# Gradient Descent based Weight Learning for Grouping Problems: Application on Graph Coloring and Equitable Graph Coloring

Olivier Goudet [a], Béatrice Duval [a], Jin-Kao Hao [a,b,*]

[a] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[b] *Institut Universitaire de France, 1 Rue Descartes, 75231 Paris, France*

**Abstract**

A grouping problem involves partitioning a set of items into mutually disjoint groups or clusters according to some guiding decision criteria and imperative constraints. Grouping problems have many relevant applications and are computationally difficult. In this work, we present a general weight learning based optimization framework for solving grouping problems. The central idea of our approach is to formulate the task of seeking a solution as a real-valued weight matrix learning problem that is solved by first order gradient descent. A practical implementation of this framework is proposed with tensor calculus in order to benefit from parallel computing on GPU devices. To show its potential for tackling difficult problems, we apply the approach to two typical and well-known grouping problems (graph coloring and equitable graph coloring). We present large computational experiments and comparisons on popular benchmarks and report improved best-known results (new upper bounds) for several large graphs.

*Keywords*: Learning-based problem solving; heuristics; gradient descent; combinatorial search problems; grouping problems; graph coloring.

## 1 Introduction

Grouping problems generally involves partitioning a set of items into mutually disjoint groups or clusters according to some guiding decision criteria and

* Corresponding author.
*Email addresses:* `olivier.goudet@univ-angers.fr` (Olivier Goudet), `beatrice.duval@univ-angers.fr` (Béatrice Duval), `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao).

imperative constraints [12]. For example, given a graph $G = (V, E)$ with node set $V$ and edge set $E$, the popular NP-hard graph $k$-coloring problem is to color the nodes of $V$ with $k$ available colors in such a way that two nodes linked by an edge in $E$ must receive two different colors. Equivalently, this problem can be stated in terms of a grouping problem, i.e., to partition the nodes of $V$ into $k$ color groups such that each color group gathers the nodes receiving the same color. Additional examples of grouping problems include other coloring problems (e.g., equitable coloring and sum coloring) [27], and well-known NP-hard problems such as graph $k$-partitioning [4,8], bin packing [10] and maximally diverse grouping [7,25]. Grouping problems are also useful models to formulate a number of relevant applications related to, e.g., assembly line balancing [11], timetabling [28], single batchmachine scheduling [23] and routing [1]. More illustrations of grouping problems and their applications can be found in [12,23,46].

From a perspective of computational complexity, many grouping problems including those mentioned above are difficult search problems. As a result, finding satisfactory solutions to such problems represents a real challenge from an algorithmic perspective. Given the importance of grouping problems, various solution methods have been investigated in the literature to solve grouping problems. However, a majority of existing studies on grouping problems focuses on particular problems (e.g. the above-mentioned problems) and it is difficult to generate their results to other settings. Among the scarce general studies on grouping problems, grouping genetic algorithms (GGA) introduced in [12] are certainly the most representative and significant. As a specialization of the general framework of genetic algorithms, GGA defines a special group encoding scheme and the associated genetic operators that manipulate groups instead of group members. Following the idea of GGA, a grouping version of the particle swarm optimization algorithm (PSO) was adapted to grouping problems in [23]. In the adapted method (called GPSO), the particle position and velocity updating equations of PSO are modified to preserve the major characteristics of the original PSO equations and to take into account the structure of grouping problems. Contrary to GGA examining discrete spaces, GPSO works in both continuous and discrete spaces. Recently, an original reinforcement learning based local search approach (RLS) was proposed in [46]. RLS introduces the idea of using a probability matrix to specify an item-to-group assignment and uses information learned from improved solutions given by local search to update the probability matrix. As such, the method learns a generative model of solution that is used to iteratively seed a local search procedure. RLS provides one of the main inspirations of our work.

In this work we are interested in solving grouping problems from a general perspective by proposing a generic method that can be applied to different grouping problems. The contributions of this work can be summarized as follows.

First, we introduce a weight learning based solution framework, called Gp-Grad, for solving grouping problems. We adopt the idea of using a weight matrix to specify a relative confidence score for each item to belong to each group. This weight matrix is updated step by step by gradient descent based learning in order to reach a legal solution. This approach benefits from GPU accelerated training and is able to deal with multiple parallel solution evaluations. As the first study of using gradient descent based weight learning to solve grouping problems, this work enriches the still limited toolkit of general solution methods for this important class of problems.

Second, we demonstrate the usefulness of the proposed framework by applying it on two well-known NP-hard grouping problems: the conventional Graph Coloring Problem (GCP) and the Equitable graph Coloring Problem (ECP), which have been intensively studied in the literature (see e.g., [16,30,42,35,45] for the GCP and [9,26,41,43] for the ECP). For both problems, we present large computational experiments on well-known benchmark graphs and show the competitiveness of our approach compared to the best performing state-of-the-art algorithms. In particular, we report improved best results for several large geometric graphs and large random graphs for the ECP.

Third, this work shows the viability of formulating a discrete grouping problem as a real-valued weight learning problem. The competitive results on two challenging representative grouping problems invite more investigations of testing the proposed approach to other grouping problems and applications.

The remainder of the paper is organized as follows. Section 2 describes the proposed framework for solving grouping problems. Sections 3 and 4 are devoted to the application of the framework to the GCP and ECP respectively. Section 5 provides illustrative toy examples on the gradient descent learning. Section 6 reports on computational results of our algorithms compared to the state of the art. Section 7 reviews related heuristics of the literature for the GCP and ECP. Section 8 discusses the contributions and presents perspectives for future work.

## 2 Gradient descent based learning for grouping problems

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 $g_i$ ($i = 1, ..., k$), such that $\cup_{i=1}^{k} g_i = V$ and $g_i \cap g_j = \emptyset$ for $i \neq j$, while taking into account some specific constraints and optimization objectives. We denote by $\Omega_k$ the search space of all possible grouping assignments with $k$ groups (candidate solutions) for a given problem instance.

In this work, we assume that the number of groups $k$ is fixed. For grouping problems where the task is to find a minimum number of groups, as the bin packing problem and graph coloring problem, we can successively solve the problem with decreasing values of $k$. Each time a solution is found for a given $k$, it gives an upper bound of the optimal number of groups. For instance, for the general graph coloring problems which aims to determine the chromatic number of a graph (i.e., the smallest number of colors $k$ for which a $k$-coloring exists), we typically solve a series of $k$-coloring problems with decreasing $k$ [15].

