_j such that the associated probability p_{ij} has the maximum value. This strategy is intuitively reasonable, but may cause the algorithm to be trapped rapidly.

- Roulette wheel selection: the item selects its group based on its probability vector and the chance for the item to select group g_j is proportional to the probability p_{ij}. Thus a group with a large (small) probability has more (less) chance to be selected.

- Hybrid selection: this strategy combines the random selection and greedy selection strategies in a probabilistic way; with a noise probability ω, random selection is applied; with probability $1-\omega$, greedy selection is applied.

As we show in Section 4.3.3, the group selection strategy greatly affects the performance of the RLS approach. After experimenting the above strategies, we adopted the hybrid selection strategy which combines randomness and greediness and is controlled by the noise probability ω. The purpose of selecting a group with maximum probability (greedy selection) is to make an attempt to correctly select the group for an item that is most often falsified at a local optimum. Selecting such a group for this item may help the search to escape from the current trap. On the other hand, using the noise probability has the advantage of flexibility by switching back and forth between greediness and randomness. Also, this allows the algorithm to occasionally move away from being too greedy. This hybrid group selection strategy proves to be better than the roulette wheel selection strategy, as confirmed by the experiments of Section 4.3.3.

3.3. Descent-based local search for solution improvement

Even if any optimization procedure can be used to improve a given starting grouping solution, for the reason of simplicity, we employ a simple and fast descent-based local search (DB-LS) procedure in this work. To explore the search space, DB-LS iteratively makes transitions from the incumbent solution to a neighboring solution according to a given neighborhood relation such that each transition leads to a better solution. This iterative improvement process continues until no improved solution exists in the neighborhood in which case the incumbent solution corresponds to a local optimum with respect to the neighborhood.

Let Ω denote the search space of the given grouping problem. Let $N : \Omega \rightarrow 2^{\Omega}$ be the neighborhood relation which associates to each solution $S \in \Omega$ a subset of solutions $N(S) \subset \Omega$ (i.e., $N(S)$ is the set of neighboring solutions of S). Typically, given a solution S, a neighboring solution can be obtained by moving an item of S from its current group to another group. Let $f : \Omega \rightarrow \mathbb{R}$ be the evaluation (or cost) function which measures the quality or cost of each grouping solution. The pseudo code of Algorithm 2 displays the general DB-LS procedure.

Descent-based local search can find a local optimum quickly. However, the local optimal solution discovered is generally of poor quality. It is fully possible

Algorithm 2 Pseudo-code of descent-based local search procedure

```
Input: $S$ - an initial candidate grouping solution;
Output: $S^*$ - the local optimum solution attained;
$f(S^*) = f(S)$;
repeat
    choose a best neighbor $S^{''}$ of $S$ such that
    $S^{''} = \arg\min_{S' \in N(S)} f(S)$;
    $S^* = S^{''}$;
    $f(S^*) = f(S^{''})$
    $S = S^*$;
until $f(S^{''}) \geqslant f(S^*)$
```

to improve the performance of RLS by replacing descent-based local search with a more advanced improvement algorithm. In RLS, we make the assumption that, if the item stays in its original group after the descent-based local search, then the item has selected the right group in the original solution, otherwise its new group in the improved solution would be the right group. This assumption can be considered to be reasonable because the descent-based local search procedure is driven by its cost function and each transition from the current solution to a new (neighboring) solution is performed only when the transition leads to an improvement.

3.4. Reinforcement learning - probability updating

Reinforcement learning is defined as how an agent should take actions in an environment so to maximize some notion of cumulative reward. Reinforcement learning acts optimally through trial-and-error interactions with an unknown environment. Actions may affect not only the immediate reward but also the next situation and all subsequent rewards. The intuition underlying reinforcement learning is that actions that lead to large rewards should be made more likely to recur. In RLS, the problem of selecting the most appropriate group for each item is viewed as a reinforcement learning problem. Through the interactions with the unknown environment, RLS evolves and gradually finds the optimal or a suboptimal solution of the problem.

At instant t, we first generate a grouping solution S_t based on the current probability matrix P_t (see Section 3.1). In other words, each item selects one suitable group from the k available groups based on its probability vector (with the group selection strategy of Section 3.2). Then solution S_t is improved by the descent-based local search procedure, leading to an improved solution $\hat{S}_t$. Now, for each item v_i, we compare its groups in S_t and $\hat{S}_t$. If the item stays in its original group (say g_u), we reward the selected group g_u (called correct group) and update its probability vector p_i according to Eq. (1):

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

where α $(0 < \alpha < 1)$ is a reward factor. When item v_i moves from its original group g_u of solution S_t to a new group (say $g_v, v \neq u$) of the improved solution $\hat{S}_t$, we penalize the discarded group g_u (called incorrect group), compensate the new group g_v (called expected group) and finally update its probability vector p_i according to Eq. (2):

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

where β $(0 < \beta < 1)$ and γ $(0 < \gamma < 1)$ are a penalization factor and compensation factor respectively. This process is repeated until each item can select its group correctly. The update of the complete probability matrix P is bounded by $\mathcal{O}(n \times k)$ in terms of time complexity.

Note that our learning scheme is different from general reinforcement learning schemes such as linear reward-penalty, linear reward-inaction and linear reward-ϵ-penalty. The philosophy of these schemes is to increase the probability of selecting an action in the event of success and decrease the probability in the case of a failed signal. Unlike these general schemes, our learning scheme not only rewards the correct group and penalizes the incorrect group, but also compensates the expected group.

3.5. Reinforcement learning - probability smoothing

The intuition behind the probability smoothing technique is that old decisions that were made long ago are no longer helpful and may mislead the current search. Therefore, these aged decisions should be considered less important than the recent ones. In addition, all items are required to correctly select their suitable groups in order to produce a legal grouping solution. It is not enough that only a part of items can correctly select their groups. Based on these two considerations, we introduce a probability smoothing technique to reduce the group probabilities periodically.

Our probability smoothing strategy is inspired by forgetting mechanisms in smoothing techniques in clause weighting local search algorithms for satisfiability (SAT) (Hutter, Tompkins & Hoos, 2002; Ishtaiwi, Thornton, Sattar & Pham, 2005). Based on the way that weights are smoothed or forgotten, there are four available forgetting or smoothing techniques for minimum vertex cover (MVC) and SAT:

- Decrease one from all clause weights which are greater than one like in PAWS (Thornton, Duc, Stuart & Valnir, 2004).
- Pull all clause weights toward their mean value using the formula $w_i = \rho \cdot w_i + (1-\rho) \cdot \overline{w_i}$ like in ESG (Schuurmans, Southey & Holte, 2001) and SAPS (Hutter, Tompkins & Hoos, 2002).

- Transfer weights from neighboring satisfied clauses to unsatisfied ones like in DDWF (Ishtaiwi, Thornton, Sattar & Pham, 2005).

- Reduce all edge weights using the formula $w_i = \lfloor \rho \cdot w_i \rfloor$ when the average weight achieves a threshold like in NuMVC (Cai, Su, Luo & Sattar, 2013).

The probability smoothing strategy adopted in our RLS approach works as follows (see Algorithm 3). For an item, each possible group is associated with a value between 0 and 1 as its probability, and each group probability is initialized as $1/k$. At each iteration, we adjust the probability vector based on the obtained feedback information (i.e., reward, penalize or compensate a group). 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. It is obvious that the smoothing technique used in RLS is different from the above-mentioned four techniques. To the best of our knowledge, this is the first time a smoothing technique is introduced into local search algorithms for grouping problems.

Algorithm 3 Pseudo-code of the probability smoothing procedure

```
Input:
    $P_t$: probability matrix at instant $t$;
    $p_0$: smoothing probability;
    $\rho$: smoothing coefficient;
Output: new probability matrix $P_t$ after smoothing;
for $i = 1$ to $n$ do
    $p_{iw} = max\{p_{ij}, j = 1, 2, ..., k\}$;
    if $p_{iw} > p_0$ then
        for $j = 1$ to $k$ do
            if $j = w$ then
                $p_{ij}(t) = \rho \cdot p_{ij}(t-1)$;
            else
                $p_{ij}(t) = \frac{1-\rho}{k-1} \cdot p_{iw}(t-1) + p_{ij}(t-1)$;
            end if
        end for
    end if
end for
```

4. RLS applied to graph coloring: a case study

This section presents an application of the proposed RLS method to the well-known graph coloring problem which is a typical grouping problem. After presenting the descent-based local search procedure for the problem, we first conduct an experimental analysis of the RLS approach by investigating the influence of its three important components, i.e., the reinforcement learning mechanism, the probability smoothing technique and the group selection strategy. Then we present computational results attained by the proposed RLS method in comparison with a number of existing local search algorithms over well-known DIMACS and COLOR02 benchmark instances.

4.1. Graph coloring and local search coloring algorithm

GCP is one of the most studied combinatorial optimization problems (Garey & Johnson, 1979). GCP is also a relevant representative of grouping problems. Given an undirected graph $G = (V, E)$, where V is the set of $|V| = n$ vertices and E is the set of $|E| = m$ edges, a legal k-coloring of G is a partition of V into k mutually disjoint groups (called color classes) such that two vertices linked by an edge must belong to two different color classes. GCP is to determine the smallest k for a graph G such that a legal k-coloring exists. This minimum number of groups (i.e., colors) required for a legal coloring is the *chromatic number* $\chi(G)$. When the number of color classes k is fixed, the problem is called k-coloring problem (k-GCP for short). As a grouping problem, items in GCP correspond to vertices and groups correspond to color classes.

GCP can be approximated by solving a series of k-GCP (with decreasing k) as follows (Galinier, Hamiez, Hao & Porumbel, 2013). For a given G and a given k, we use our RLS approach to solve k-GCP by seeking a legal k-coloring. If such a coloring is successfully found, we decrease k and solve the new k-GCP again. We repeat this process until no legal k-coloring can be reached. In this case, the last k for which a legal k-coloring has been found represents an approximation (upper bound) of the chromatic number of G. This general solution approach has been used in many coloring algorithms including most of those reviewed below, and is also adopted in our work.

Given the theoretical and practical interest of GCP, a huge number of coloring algorithms have been proposed in the past decades (Galinier, Hamiez, Hao & Porumbel, 2013; Johnson & Trick, 1996; Malaguti & Toth, 2009). Among them, algorithms based on local search are certainly the most popular approaches, like simulated annealing (SA) (Johnson, Aragon, McGeoch & Schevon, 1991), tabu search (TS) (Hertz & de Werra, 1987; Galinier & Hao, 1999), guided local search (GLS) (Chiarandini, 2005), iterated local search (ILS) (Chiarandini & Stützle, 2002), quantum annealing algorithms (Titiloye & Crispin, 2011) and focused walk based local search (FWLS) (Wu, Luo & Su, 2013). Population-based hybrid algorithms represent another class of complex approaches which typically combine local search and dedicated recombination crossover operators (Fleurent & Ferland, 1996; Galinier & Hao, 1999; Lü & Hao, 2010; Malaguti, Monaci & Toth, 2008; Porumbel, Hao & Kuntz, 2010b). Recent surveys of algorithms for GCP can be found in (Galinier, Hamiez, Hao & Porumbel, 2013; Malaguti & Toth, 2009).

To apply the proposed RLS approach to k-GCP, we need to specify three important ingredients of the descent-based local search in RLS, i.e., the search space, the neighborhood and the evaluation function. First, a legal or illegal k-coloring can be represented by $S = \{g_1, g_2, ..., g_k\}$ such that g_i is the group of vertices receiving color i. Therefore, the search space Ω is composed of all possible legal and illegal k-colorings. The evaluation function $f(S)$ counts the number of conflicting edges inducted by S such that:

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

where $\delta(u,v) = 1$, if $u \in g_i, v \in g_j$ and $i = j$, and otherwise $\delta(u,v) = 0$. Accordingly, a candidate solution S is a legal k-coloring S if $f(S) = 0$.