According to this definition of a grouping problem, a candidate solution in the search space $\Omega_k$ can be represented as a collection of groups: $S = \{g_1, \dots, g_k\}$. In this work, we use an equivalent matrix formulation of a candidate solution: for a fixed number of groups $k$, a candidate solution will be represented as a matrix $S = \{s_{ij}\}$ in $\{0,1\}^{n \times k}$, with $s_{ij} = 1$ and $s_{il} = 0$ for $l \neq j$, if the $i$-th item belongs to the group $j$. The $i$-th row of the candidate solution matrix defines the group assignment vector of the $i$-th item and is denoted by $s_i$.

We assume that the constraints and the optimization objective of the grouping problem can be formulated with a single evaluation function $f : \{0,1\}^{n \times k} \rightarrow \mathbb{R}$, where $f(S)$ is the evaluation (fitness) of the candidate solution $S$. The goal of the problem is to find a solution $S$ such that $f(S)$ is minimum.

## *2.1 Weight formulation of the problem*

Inspired by the work of [46], we express a (discrete) grouping (a candidate solution) as a continuous grouping by defining a weight matrix $W = \{w_{i,j}\}$ in $\mathbb{R}^{n \times k}$ composed of $n$ real numbers vectors of size $k$ [1]. This weight matrix corresponds to a learned and soft assignment of each item in each group. The weight matrix representation has the advantages of being more flexible and more informative than the binary group assignment. Indeed, during the search, for each item, the algorithm can learn to reject some group assignments by decreasing the associated weights, confirm some group assignments by increasing their weights, or even set almost equal weights for uncertain group assignments. The $i$-th row of the weight matrix defines the weight vector of the $i$-th item and is denoted by $w_i$.

**Item-to-group assignment** Given a weight matrix $W$, the associated solution (grouping) $S$ can be derived using the following procedure: for $i, j \in [\![1, n]\!] \times [\![1, k]\!]$, $s_{i,j} = 1$ if $j = \underset{l \in 1,\dots,k}{\operatorname{argmax}}\, w_{il}$ and $s_{ij} = 0$ if $j \neq \underset{l \in 1,\dots,k}{\operatorname{argmax}}\, w_{il}$.

[1] Note that the weight matrix can contain any real number while the probability matrix of [46] is composed of non-negative numbers.

Each item $v_i$ selects its group with the maximum weight in $w_i$. By abuse of notation and for simplicity, we rewrite this group selection for each item as a single function $S : \mathbb{R}^{n \times k} \to \{0,1\}^{n \times k}$:

$$S(W) = \text{one\_hot}(\text{argmax}(W)) \tag{1}$$

where *argmax* and *one_hot* operations are applied along the last axis of the matrix (group axis). Figure 1 shows an example with 4 items and 3 groups.

$$\overset{\text{W}}{\begin{bmatrix} -0.5 & 1 & 5 \\ 0.7 & -0.9 & 0.6 \\ 1 & -0.3 & 7 \\ 0.2 & 0.3 & -0.1 \end{bmatrix}} \xrightarrow{\text{argmax}} \begin{bmatrix} 3 \\ 1 \\ 3 \\ 2 \end{bmatrix} \xrightarrow{\text{one\_hot}} \overset{\text{S}}{\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}$$

Fig. 1. Example of group selection $S$ from a weight matrix $W$.

**Fitness evaluation** Using the current solution $S(W)$ we can compute the fitness function $f \circ S(W)$ of the problem. As $S$ derives from $W$, the given grouping problem becomes then the one of finding $W$ such that $f \circ S(W)$ is minimum. As we explain below, we will use first order gradient descent to solve this continuous optimization problem.

**Gradient evaluation** To tackle this real-valued optimization problem by first order gradient descent over $W$, we need to compute the gradient $\nabla_W f$ of the evaluation function $f$ with respect to $W$. This is a real matrix of size $n \times k$ whose element $(i,j)$ is $\frac{\partial f}{\partial w_{i,j}}$. By applying the chain rule [39], it gives for $i, j \in [\![1,n]\!] \times [\![1,k]\!]$:

$$\frac{\partial f}{\partial w_{i,j}} = \sum_{l=1}^{k} \frac{\partial s_{i,l}}{\partial w_{i,j}} \times \frac{\partial f}{\partial s_{i,l}} \tag{2}$$

### *2.2 Softmax approximation*

The function $S$ is differentiable almost everywhere with respect to $W$, but each term $\frac{\partial s_{i,l}}{\partial w_{i,j}}$ entering in equation (2) is always equal to zero. Therefore, we will use the *softmax* function as a continuous, differentiable approximation to *argmax*, with informative gradient.

$S$ can be approximated by a matrix $P$ of size $n \times k$, where each coefficient $p_{i,j}$ is computed with the elements of $W$ using the *softmax* function [6] as:

$$p_{i,j} = \frac{e^{\theta w_{i,j}}}{\sum_{l=1}^{k} e^{\theta w_{i,l}}}, \text{ for } i,j \in [\![1,n]\!] \times [\![1,k]\!] \tag{3}$$

Each coefficient $p_{i,j} \in [\![0,1]\!]$ of the matrix $P$ denotes the probability that the $i$-th item $v_i$ selects the $j$-th group $g_j$ as its group and $\theta$ is a control parameter in $\mathbb{R}^{+*}$. In the following we rewrite this probability evaluation for each item with a single matrix equation as:

$$P(W) = \text{softmax}(\theta W) \tag{4}$$

where the *softmax* function is applied along the last axis (group axis) for each item.

For any fixed $W$, as $\theta$ goes toward infinity, $P(W)$ converges toward $S(W)$. As an example for an item $v_i$, if the weight vector is $w_i = [-0.5, 0.5, 1, 1.2]$, the group selected for this item is $g_4$. Its corresponds to $s_i = [0, 0, 0, 1]$. Figure 2 shows the smooth approximation $p_i$ depending on the value of the parameter $\theta$ from 0.1 to 100. When the value of $\theta$ increases, the probability vector $p_i$ goes toward $s_i = [0, 0, 0, 1]$.