The neighborhood of a given k-coloring is constructed by moving a conflicting vertex v from its original group g_i to another group $g_j (i \neq j)$ (Galinier & Hao, 1999). Therefore, for a k-coloring S with cost $f(S)$, the size of the neighborhood is bounded by $\mathcal{O}(f(S) \times k)$. To evaluate each neighboring solution efficiently, our descent-based local search adopts the fast incremental evaluation technique introduced in (Fleurent & Ferland, 1996; Galinier & Hao, 1999). The principle is to maintain a gain matrix which records the variation $\Delta = f(S') - f(S)$ between the incumbent solution S and every neighboring solution S'. After each solution transition from S to S', only the affected elements of the gain matrix are updated accordingly.

The descent-based local search procedure starts with a random solution taken from the search space Ω and iteratively improves this solution by a neighboring solution of better quality according to the evaluation function f. This process stops either when a legal k-coloring is found (i.e., a solution with $f(S) = 0$), or no better solution exists among the neighboring solutions (in this later case, a local optimum is reached).

4.2. Benchmark instances and experimental settings

We show extensive computational results on two sets of the well-known DIMACS[1] and COLOR02[2] coloring benchmark instances. These instances are the most widely used benchmark instances for assessing the performance of graph coloring algorithms.

The used DIMACS graphs can be divided into six types:

- *Standard random graphs* are denoted as DSJC$n.d$, where n and d indicate respectively the number of vertices and the edge density of the graph.

- *Random geometric graphs* are composed of R125.x, R250.x, DSJR500.x and R1000.x, graphs with letter c in x being complements of geometric graphs.

- *Flat graphs* are structured graphs generated from an equi-partitioning of vertices into k sets. These graphs are denoted as flat$n$_k_0, where n and k are the number of vertices and the chromatic number respectively.

- *Leighton graphs* are random graphs of density below 0.25. These graphs are denoted as le450_kx, where 450 is the number of vertices, $k \in \{15, 25\}$ is the chromatic number of the graph, $x \in \{a, b, c, d\}$ is a letter to indicate different graph instances with the same characteristics.

[1] Available at: ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/

[2] Available at: http://mat.gsia.cmu.edu/COLOR02/

- *Scheduling graphs*, i.e., `school1` and `school1_nsh` come from school timetabling problems.

- *Latin square graph*, i.e., `latin_square_10`.

The used COLOR02 graphs are of three types:

- *Queen graphs* are highly structured instances and their edge density decreases with their size. The graphs are denoted as `queen`x`_`x, where $x \in \{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$, with an exception, i.e., queen8_12.

- *Mycile graphs* are denoted as `mycile`k, where $k \in \{3, 4, 5, 6, 7\}$. These graphs are based on the Mycielski transformation.

- *Miles Graphs* (`miles`x, with $x \in \{250, 500, 750, 1000, 1500\}$) are similar to geometric graphs in that vertices are placed in space with two vertices connected if they are close enough.

Table 1: Parameters of Algorithm RLS

Parameters	Section	Description	Values
ω	3.2	noise probability	0.200
α	3.4	reward factor for correct group	0.100
β	3.4	penalization factor for incorrect group	(0, 0.45]
γ	3.4	compensation factor for expected group	0.300
ρ	3.5	smoothing coefficient	0.500
p_0	3.5	smoothing threshold	0.995

Our RLS algorithm was coded in C and compiled using GNU g++ on a computer with an Intel E5440 processor with 2.83 GHZ and 2G RAM under Linux. To obtain our experimental results, each instance was solved 20 times independently with different random seeds. Each execution was terminated when a legal k-coloring is found or the number of iterations without improvement reaches its maximum allowable value ($I_{max} = 10^6$). In our experiments, all parameters were fixed except for the penalization factor β that varies between 0 and 0.45. Table 1 shows the descriptions and setting of the parameters used for our experiments.

4.3. Analysis of key components of the RLS approach

We first show an analysis of the main ingredients of the RLS approach: reinforcement learning mechanism, probability smoothing technique and group selection strategies. This study allows us to better understand the behavior of the proposed RLS approach and shed light on its inner functioning.

Table 2: Comparative results of RLS (with reinforcement learning) and RLS_0 (without reinforcement learning) on the DIMACS graphs. Smaller k values are better

		RLS_0			RLS			
Instance	χ/k^*	k_1	#hit	time(s)	k_2	#hit	time(s)	Δk
DSJC125.1	5/5	6	12/20	8.71	**5**	20/20	5.68	-1
DSJC125.5	17/17	22	17/20	42.03	**17**	15/20	38.40	-5
DSJC125.9	44/44	51	05/20	155.21	**44**	20/20	9.56	-7
DSJC250.1	?/8	11	20/20	42.91	**8**	20/20	53.11	-3
DSJC250.5	?/28	40	04/20	238.92	**29**	20/20	91.41	-11
DSJC250.9	?/72	94	20/20	572.20	**75**	01/20	181.36	-19
DSJC500.1	?/12	18	02/20	747.64	**13**	20/20	16.70	-5
DSJC500.5	?/47	74	04/20	3768.51	**50**	09/20	1713.95	-24
DSJC1000.1	?/20	32	18/20	8730.46	**21**	20/20	1223.18	-11
DSJR500.1	12/12	13	05/20	281.48	**12**	20/20	1.91	-1
DSJR500.1c	85/85	97	01/20	662.43	**85**	02/20	699.63	-12
flat300_20_0	20/20	44	02/20	887.89	**20**	10/20	99.06	-24
flat300_26_0	26/26	45	02/20	359.32	**26**	19/20	450.28	-19
flat300_28_0	28/28	45	02/20	1035.29	**32**	19/20	173.32	-13
flat1000_76_0	76/81	135	01/20	2779.32	**89**	02/20	11608.95	-46
le450_15a	15/15	21	09/20	296.46	**15**	19/20	103.00	-6
le450_15b	15/15	21	20/20	217.78	**15**	09/20	326.27	-6
le450_15c	15/15	30	01/20	640.65	**15**	16/20	307.91	-15
le450_15d	15/15	31	19/20	464.74	**15**	14/20	210.90	-16
le450_25a	25/25	28	18/20	171.98	**26**	20/20	14.29	-2
le450_25b	25/25	26	01/20	346.29	**25**	01/20	90.37	-1
le450_25c	25/25	37	14/20	720.34	**26**	13/20	181.39	-11
le450_25d	25/25	36	03/20	935.19	**26**	07/20	438.22	-10
R125.1	5/5	**5**	20/20	< 0.01	**5**	20/20	< 0.01	0
R125.1c	46/46	**46**	15/20	166.58	**46**	20/20	0.28	0
R125.5	?/36	42	07/20	80.26	**38**	01/20	28.66	-4
R250.1	8/8	**8**	20/20	0.04	**8**	20/20	< 0.01	0
R250.1c	64/64	67	03/20	767.12	**64**	20/20	14.40	-3
R1000.1	20/20	24	20/20	300.60	**21**	20/20	261.90	-3
school1	14/14	39	05/20	742.21	**14**	18/20	17.65	-25
school1_nsh	14/14	36	02/20	674.86	**14**	19/20	1.83	-22
latin_square_10	?/97	169	02/20	3961.96	**99**	10/20	12946.53	-70

4.3.1. Effectiveness of the reinforcement learning mechanism

To verify the effectiveness of the reinforcement learning mechanism used in RLS, we made a comparison between RLS and its variant RLS$_0$ where we removed the reinforcement learning mechanism from RLS and randomly restarted the search each time the DB-LS procedure attains a local optimum.

The investigation was conducted on the 32 DIMACS instances and each algorithm was run 20 times to solve each instance. The comparative results of RLS and RLS$_0$ are provided in Table 2. For each graph, we list the known chromatic number χ ('?' if χ is still unknown) and the best k^* (upper bound of χ) reported in the literature. For each algorithm, we indicate the best (the smallest) k value for which the algorithm attains a legal k-coloring, the number of such successful runs over 20 executions (#hit), and the time in seconds to hit the reported k-coloring averaged over the successful runs (time(s)). The differences between the best k of RLS and the best k of RLS$_0$ are provided in the last column (Δk). The results show that RLS significantly outperforms RLS$_0$ in terms of the best k value for 29 out of 32 instances (indicated in bold). For example, on instance `flat300_26_0`, RLS attains the chromatic number k (i.e., $\chi = 26$) while RLS$_0$ needs 45 colors to color it legally. Specially, we observe that RLS has a larger improvement on hard instances than on easy instances. For example, `latin_square_10` and `flat1000_76_0` are two well-known hard instances, RLS achieves the two largest improvements, i.e., using 70 and 46 fewer colors than RLS$_0$. In summary, RLS attains better results on 29 out of 32 instances compared to its variant with the reinforcement learning mechanism disabled. This experiment confirms the effectiveness of the reinforcement learning mechanism to help the descent-based local search to attain much better results.

4.3.2. Effectiveness of the probability smoothing technique

To study the effectiveness of the probability smoothing technique used in RLS, we compare RLS with its alternative algorithm RLS$_1$, which is obtained from RLS by adjusting the probability updating scheme. More specifically, RLS$_1$ works in the same way as RLS, but it does not use the probability smoothing strategy, that is, line 10 in Algorithm 1 is removed. For this experiment, by following (Galinier & Hao, 1999), we use *running profiles* to observe the change of evaluation function f over the number of iterations. Running profiles provide interesting information about the convergence of the studied algorithms.

The running profiles of RLS and RLS$_1$ are shown in Figure 3 on two selected instances: Fig. 3(a) for `flat300_28_0` ($k = 32$), and Fig.3(b) for `latin_square_10` ($k = 101$). We observe that though both algorithms successfully obtain a legal k-coloring, RLS converges to the best solution more quickly than RLS$_1$, i.e., the objective value f of RLS decreases more quickly than that of RLS$_1$. Consequently, RLS needs less iterations to attain a legal solution. This experiment demonstrates the benefit of using probability smoothing technique in RLS.

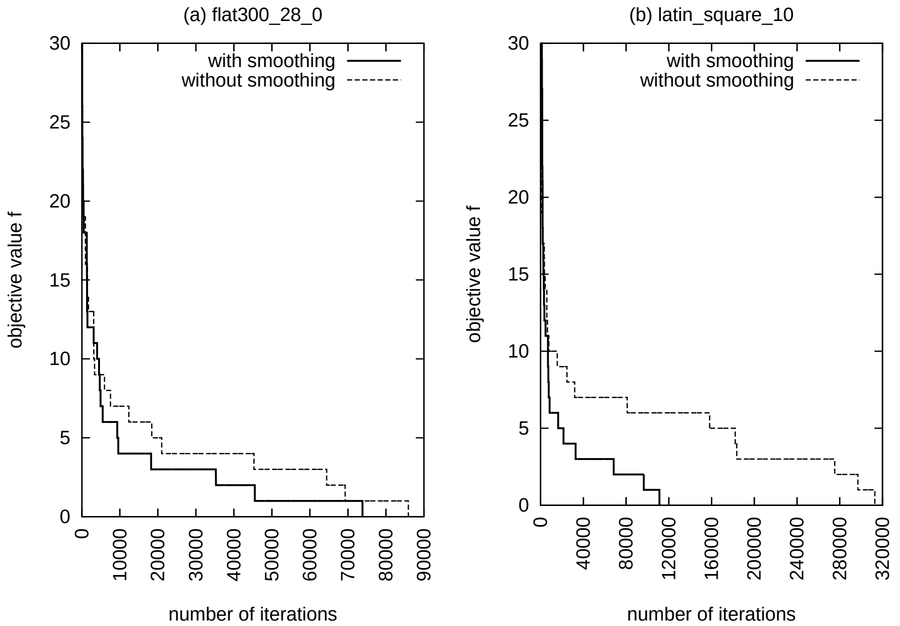


Figure 3: Running profile of RLS (with smoothing) and RLS$_1$ (without smoothing) on instance `flat300_28_0` and `latin_square_10`

4.3.3. Comparison of different group selection strategies

The group selection strategy plays an important role in RLS. At each iteration, each vertex selects a suitable group based on the group selection strategy to produce a new solution for the next round of the descent-based local search optimization. In this section, we show an analysis of the group selection strategies to confirm the interest of the adopted hybrid strategy which combines random and greedy strategies.