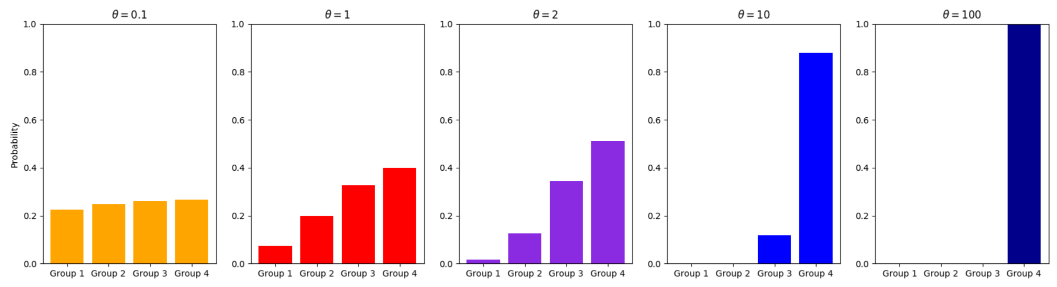

Fig. 2. Value of the probability vector $p_i$ for different values of the parameter $\theta$. When $\theta$ goes toward infinity $p_i$ goes toward $s_i$.

This kind of *softmax* approximation has notably been used to solve assignment problems in statistical physics frameworks [36,18], where at each iteration step of the optimization process the control parameter $\theta$ is increased in a deterministic annealing scheme, such that $p_{i,j}$ goes toward $s_{i,j}$. More recently this kind of relaxation with the *softmax* function has been employed to learn probabilistic sampling of categorical variables in stochastic computational graphs [21] or to apply $L_0$ parameters regularization for automatic neural network structures learning [29,22].

## *2.3 Gradient descent learning of the weights*

For $i, j, l$, we approximate $\frac{\partial s_{i,l}}{\partial w_{i,j}}$ by $\frac{\partial p_{i,l}}{\partial w_{i,j}} = \theta p_{i,l}(\delta_{j,l} - p_{i,j})$, with $\delta_{j,l}$ the Kronecker symbol equal to 1 if $j = l$ and 0 otherwise (details given in Appendix A). Using this Straight-Through (ST) gradient estimator [3] of $\frac{\partial s_{i,l}}{\partial w_{i,j}}$ in equation (2), we obtain for $i, j \in [\![1, n]\!] \times [\![1, k]\!]$:

$$\frac{\partial f}{\partial w_{i,j}} \approx \sum_{l=1}^{k} \frac{\partial p_{i,l}}{\partial w_{i,j}} \times \frac{\partial f}{\partial s_{i,l}} \tag{5}$$

$$= \sum_{l=1}^{k} \theta p_{i,l}(\delta_{j,l} - p_{i,j}) \times \frac{\partial f}{\partial s_{i,l}} \tag{6}$$

$$= \theta p_{i,j} \frac{\partial f}{\partial s_{i,j}} - \theta p_{i,j} \sum_{l=1}^{k} p_{i,l} \times \frac{\partial f}{\partial s_{i,l}} \tag{7}$$

In our framework however, we do not use any sequential loop to compute the gradient of each parameter. Instead we simply compute a tensor product to get the full gradient matrix. This operation can easily be parallellized on GPU devices. With tensor operations, the $n \times k$ equations (7) for $i, j \in [\![1, n]\!] \times [\![1, k]\!]$ become then a single equation:

$$\nabla_W f = \theta P \odot (\nabla_S f - (P \odot \nabla_S f) \cdot J) \tag{8}$$

where $\nabla_S f = \{\frac{\partial f}{\partial s_{i,j}}\}_{ij}$ is the gradient matrix of size $n \times k$ of $f$ with respect to $S$, $J$ is a matrix of 1's of size $k \times k$, $\odot$ is the Hadamard product (element-wise product) and $\cdot$ is the dot product between two matrices. This gradient $\nabla_S f$ depends on the grouping assignment problem. In general as we will see later with practical implementations (Section 3), this gradient is well defined and can easily be derived with tensor calculation.

Once the gradient with respect to the matrix of parameters $W$ is computed according to equation (8), we use first order gradient descent to update the parameters at each iteration:

$$W \leftarrow W - \eta \nabla_W f \tag{9}$$

where $\eta > 0$ is a fix learning rate. This kind of first order optimization is classically employed to learn neural networks (in its stochastic version) and is known to be very efficient to learn models with huge number of parameters. Alternative optimization gradient updates could be applied such as second order gradient descent with Hessian matrix, adaptive learning rate or momentum [24]. However for the reason of simplicity, we employ a simple and fast first order gradient descent procedure in this work.

### *2.4 Weight smoothing*

Inspired by the Reinforcement Learning Search (RLS) framework [46], we apply a weight smoothing procedure in order to give the possibility for the method to forget old decisions made long ago and no longer helpful: every $nb_{iter}$ iterations, each parameter of the weight matrix is divided by a constant number $\rho \geq 1$ which can be done with a single tensor computation according to:

$$W \leftarrow W/\rho \tag{10}$$

One notices that if $\rho$ goes toward infinity $W$ goes toward zero and the probability matrix $P$ goes toward a matrix whose elements are all equal to $\frac{1}{k}$ (corresponding to forget everything).

### *2.5 General GpGrad framework*

At initialization, each weight $w_{i,j}$ of the matrix $W$ is sampled according to normal distribution with zero mean and $\sigma_0$ standard deviation. Then, at each iteration, the proposed GpGrad framework involves four steps summarized in Algorithm 1: group selection, fitness evaluation, weights update with gradient descent and weight smoothing.

We highlight in blue the steps of the framework that are problem dependent, i.e, the fitness and gradient evaluations. All other steps could be used as it stands for different grouping problems.

The only stochastic part in GpGrad concerns the initialization of the tensor $W = W_0$ as the weights are drawn randomly according to a normal law. This random initialization has an impact on the search. Therefore for practical performance evaluation, a GpGrad algorithm will be run multiple times with different random seeds $r$ to initialize the pseudo-random generator of the normal law.