The investigation was carried out between RLS and its variant RLS$_2$, obtained from RLS by means of replacing the hybrid group selection strategy with the roulette wheel selection strategy. In the experiment, each instance was tested 20 times independently with different random seeds. The number of successful runs, the average number of iterations and the average running time of successful runs are reported.

Table 3 reports the comparative results of this experiment on four chosen instances. The results indicate that RLS significantly outperforms RLS$_2$ in terms of the best k value and the number of successful running times (or the number of iterations). For example, on instance `DSJR500.1c`, RLS colors this graph with 85 colors, while RLS$_2$ needs two more colors ($k = 87$) to color it. A similar observation can be found on instance `le450_25c`, for which RLS obtains a legal 26-coloring, while RLS$_2$ only obtains a 27-coloring. Furthermore, when they need the same number of colors to color a graph, RLS typically achieves

Table 3: Comparative performance of RLS (with its hybrid group selection strategy) and RLS_2 (with the roulette wheel selection strategy). Smaller k and larger #hit are better

	RLS_2			RLS		
Instance	k_1(#hit)	#iter	time(s)	k_2(#hit)	#iter	time(s)
le450_25c	26(0/20)	-	-	26(13/20)	4.7×10^6	181.39
	27(20/20)	7.0×10^5	26.86	27(20/20)	1.5×10^6	61.13
DSJR500.1	12(0/20)	-	-	12(20/20)	7.8×10^4	1.91
	13(20/20)	2.0×10^6	50.42	13(20/20)	3.0×10^3	0.10
DSJR500.1c	85(0/20)	-	-	85(02/20)	4.6×10^6	699.63
	86(0/20)	-	-	86(20/20)	3.6×10^6	529.47
	87(20/20)	3.2×10^6	361.97	87(20/20)	6.9×10^5	108.12
DSJC1000.1	21(09/20)	2.0×10^7	1508.48	21(20/20)	1.4×10^7	1223.18
	22(20/20)	6.0×10^5	41.82	22(20/20)	8.0×10^5	64.77

the result with a smaller number of iterations. This experiment confirms the interest of the adopted hybrid selection strategy.

4.4. Computational results of RLS and comparisons

We now turn our attention to a comparative study of the proposed RLS approach with respect to some well-known coloring algorithms in the literature. The five reference algorithms are all based on *advanced local search* methods including the prominent simulating annealing (SA) algorithm (Johnson, Aragon, McGeoch & Schevon, 1991), the improved tabu search (TS) algorithm (Galinier & Hao, 1999), the guided local search (GLS) algorithm (Chiarandini, 2005), the iterative local search (ILS) algorithm (Chiarandini & Stützle, 2002) and the focused walk based local search (FWLS) algorithm (Wu, Luo & Su, 2013). Following the literature, we focus on the quality of the solutions found, i.e., the smallest number of colors k needed for an algorithm to find a legal k-coloring for a given graph. Indeed, given that the compared algorithms were tested on different computing platforms with various stopping conditions, it seems difficult to exactly compare the computing times. For indicative purposes, we include the running times of the proposed RLS approach, which remain comparable with those of the compared algorithms (ranging from seconds to hours). This comparison is not exhaustive, yet it allows us to assess the interest of using the adopated learning mechanism to boost a very simple descent procedure.

We present in Tables 4 and 5 the results of RLS together with the best solutions of the reference algorithms. We list the number of vertices (n) and edges (m) of each graph, the known chromatic number χ ('?' if χ is unknown) and the best k^* (upper bound of χ) reported in the literature. For each algorithm, we list the best (the smallest) k for which a legal k-coloring is attained. For the proposed RLS algorithm, we additionally give the number of successful runs over 20 executions (#hit) and the average time in seconds to hit the reported k-coloring over the successful runs (time(s)). A summary of the comparisons between our RLS algorithm and each reference algorithm is provided at the

Table 4: Comparative results of RLS and five advanced local search algorithms on DIMACS graphs