**Algorithm 1** GpGrad framework

```
Input: problem instance, number of available groups $k$ and random seed $r$;

Output: a legal grouping if it is found;
$W \leftarrow W_0$ /* Initialize the parameter matrix: $w^0_{i,j} \sim \mathcal{N}(0, \sigma_0)$ with random seed $r$*/
while stopping condition not reached do
    1) $S = \text{one\_hot}(\text{argmax}(W))$;                    /* Group selection*/
    2) Compute the fitness $f(S)$;    /* Problem specific fitness evaluation*/
    3) Weights update
        i) Compute $\nabla_S f$         /* Problem specific gradient evaluation */
        ii) $P = \text{softmax}(\theta W)$;                  /* softmax approximation*/
        iii) $\nabla_W f = \theta P \odot (\nabla_S f - (P \odot \nabla_S f) \cdot J)$   /* Gradient w.r.t $W$ */
        iv) $W \leftarrow W - \eta \nabla_W f$;              /*First order gradient descent*/
    4) Every $nb_{iter}$ iterations do $W \leftarrow W/\rho$;          /*Weight smoothing*/
end while
```

## 3 TensCol: an application of the GpGrad framework for graph coloring

In this section we present TensCol, an implementation of the GpGrad framework for the Graph Coloring Problem, which is a well-known grouping problem. Specifically, we consider the graph $k$-coloring problem ($k$ is given) which aims to find a legal $k$-coloring for a given graph $G = (V, E)$, that is, a partition of the nodes (i.e., the items) of $V$ into $k$ color groups such that any pair of nodes in any color group are not linked by an edge in $E$ (such a color group is also called an independent set). In other words, a legal $k$-coloring implies that two nodes linked by an edge necessarily receive two different colors and thus belong to different color groups.

For the purpose of optimization, we define an evaluation (fitness) function that minimizes the number of color conflicts. Formally, for a given graph $\mathcal{G} = (V, E)$ with $k$ available colors, a candidate solution is a partition of vertices into $k$ color groups $S = \{g_1, ..., g_k\}$ (in other words, $g_i$ is the group of vertices receiving color $i$). The evaluation function $f(S)$ is used to count the conflicting edges induced by $S$:

$$f(S) = \sum_{\{u,v\} \in E} \delta(u, v) \tag{11}$$

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

The GCP has the following property concerning symmetric solutions. Let $\pi : \{1, 2, ..., k\} \to \{1, 2, ..., k\}$ be an arbitrary permutation of the set of colors and let $S = \{g_1, ..., g_k\}$ be a k-coloring. Then $S^\pi = \{g_{\pi(1)}, ..., g_{\pi(k)}\}$ is also a $k$-coloring which is strictly equivalent to $S$. In other words, $S$ and $S^\pi$ 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. Symmetric solutions are known to be a source of difficulties for many search algorithms applied on the GCP [37]. In this work, we show how this problem can be easily avoided within the proposed GpGrad approach.

## *3.1 Matrix formulation of the graph coloring problem*

For a graph $G = (V, E)$ with $n$ nodes, we note $A = \{a_{i,j}\}_{i,j=1..n}$ its adjacency symmetric matrix. There is a link between two nodes $v_i$ and $v_j$ in $G$ if and only if $a_{i,j} = 1$. For this problem $S = \{s_{i,j}\}$ in $\{0, 1\}^{n\times k}$ is the matrix corresponding to the color assignment (or grouping assignment) of each node $v_i$ to a color (or group $g_j$). For $i \in [\![1, n]\!]$, $\sum_{j=1}^{k} s_{i,j} = 1$ as one and only one color has to be assigned to each node.

If we compute $V = S \cdot S^T$, we obtain a matrix in $\{0, 1\}^{n\times n}$ with coefficients:

$$v_{i,j} = \sum_{l=1}^{k} s_{i,l} s_{j,l} \tag{12}$$

One notices that $v_{i,j} = 1$ if and only if the two nodes $v_i$ and $v_j$ have the same color (or belong to the same group). Interestingly enough this *association matrix* is independent by permutation of the $k$ colors (corresponding to permutation of the columns in $S$).

We also introduce the *wrong grouping assignment matrix* or *conflict matrix* $C = A \odot V$ in $\{0, 1\}^{n\times n}$, where $\odot$ refers to element-wise product (Hadamard product). $C$ has coefficients $c_{i,j} = a_{i,j} v_{i,j}$ and such that $c_{i,j} = 1$ if and only if the two nodes $v_i$ and $v_j$ are in conflict, meaning that they are in the same group ($v_{i,j} = 1$) and there exists an edge in $G$ between the two nodes ($a_{i,j} = 1$). The total number of conflicts or *score* for a candidate solution $S$ is then:

$$f(S) = \frac{1}{2} \sum_{i,j=1}^{n} c_{i,j} \tag{13}$$

For simplicity we rewrite this equation as:

$$f(S) = \frac{1}{2} sum(C) \tag{14}$$

$$= \frac{1}{2} sum(A \odot (S \cdot S^T)) \tag{15}$$

with $sum(\cdot)$ corresponding to the summation operation of all matrix elements.

## *3.2 Gradient evaluation*

Using the matrix formulation of $f(S)$, we can now compute the gradient $\nabla_S f$ of the GCP problem that we will use in the GpGrad framework of Algorithm 1 in order to learn the group assignments.

As $f(S) = \frac{1}{2} \sum_{i,j=1}^{n} a_{i,j} \sum_{l=1}^{k} s_{i,l} s_{j,l}$, for $i, j \in [\![1, n]\!] \times [\![1, k]\!]$, the partial derivative of $f(S)$ with respect to an element $s_{i,j}$ of $S$ is (details given in Appendix B):

$$\frac{\partial f(S)}{\partial s_{i,j}} = \sum_{l=1}^{n} a_{i,l} s_{l,j} \tag{16}$$

The interpretation of equation (16) is obvious as when $s_{i,j}$ goes from the value 0 to 1, we expect the cost $f(S)$ to increase by the value $\frac{\partial f(S)}{\partial s_{i,j}}$, and this term corresponds effectively to $\sum_{l=1}^{n} a_{i,l} s_{l,j}$, the number of adjacent nodes to $v_i$ having the color $j$ and thus equal to the number of new conflicts created when the color (or group) $j$ is assigned to the node $v_i$.

Using matrix computation, equation (16) for $i, j \in [\![1, n]\!] \times [\![1, k]\!]$ becomes then:

$$\nabla_S f = A \cdot S \tag{17}$$

where the gradient $\nabla_S f$ is a real matrix of size $n \times k$ whose elements $(i, j)$ are equal to $\frac{\partial f(S)}{\partial s_{i,j}} = \sum_{l=1}^{n} a_{i,l} s_{l,j}$ [2].

[2] Notice that the term $A \cdot S$ corresponds to the $\gamma$-matrix used in an efficient implementation proposed in [13] of the popular TabuCol algorithm for the GCP [20].

## *3.3 Algorithm TensCol$_0$*

By incorporating equations (15) and (17) in the GpGrad framework of Algorithm 1 we obtain a first algorithm called TensCol$_0$ whose pseudo-code is displayed in Algorithm 2. The algorithm runs until a stopping condition is reached: when $f(S) = 0$ (i.e., a legal $k$-coloring is found) or the number of iterations is greater than $maxIter$ (a parameter).

**Algorithm 2** TensCol$_0$ implementation of the GpGrad framework for the GCP problem

```
**Input:** graph $G$ with adjacency matrix $A$, available colors $\{1, 2 \ldots, k\}$, a random seed $r$, and number of maximum allowed iterations $maxIter$;
**Output:** a legal $k$-coloring of $G$ if it is found;
$W \leftarrow W_0$  /* Initialize the parameter matrix: $w^0_{i,j} \sim \mathcal{N}(0, \sigma_0)$ with random seed $r$ */
$t \leftarrow 0$
**repeat**
    1) $S = \text{one\_hot}(\text{argmax}(W))$;                      /* Group selection*/
    2) Compute the fitness $f(S) = \frac{1}{2} sum(A \odot (S \cdot S^T))$;   /* GCP fitness*/
    3) Weights update
        i) Compute $\nabla_S f = A \cdot S$                 /* GCP gradient w.r.t $S$ */
        ii) $P = \text{softmax}(\theta W)$;                 /* softmax approximation*/
        iii) $\nabla_W f = \theta P \odot (\nabla_S f - (P \odot \nabla_S f) \cdot J)$   /* Gradient w.r.t $W$ */
        iv) $W \leftarrow W - \eta \nabla_W f$;            /*First order gradient descent*/
    4) Every $nb_{iter}$ iterations do $W \leftarrow W/\rho$;        /*Weight smoothing*/
    $t \leftarrow t + 1$
**until** $f(S) = 0$ or $t > maxIter$
**if** $f(S) = 0$ **then**
    **return** $S$
**end if**
```

One can notice that contrary to local search based coloring algorithms where only one node changes its color at each iteration, TensCol$_0$ may change the colors of many nodes at each iteration. We will highlight this property in Section 5 on a toy example. It proves to be quite useful to deal with large graphs.

However, this single solution update with TensCol$_0$ can easily get trapped in local optima and does not enable a large exploration of the search space for difficult instances. Therefore, we propose an extension of this algorithm called TensCol which runs multiple TensCol$_0$ on a CUDA parallel computer to significantly increase search diversification and reinforce search intensification as well.

## 3.4 TensCol, a population-based implementation of TensCol$_0$ for the GCP

To improve the search capacity of the TensCol$_0$ algorithm presented in the last section, we introduce below a population-based extension of the algorithm.

### 3.4.1 Population-based implementation with multiple parallel candidate solutions

The first idea is to simultaneously minimize a set of $D$ candidate solutions with CUDA parallel computation. Each solution corresponds to a candidate $k$-coloring that we note $S^d$ with $d \in [\![1, D]\!]$. In this work we will set $D = 200$ for the graph coloring problem. We use the same framework GpGrad, but instead of matrix, we will use three dimensional tensors, to take advantage of CUDA computation on GPU hardware.

We use the same notation as in Section 2 except that we write the different variables in bold in order to indicate that we use three dimensional tensors instead of matrices. Each of these three dimensional tensors has a first axis of size $D$, the other axis are of size $n$ and $k$.

The tensor of weights becomes then $\mathbf{W} = \{w_{d,i,j}\}$ in $\mathbb{R}^{D\times n\times k}$. The probability tensor $\mathbf{P}$ for the $D$ solutions can be computed in parallel from $\mathbf{W}$ with $\mathbf{P} = \text{softmax}(\theta\mathbf{W}))$, where softmax denotes softmax tensor computation along the last axis of the three dimensional tensor $\mathbf{W}$. The global tensor of these $D$ candidate solutions is defined by $\mathbf{S} = \{s_{d,i,j}\}$ in $\{0,1\}^{D\times n\times k}$ as $\mathbf{S} = \text{one\_hot}(\text{argmax}(\mathbf{W}))$, where the argmax and one_hot operators are applied along the last axis (group axis).

We note $\mathbf{S}'$ the transposed three dimensional tensor of solutions obtained from $\mathbf{S}$ by swapping its second and third axis. For the $D$ candidate solutions, the *association tensor* is then $\mathbf{V} = \mathbf{S}\cdot\mathbf{S}'$, where $\cdot$ is the dot product between two tensors. This corresponds to the sum product over the last axis of $\mathbf{S}$ and the second-to-last axis of $\mathbf{S}'$. For $d, i, j \in [\![1, D]\!] \times [\![1, n]\!] \times [\![1, k]\!]$, an element $v_{d,i,j}$ of $\mathbf{V}$ is given by:

$$v_{d,i,j} = \sum_{l=1}^{k} s_{d,i,l}s'_{d,l,j} \tag{18}$$

$$= \sum_{l=1}^{k} s_{d,i,l}s_{d,j,l} \tag{19}$$

We note $\mathbf{A}$, the three dimensional tensor of size $D \times n \times k$ such that for

$d \in [\![1, D]\!]$, each of its element is $\mathbf{a}_{d,i,j} = a_{i,j}$. It corresponds to $D$ duplication of the same matrix $A$.

By performing the element-wise product between the tensors $\mathbf{A}$ and $\mathbf{V}$, we obtain the global *conflict tensor* for the $D$ solutions as $\mathbf{C} = \mathbf{A} \odot \mathbf{V}$, where each element $c_{d,i,j}$ of $\mathbf{C}$ is given by:

$$\begin{aligned} c_{d,i,j} &= a_{d,i,j} v_{d,i,j} && (20) \\ &= a_{i,j} \sum_{l=1}^{k} s_{d,i,l} s_{d,j,l} && (21) \end{aligned}$$

The vector of fitness for the $D$ solutions can be computed with a single tensor operation from $\mathbf{C}$ as $(f(S^1), ..., f(S^D)) = \frac{1}{2} sum(\mathbf{C}, (2,3))$ with $sum(\cdot, (2,3))$ corresponding to the summation operation along second and third axis. For $d \in [\![1, d]\!]$:

$$\begin{aligned} f(S^d) &= \frac{1}{2} \sum_{i,j=1}^{n} c_{d,i,j} && (22) \\ &= \frac{1}{2} \sum_{i,j=1}^{n} a_{i,j} \sum_{l=1}^{k} s_{d,i,l} s_{d,j,l} && (23) \end{aligned}$$

### *3.4.2 Penalization for diversity and bonus for intensification*

Minimizing independently $D$ candidate solutions in parallel accelerates the algorithm, but each of the candidate solution can still easily get trapped in a poor local optimum. The idea that we employ in the following is to introduce two dependency terms linking the different solutions for two purposes: to avoid that they repeat the same mistakes by creating conflicts for the same pairs of nodes (diversity of errors), and to encourage them to make the same correct group assignments for each pair of nodes not connected by any link (intensification of good parts of the solution).