				RLS			TS	SA	GLS	ILS	FWLS
Instance	n	m	χ/k^*	k	#hit	time(s)	k	k	k	k	k
DSJC125.1	125	736	5/5	**5**	20/20	5.68	5	6	5	5	5
DSJC125.5	125	3,891	17/17	**17**	15/20	38.40	17	18	18	17	17
DSJC125.9	125	6,961	44/44	**44**	20/20	9.56	44	44	44	44	45
DSJC250.1	250	3,218	?/8	**8**	20/20	53.11	8	9	8	8	8
DSJC250.5	250	15,668	?/28	29	20/20	91.41	29	29	29	28	29
DSJC250.9	250	27,897	?/72	75	01/20	181.36	72	72	72	72	73
DSJC500.1	500	12,458	?/12	**13**	20/20	16.70	13	14	13	13	13
DSJC500.5	500	62,624	?/47	**50**	09/20	1713.95	50	51	52	50	51
DSJC1000.1	1,000	49,629	?/20	**21**	20/20	1223.18	21	23	21	21	21
DSJR500.1	500	3,555	12/12	**12**	20/20	1.91	12	-	-	12	-
DSJR500.1c	500	121,275	85/85	**85**	02/20	699.63	94	89	85	-	-
flat300_20_0	300	21,375	20/20	**20**	10/20	99.06	20	20	20	20	-
flat300_26_0	300	21,633	26/26	**26**	19/20	450.28	26	32	33	26	26
flat300_28_0	300	21,695	28/28	32	19/20	173.32	32	33	33	31	28
flat1000_76_0	1,000	246,708	76/81	**89**	02/20	11608.95	91	89	92	89	90
le450_15a	450	8,168	15/15	**15**	19/20	103.00	15	16	15	15	15
le450_15b	450	8,169	15/15	**15**	09/20	326.27	15	16	15	15	15
le450_15c	450	16,680	15/15	**15**	16/20	307.91	16	23	15	15	15
le450_15d	450	16,750	15/15	**15**	14/20	210.90	16	22	15	15	15
le450_25a	450	8,260	25/25	26	20/20	14.29	25	-	-	-	25
le450_25b	450	8,263	25/25	**25**	01/20	90.37	25	-	-	-	25
le450_25c	450	17,343	25/25	**26**	13/20	181.39	26	27	26	26	26
le450_25d	450	17,425	25/25	**26**	07/20	438.22	26	28	26	26	26
R125.1	125	209	5/5	**5**	20/20	< 0.01	5	-	-	-	-
R125.1c	125	7,501	46/46	**46**	20/20	0.28	47	-	-	-	-
R125.5	125	3,838	?/36	38	01/20	28.66	36	-	-	-	-
R250.1	250	867	8/8	**8**	20/20	< 0.01	8	-	-	-	-
R250.1c	250	30,227	64/64	**64**	20/20	14.40	72	-	-	-	-
R1000.1	1,000	14,348	20/20	21	20/20	261.90	20	-	-	-	-
school1	385	19,095	14/14	**14**	18/20	17.65	14	14	14	-	14
school1_nsh	352	14,612	14/14	**14**	19/20	1.83	14	14	14	-	14
latin_square_10	900	307,350	?/97	**99**	04/20	12946.53	105	101	102	103	-
better			0/32	-	-	-	7/32	16/23	5/22	1/21	3/22
equal			18/32	-	-	-	21/32	6/23	16/22	17/21	17/22
worse			14/32	-	-	-	4/32	1/23	1/22	3/21	2/22

bottom of these tables. The rows 'better', 'equal', and 'worse' respectively indicate the number of instances for which our RLS algorithm achieves a better, an equal, and a worse result compared to each reference algorithm over the total number of instances for which the reference algorithm reported its results. The results of the reference algorithms are extracted from the literature except for TS which was run on the same computing platform as RLS. In these tables, '−' indicates that the result of the algorithm on this instance is unavailable in the literature. When a result of RLS is no worse than any result of the competing algorithms, the result is marked in bold.

Table 5: Comparative results of RLS and five advanced local search algorithms on COLOR02 graphs

				RLS			TS	SA	GLS	ILS	FWLS
Instance	n	m	χ/k^*	k	#hit	time(s)	k	k	k	k	k
miles250	128	387	8/8	**8**	20/20	< 0.01	8	-	-	-	-
miles500	128	1,170	20/20	**20**	20/20	0.02	20	-	-	-	-
miles750	128	2,113	31/31	**31**	20/20	0.09	31	-	-	-	-
miles1000	128	3,216	42/42	**42**	20/20	5.83	42	-	-	-	-
miles1500	128	5,189	73/73	**73**	20/20	0.47	73	-	-	-	-
myciel3	11	20	4/4	**4**	20/20	< 0.01	4	-	-	-	4
myciel4	23	71	5/5	**5**	20/20	< 0.01	5	-	-	-	5
myciel5	47	236	6/6	**6**	20/20	< 0.01	6	-	-	-	6
myciel6	95	755	7/7	**7**	20/20	< 0.01	7	-	-	-	7
myciel7	191	2,360	8/8	**8**	20/20	< 0.01	8	-	-	-	8
queen5_5	25	160	5/5	**5**	20/20	< 0.01	5	-	-	-	5
queen6_6	36	290	7/7	**7**	20/20	0.07	7	7	7	7	7
queen7_7	49	476	7/7	**7**	20/20	0.31	7	7	7	7	7
queen8_8	64	728	9/9	**9**	20/20	0.09	9	9	9	9	9
queen8_12	96	1,368	12/12	**12**	20/20	0.83	12	12	12	12	12
queen9_9	81	2,112	10/10	**10**	20/20	1.18	10	10	10	10	10
queen10_10	100	2,940	?/11	**11**	20/20	7.96	11	11	11	11	11
queen11_11	121	3,960	?/11	**12**	20/20	11.71	12	12	12	12	12
queen12_12	144	5,192	?/12	**13**	20/20	25.46	13	14	13	13	13
queen13_13	169	6,656	?/13	**14**	20/20	17.60	14	15	15	14	14
queen14_14	196	8,372	?/14	**15**	14/20	36.66	15	16	16	15	15
queen15_15	225	10,360	?/15	**16**	11/20	39.49	16	17	17	16	16
queen16_16	256	12,640	?/16	**17**	11/20	266.02	18	17	18	18	18
better			0/23	-	-	-	1/23	4/12	4/12	1/12	1/18
equal			17/23	-	-	-	22/23	8/12	8/12	11/12	17/18
worse			6/23	-	-	-	0/23	0/12	0/12	0/12	0/18

From Table 4 which concerns the DIMACS graphs, we observe that RLS is competitive and even achieves a better performance on some graphs. For instance, compared to SA, RLS finds 16 better solutions out of the 23 instances tested by SA. The comparison with TS is more informative and meaningful given that RLS and TS share the same data structures, both were programmed in C and were run on the same computer. We observe that despite the simplicity of its descent-based coloring procedure, RLS attains 7 better results (v.s. 4 worse results) compared to TS. Additionally, we observe that RLS achieves more 'better' results than 'worse' results compared to the reference algorithms except for ILS. Nevertheless, since the results of ILS for 9 instances are unavailable, it is difficult to draw a clear conclusion.

When we compare our RLS algorithm with the five reference algorithms on the COLOR02 graph instances (Table 5), we observe that RLS dominates these algorithms. Specifically, RLS achieves no worse results on these instances than any of the reference algorithms, and obtains 4 better solutions than SA and GLS, 1 better solution than TS, ILS, and FWLS respectively.