**3.4.2.1 Group concentration matrix** To do this, we first perform a summation of the three dimensional association tensor $\mathbf{V}$ along the first axis. We obtain the *group concentration matrix* of size $n \times n$:

$$\bar{V} = sum(\mathbf{V}, 1) \qquad (24)$$

with each element $\bar{v}_{i,j}$ of $\bar{V}$ given by:

$$\bar{v}_{i,j} = \sum_{d=1}^{D} v_{d,i,j} \tag{25}$$

$\bar{v}_{i,j}$ is equal to the number of candidate solutions assigning the nodes $v_i$ and $v_j$ in the same group. It corresponds to a measure of similarity between the $D$ candidate solutions. The construction of this *group concentration matrix* is independent by permutation of the $k$ colors (or $k$ groups) of each candidate solution $S^d$.

**3.4.2.2 Diversity penalization term for identical and incorrect group assignments** Using this *group concentration matrix* $\bar{V}$, we compute the diversity penalization term $\kappa(\mathbf{S}) = \text{sum}(A \odot \bar{V}^{\circ\alpha})$, where $\text{sum}(\cdot)$ corresponds to a sum of all matrix elements and $^\circ$ designates element-wise power tensor calculation:

$$\kappa(\mathbf{S}) = \sum_{i,j=1}^{n} a_{i,j}\bar{v}_{i,j}^{\alpha} \tag{26}$$

$$= \sum_{i,j=1}^{n} a_{i,j}(\sum_{d=1}^{D} v_{d,i,j})^{\alpha} \tag{27}$$

where $\alpha$ is a parameter greater than 1 in order to penalize similar conflicts of different candidate solutions. By minimizing this term, the solutions are discouraged to make the same mistakes for the same pair of connected nodes.

**3.4.2.3 Bonus term for identical and correct group assignments** Then using the same idea we compute the bonus term $\varpi(\mathbf{S}) = \text{sum}(\bar{A} \odot \bar{V}^{\circ\beta})$, where $\bar{A}$ is the adjacency matrix of the complement graph of $G$:

$$\varpi(\mathbf{S}) = \sum_{i,j=1}^{n} (1 - a_{i,j})\bar{v}_{i,j}^{\beta} \tag{28}$$

$$= \sum_{i,j=1}^{n} (1 - a_{i,j})(\sum_{d=1}^{D} v_{d,i,j})^{\beta} \tag{29}$$

with $\beta$ greater than 1. By maximizing this bonus term the $D$ candidate solutions are encouraged to make the same grouping assignments for any non-connected pair of nodes.

**3.4.2.4 Global loss objective** From now, we denote by $t \in [\![0, maxIter]\!]$ the iteration step of the algorithm. Indeed, we empirically noticed that the introduction of the bonus and diversity penalization terms with factors depending linearly on the iteration step $t$ during the optimization process improved the results. The intuitive idea is that by modifying step by step the fitness landscape, the algorithm will get out of local minimum traps.

The global loss objective of the model consists in a weighted aggregation between the following three criteria: (i) the sum of the fitness of the $D$ candidate solutions (equation (23)), (ii) the diversity penalization term (equation (27)) and (iii) the bonus term (equation (29)), subtracted in order to maximize it:

$$\mathcal{L}(\mathbf{S})(t) = \sum_{d=1}^{D} f(S^d) + \lambda t \kappa(\mathbf{S}) - \mu t \varpi(\mathbf{S}) \tag{30}$$

with $\lambda > 0$ and $\mu > 0$ two weighting parameters.

**3.4.2.5 Gradient of the loss** At iteration $t$, the gradient of this loss with respect to the tensor $\mathbf{S}$ of $D$ solutions is:

$$\nabla_{\mathbf{S}}\mathcal{L}(t) = A \cdot \mathbf{S} + 2\alpha\lambda t(A \odot \bar{V}^{\circ(\alpha-1)}) \cdot \mathbf{S} - 2\beta\mu t(\bar{A} \odot \bar{V}^{\circ(\beta-1)}) \cdot \mathbf{S} \tag{31}$$

Details are given in Appendix C.

### *3.4.3 TensCol algorithm*

By incorporating equations (30) and (31) in the GpGrad framework of Algorithm 1, we obtain the TensCol algorithm whose pseudo-code is displayed in Algorithm 3.

One can notice from Algorithm 3 that we do not need to evaluate the penalization term $\kappa(\mathbf{S})$ and bonus terms $\varpi(\mathbf{S})$ at each iteration. It saves computational and memory resources to compute only their gradients with respect to $\mathbf{S}$. The algorithm runs until a stopping condition is reached: when the fitness of one of the $D$ candidate solutions reaches 0 (i.e., a legal $k$-coloring is found) or when the number of iterations becomes greater than $maxIter$.

**Algorithm 3** TensCol for the GCP

```
**Input:** graph $G$ with adjacency matrix $A$, available colors $\{1, 2 \ldots, k\}$, random seed $r$, and number of maximum allowed iterations $maxIter$;
**Output:** a legal $k$-coloring $G$ if it is found;
$\mathbf{W} \leftarrow \mathbf{W_0}$    /* Initialize the parameter matrix: $w^0_{i,j,d} \sim \mathcal{N}(0, \sigma_0)$ with random seed $r$ */
$t \leftarrow 0$
**repeat**
  1) $\mathbf{S} = \text{one\_hot}(\text{argmax}(\mathbf{W}))$
  2) Fitness evaluation (forward phase)
    i) $\mathbf{V} = \mathbf{S} \cdot \mathbf{S}'$
    ii) $\mathbf{C} = \mathbf{A} \odot \mathbf{V}$
    iii) Compute the fitness vector $(f(S^1), ..., f(S^D)) = \frac{1}{2}\text{sum}(\mathbf{C}, (2, 3))$
  3) Parameters update (backward phase)
    i) $\bar{V} = \text{sum}(\mathbf{V}, 1)$
    ii) Compute $\nabla_{\mathbf{S}}\mathcal{L}(t) = A \cdot \mathbf{S} + 2\alpha\lambda t(A \odot \bar{V}^{\circ(\alpha-1)}) \cdot \mathbf{S} - 2\beta\mu t(\bar{A} \odot \bar{V}^{\circ(\beta-1)}) \cdot \mathbf{S}$
    iii) $\mathbf{P} = \text{softmax}(\theta\mathbf{W})$;
    iv) $\nabla_{\mathbf{W}}\mathcal{L} = \theta\mathbf{P} \odot (\nabla_{\mathbf{S}}\mathcal{L} - (\mathbf{P} \odot \nabla_{\mathbf{S}}\mathcal{L}) \cdot J)$
    v) $\mathbf{W} \leftarrow \mathbf{W} - \eta\nabla_{\mathbf{W}}\mathcal{L}$;
  4) Every $nb_{iter}$ do $\mathbf{W} \leftarrow \mathbf{W}/\rho$;
  $t \leftarrow t + 1$
**until** $\underset{d}{min} f(S^d) = 0$ or $t > maxIter$
**if** $\underset{d}{min} f(S^d) = 0$ **then**
  **return** $S^{d^*}$ with $d^* = \underset{d}{argmin} f(S^d)$
**end if**
```

## 4 TensCol for equitable graph coloring

Now we present an application of the TensCol algorithm presented in the last section for the equitable graph coloring problem (ECP), a variant of the graph coloring problem (GCP). The equitable graph coloring problem involves finding a minimum $k$ for which an equitable legal $k$-coloring exists.

An equitable legal $k$-coloring of $G$ is a partition of the vertex set $V$ of graph $G = (V, E)$ into $k$ disjoint independent sets $\{V_1, V_2, ..., V_k\}$ satisfying the *equity constraint*, i.e., $|\#V_i - \#V_j| \leq 1, i \neq j, 1 \leq i, j \leq k$, where $\#V_i$ denotes the cardinality of the group $V_i$. In other words, in an equitable legal $k$-coloring, the color groups have the same or similar group sizes (limited to a size difference of at most one).

In previous heuristic algorithms proposed for the ECP, the search is often made by alternating feasible and infeasible searches respectively in the space of equitable and non-equitable colorings, using very specific operators such as two or three swap operators preserving the equity constraint [41,43] (see Section 7).

With the GpGrad framework we show that we can naturally handle this problem by only modifying the gradient formulation given by equation (31) to take into account this equitable constraint.

## *4.1 TensCol with equitable coloring constraint*

For a $k$-coloring to be equitable the number of nodes assigned to each color group must be equal to $c_1 = \lfloor \frac{n}{k} \rfloor$ or $c_2 = \lfloor \frac{n}{k} \rfloor + 1$ where $\lfloor \cdot \rfloor$ denotes the integer part of a positive real number and $n$ is the number of vertices of $G$, with a particular case of $c_1 = c_2$ when $n$ is divisible by $k$.

We define the *equity fitness* $e(S^d)$ of a candidate solution $S^d$ as:

$$e(S^d) = \sum_{l=1}^{k} min(|\sum_{i=1}^{n} s_{d,i,l} - c_1|, |\sum_{i=1}^{n} s_{d,i,l} - c_2|) \tag{32}$$

It corresponds to the total number of surplus or deficit in term of number of nodes in the groups with respect to both admissible number of nodes $c_1$ and $c_2$ in each group. A legal solution $S^d$ of the ECP must satisfy $e(S^d) = 0$.

We denote by $e(\mathbf{S}) = \sum_{d=1}^{D} e(S^d)$ the global equity fitness for $D$ candidate solutions, with for $d \in [\![1, D]\!]$, $e(S^d)$ given by equation (32).

By convention for this problem we set $\frac{d|x|}{dx}_{x=0} = 0$. Therefore the partial derivative of $e(\mathbf{S})$ with respect to an element $s_{d,i,j}$ is:

$$\frac{\partial e(\mathbf{S})}{\partial s_{d,i,j}} = \begin{cases} 0 & \text{if } |\#V_{d,j} - c_1| = 0 \textit{ or } |\#V_{d,j} - c_2| = 0 \\ \frac{\#V_{d,j} - c_1}{|\#V_{d,j} - c_1|} & \text{if } |\#V_{d,j} - c_1| < |\#V_{d,j} - c_2| \\ \frac{\#V_{d,j} - c_2}{|\#V_{d,j} - c_2|} & \text{if } |\#V_{d,j} - c_2| < |\#V_{d,j} - c_1| \end{cases} \tag{33}$$

with $\#V_{d,j} = \sum_{i=1}^{n} s_{d,i,j}$ the total number of vertices with color $j$ for the candidate solution $d$.

Knowing that $c_1 \leq c_2$, equation (33) can be rewritten in a simpler form as:

$$\frac{\partial e(\mathbf{S})}{\partial s_{d,i,j}} = \begin{cases} 0 & \text{if } \#V_{d,j} = c_1 \textit{ or } \#V_{d,j} = c_2 \\ \\ +1 & \text{if } \#V_{d,j} > c_2 \\ \\ -1 & \text{if } \#V_{d,j} < c_1 \end{cases} \tag{34}$$

One can notice that this partial derivative term $\frac{\partial e(\mathbf{S})}{\partial s_{d,i,j}}$ does not depend on the index $i$ of the vertex $v_i$. Indeed the equitable constraint is a group constraint. Therefore the same gradient is applied to all nodes of the same group. According to equation (34) this gradient is equal to:

(1) 0 if the group already satisfies the constraint with $\#V_{d,j} = c_1$ or $\#V_{d,j} = c_2$ (no need to remove or add any nodes in this group)
(2) +1 if the group is composed of more items than required. Therefore, this gradient will put an equivalent pressure for all nodes to go to another group (if already in it) or not to join the group (if not already in it).
(3) -1 if this group is not composed of enough nodes, that will encourage the nodes to stay in this group or to join this group (if not already in it).

For the TensCol algorithm with the ECP constraint we employ the same GpGrad framework as presented in Algorithm 3 except that we add the ECP gradient of equation (34). This ECP constraint is introduced progressively with a weight increasing at each epoch by a constant value $\nu$. We also remove the bonus term $\varpi(\mathbf{S})$ as we empirically noticed that encouraging the different candidate solutions to do the same group assignments for the same pair of nodes degrades the quality of the algorithm for the ECP (this is because it is better that the different candidate solutions propose different grouping assignments for each given pair of nodes in order to explore a wider space of equitable colorings). However we keep the diversity penalization term $\kappa(\mathbf{S})$ with associated gradient $2\alpha\lambda t(A \odot \bar{V}^{\circ(\alpha-1)}) \cdot \mathbf{S}$. The TensCol algorithm for the ECP is displayed in Algorithm 4.