We also find that the proposed RLS method even achieves competitive performances compared to some complex population algorithms proposed in recent years, such as ant-based algorithm (Bui, Nguyen, Patel & Phan, 2008) (2008), and modified cuckoo optimization algorithm (Mahmoudi & Lotfi, 2015) (2015). Meanwhile, given the very simplicity of its underlying local search procedure, it is no surprise that RLS alone cannot compete with the most powerful population-based coloring algorithms like (Galinier & Hao, 1999; Lü & Hao, 2010; Malaguti, Monaci & Toth, 2008; Porumbel, Hao & Kuntz, 2010b; Titiloye & Crispin, 2011; Wu & Hao, 2012). Indeed, these algorithms are typically complex hybrid algorithms mixing several approaches like evolutionary computing and local optimization. On the other hand, given the way the proposed RLS approach is composed, it would be interesting to replace the simple descent-based local search by any of these advanced coloring algorithms and investigate the proposed reinforcement learning mechanism in comparison with these advanced coloring algorithms.

5. Conclusion and discussion

In this paper, we proposed a reinforcement learning based optimization approach for solving the class of grouping problems. The proposed RLS approach combines reinforcement learning techniques with a descent-based local search procedure. Reinforcement learning is used to maintain and update a set of probability vectors, each probability vector specifying the probability that an item belongs to a particular group. At each iteration, RLS builds a starting grouping solution according to the probability vectors and with the help of a group selection strategy. RLS then applies a descent-based local search procedure to improve the given grouping solution until a local optimum is reached. By comparing the starting solution and the ending local optimum solution, the probability vector of each item is updated accordingly with the help of a reward-penalty-compensation strategy. To the best of our knowledge, this is the first attempt to propose a formal model that combines reinforcement learning and local search for solving grouping problems.

The viability of the proposed approach is verified on the well-known graph coloring problem (a representative grouping problem). Our experimental outcomes indicated that RLS, despite the simplicity of its basic coloring procedure, competes favorably in comparison with five popular local search algorithms. Meanwhile, given the competitiveness of the graph coloring problem, RLS cannot really competes with the most advanced coloring algorithms which are based on population-based complex hybrid schemes. It would be interesting to investigate the benefit of combining the proposed reinforcement learning techniques with these approaches, which constitutes our ongoing research.

The current work can be improved according to two directions. First, the proposed RLS approach relies on a pure descent-based local search procedure for its local improvement phase. Even if a descent procedure can quickly converge to local optimal solutions, it is not reasonable to expect very high quality solutions from such a basic search procedure. Fortunately, since the local improvement phase in RLS can be considered as a black-box, one can replace the descent search by any advanced optimization procedure to reinforce the performance of the whole RLS framework. Second, the learning phase of the proposed approach is based on a probability matrix which needs to be updated upon the discovery of a new local optimal solution, which induces an extra-computational cost. To lower this cost, it would be interesting to develop streamlining techniques that can ensure fast and incremental probability updates.

To conclude, this work demonstrates that reinforcement learning constitutes a valuable means to increase the performance of an optimization algorithm by learning useful information from discovered local optimum solutions. Given that the proposed reinforcement learning based local search is a general-purpose approach, it is expected that this approach will be adapted to solve other relevant grouping problems including those arising from expert and intelligent systems like bin packing and data clustering.

Acknowledgments

We are grateful to the reviewers for their helpful comments and suggestions which helped us to improve the paper. This work was partially supported by the PGMO project (2014-2015, Jacques Hadamard Mathematical Foundation, Paris). The financial support for Yangming Zhou from the China Scholarship Council (CSC, 2014-2018) is acknowledged.

References

Agustn-Blas, L., Salcedo-Sanz, S., Jiménez-Fernández, S., Carro-Calvo, L., Ser, J.D. & Portilla-Figueras, J. (2012), A new grouping genetic algorithm for clustering problems, Expert Systems with Applications, 39, 9695-9703.

Ahmed, L.N., Özcan, E., & Kheiri A. (2013), Solving high school timetabling problems worldwide using selection hyper-heuristics, Expert Systems with Applications, 42(13), 5463-5471.

Baluja, S., Barto, A., Boese, K., Boyan, J., Buntine, W., Carson, T., Caruana, R., Cook, D., Davies, S., Dean, T., & others (2000). Statistical machine learning for large-scale optimization, Neural Computing Surveys, 3, 1-58.

Battiti, R., & Brunato, M. (2014). The LION way. Machine Learning plus Intelligent Optimization, LIONlab, University of Trento.

Boyan, J.A., & Moore, A.W. (2001). Learning evaluation functions to improve optimization by local search, The Journal of Machine Learning Research, 1, 77-112.

Bui, T.N., Nguyen, T.H., Patel, C.M., & Phan, K.T. (2008). An ant-based algorithm for coloring graphs, Discrete Applied Mathematics, 156, 190-200.

Burke, E., Kendall, G., & Soubeiga, E. (2003). A Tabu-Search Hyper-heuristic for Timetabling and Rostering, Journal of Heuristics, 9, 451-470.

Cai, S., Su, K., Luo, C., & Sattar, A. (2013). NuMVC: An efficient local search algorithm for minimum vertex cover, Journal of Artificial Intelligence Research, 46(1), 687-716.

Ceberio, J., Mendiburu, A., & Lozano, J. A. (2013). The Plackett-Luce ranking model on permutation-based optimization problems, Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 494-501.

Chiarandini, M. (2005). Stochastic local search methods for highly constrained combinatorial optimisation problems, Ph.D. thesis, Technical University of Darmstadt.

Chiarandini, M., & Stützle, T. (2002). An application of iterated local search to graph coloring problem, Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, pp. 112-125.

Demange, M., Monnot, J., Pop P.C., & Reis, B. (2012). Selective graph coloring in some special classes of graphs, Lecture Notes in Computer Science, 7422, 320-331.

Demange, M., Monnot, J., Pop P.C., & Reis, B. (2014). On the complexity of the selective graph coloring problem in some special classes of graphs, Theoretical Computer Science, 540-541, 82-102.

Elhag, A., & Özcan, E. (2015). A grouping hyper-heuristic framework: Application on graph coloring, Expert Systems with Applications, 42, 5491-5507.

Falkenauer, E. (1998). Genetic Algorithms and Grouping Problems, John Wiley & Sons, Inc.

Fidanova, S., & Pop, P.C. (2016). An improved hybrid ant-local search search for the partition graph coloring problem, Journal of Computational and Applied Mathematics, 293, 55-61.

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

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

Galinier, P., Hamiez, J.P., Hao, J.K., & Porumbel, D. (2013). Recent advances in graph vertex coloring, In I. Zelinka, A. Abraham, V. Snasel (Eds.), Handbook of Optimization, 505-528.

Garey, M., & Johnson, D. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., San Franc., USA.

Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. INFORMS Journal on Computing, 21, 178-192.

Guo, Y., Goncalves, G., & Hsu, T. (2013). 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), 45-66.

Hafiz, F. & Abdennour, A. (2016), Particle Swarm Algorithm variants for the Quadratic Assignment Problems - A probabilistic learning approach, Expert Systems with Applications, 2016, 44, 413-431.

Hamiez, J.P., & Hao, J.K. (1993), An analysis of solution properties of the graph coloring problem. Metaheuristics: Computer Decision-Making, Chapter 15, pp325-346, Resende M.G.C. and de Sousa J.P. (Eds.), Kluwer.

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

Hutter, F., Tompkins, D.A.D., & Hoos, H.H. (2002). Scaling and probabilistic smoothing: efficient dynamic local search for SAT, Principles and Practice of Constraint Programming (CP), pp. 233-248.

Hutter, F., Xu, L., Hoos, H.H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: methods and evaluation, Artificial Intelligence, 206, 79-111.

Ishtaiwi, A., Thornton, J., Sattar, A., & Pham, D.N. (2005). Neighbourhood clause weight redistribution in local search for SAT, Principles and Practice of Constraint Programming (CP), pp. 772-776.

Jin, Y., Hamiez, J.P., & Hao, J.K. (2016). Algorithms for the minimum sum coloring problem: a review. Artificial Intelligence Review, DOI 10.1007/s10462-016-9485-7 (in press).

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

Johnson, D.S., & Trick M. (1996). Cliques, Coloring, and Satisfiability, DIMACS Series in Discrete Math. and Theor. Comput. Sci. , vol. 26, pp. 335-357. Am. Math. Soc., New Providence, USA.

Kashan, A. H., Kashan, M.H., & Karimiyan, S. (2013). A particle swarm optimizer for grouping problems, Information Sciences, 252, 81-95.

Lai, X.J., & Hao, J.K. (2015). Path relinking for the fixed spectrum frequency assignment problem. Expert Systems with Applications, 42(10), 4755-4767.

Lai, X.J., Hao, J.K., Lü, Z.P., & Glover, F. (2016). A learning-based path relinking algorithm for the bandwidth coloring problem. Engineering Applications of Artificial Intelligence, 52, 81-91.

Lewis, R. (2009). A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing, Computers & Operations Research, 36(7), 2295-2310.

Lewis, R., & Paechter, B. (2007). Finding feasible timetables using group-based operators, IEEE Transactions on Evolutionary Computation, 11(3), 397-413.

Lü, Z., & Hao, J. (2010). A memetic algorithm for graph coloring, European Journal of Operational Research, 203, 241-250.

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

Malaguti, E., & Toth, P. (2009). A survey on vertex coloring problems, International Transactions in Operational Research, 17(1), 1-34.

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

Miagkikh, V.V., & Punch III, W. F. (1999). Global search in combinatorial optimization using reinforcement learning algorithms, Proceedings of the Congress on Evolutionary Computation (CEC).

Pop, P.C., Hu B., & Raidl, G.R. (2013). A memetic algorithm with two distinct solution representations for the partition graph coloring problem, Lecture Notes in Computer Science, 8111, 219-227.

Porumbel, D.C., Hao, J.K., & Kuntz, P. (2010a). A search space “cartography” for guiding graph coloring heuristics, Computers & Operations Research, 37, 769-778.

Porumbel, D.C., Hao, J.-K., & Kuntz, P. (2010b). An Evolutionary Approach with Diversity Guarantee and Well-Informed Grouping Recombination for Graph Coloring, Computers & Operations Research 37(10), 1822–1832.

Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jiménez, J., Gómez, C., Huacuja, H.J., & Alvim, A.C. F. (2015). A grouping genetic algorithm with controlled gene transmission for the bin packing problem, Computers & Operations Research, 55, 52-64.

Schuurmans, D., Southey, F., & Holte, R.C. (2001). The exponentiated subgradient algorithm for heuristic boolean programming, In Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI), pp. 334-341.

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

Thornton, J., Duc, N.P., Stuart B., & Valnir F.Jr. (2004). Additive versus multiplicative clause weighting for SAT, In Proceedings of the Conference of the American Association for Artificial Intelligence (AAAI), pp. 191-196.

Titiloye, O., & Crispin, A. (2011). Quantum annealing of the graph coloring problem, Discrete Optimization, 8, 376-384.

Torkestani, J.A. & Meybodi, M. R. (2011). A cellular learning automata-based algorithm for solving the vertex coloring problem, Expert Systems with Applications, 38, 9237-9247.

Wu, Q., & Hao, J.K. (2012). Coloring large graphs based on independent set extraction, Computers & Operations Research, 39(2), 283-290.

Wu, W., Luo, C., & Su, K. (2013). FWLS: A Local Search for Graph Coloring, Frontiers in Algorithmics and Algorithmic Aspects in Information and Management, Third Joint International Conference, Proceedings, pp. 84-93.

Xin, Y., Xie, Z.Q., & Yang J. (2013). An adaptive random walk sampling method on dynamic community detection, Expert Systems With Applications, 58, 10-19.

Xu, Y., Stern, D., & Samulowitz, H. (2009). Learning adaptation to solve constraint satisfaction problems, Proceedings of Learning and Intelligent Optimization (LION III), January 14-18, Trento, Italy, https://www.microsoft.com/en-us/research/publication/learning-adaptation-to-solve-constraint-satisfaction-problems/