## 5 Toy example of grouping problems learned with TensCol$_0$

In this section, we present illustrative toy examples for the GCP and ECP to give intuitive insights on the gradient descent learning.

**Algorithm 4** TensCol for the ECP

```
**Input:** graph $G$ with adjacency matrix $A$, available colors $\{1, 2 \ldots, k\}$, random seed $r$, and number of maximum allowed iterations $maxIter$;
**Output:** a legal equitable $k$-coloring of $G$ if it is found;
$W \leftarrow W_0$    /* Initialize the parameter matrix: $w^0_{i,j,d} \sim \mathcal{N}(0, \sigma_0)$ with random seed $r$ */
$t \leftarrow 0$
**repeat**
  1) $\mathbf{S} = \text{one\_hot}(\text{argmax}(\mathbf{W}))$
  2) Fitness evaluation (forward phase)
    i) $\mathbf{V} = \mathbf{S} \cdot \mathbf{S}'$
    ii) $\mathbf{C} = \mathbf{A} \odot \mathbf{V}$
    iii) Compute the ECP fitness vector $(f(S^1) + e(S^1), ..., f(S^D) + e(S^D))$

  3) Parameters update (backward phase)
    i) $\bar{V} = \text{sum}(\mathbf{V}, 1)$
    ii) Compute $\nabla_{\mathbf{S}}\mathcal{L}(t) = A \cdot \mathbf{S} + \nu t \nabla_{\mathbf{S}} e(\mathbf{S}) + 2\alpha\lambda t(A \odot \bar{V}^{\circ(\alpha-1)}) \cdot \mathbf{S}$
    iii) $\mathbf{P} = \text{softmax}(\theta \mathbf{W})$;
    iv) $\nabla_{\mathbf{W}}\mathcal{L} = \theta \mathbf{P} \odot (\nabla_{\mathbf{S}}\mathcal{L} - (\mathbf{P} \odot \nabla_{\mathbf{S}}\mathcal{L}) \cdot J)$
    v) $\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}}\mathcal{L}$;
  4) Every $nb_{iter}$ do $\mathbf{W} \leftarrow \mathbf{W}/\rho$;
  $t \leftarrow t + 1$
**until** $\underset{d}{min}(f(S^d) + e(S^d)) = 0$ or $t > maxIter$
**if** $\underset{d}{min}(f(S^d) + e(S^d)) = 0$ **then**
  **return** $S^{d^*}$ with $d^* = \underset{d}{argmin}(f(S^d) + e(S^d))$
**end if**
```

## *5.1 GCP toy example*

In this section we use the simple TensCol$_0$ implementation of the GpGrad framework for the GCP corresponding to Algorithm 2, where only one candidate solution is evaluated at each step. We apply it on an easy instance named myciel4.col of the COLOR02 benchmark with 21 nodes and known chromatic number equal to 5. Figure 3 displays the last 3 iterations (over 8) of TensCol$_0$ to color myciel4.col with 5 colors and random seed $r = 0$. The parameters used in TensCol$_0$ are $\eta = 0.001$, $nb_{iter} = 5$, $\rho = 100$ and $\theta = 1$. The number written on each node indicates the gradient of the global score with respect to the selected color for this node. It corresponds to the total number of conflicts, i.e., adjacent nodes with the same color (cf. equation (16)). Red edges correspond to conflicting edges. Blue circles highlight the nodes changing their color from one step to another. As we can see on this figure, TensCol$_0$ tends to change the color of the nodes with the highest gradient values. One can also notice that TensCol$_0$ can change the color of more than one node at each iteration as the update is done on the full candidate solution matrix $S$ at each step. A legal coloring solution $S^*$ is shown in Figure 3-(d) with $f(S^*) = 0$.

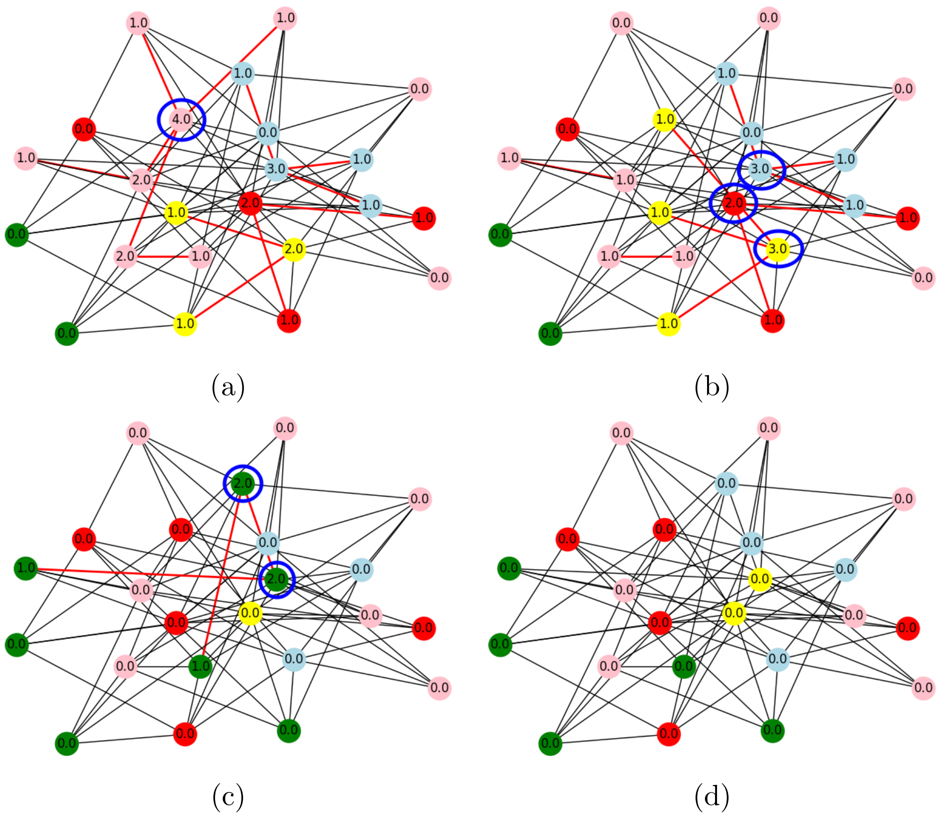

Fig. 3. **Last three steps of TensCol$_0$** to color the graph myciel4.col with 5 colors (optimal value). The number on each node indicates the gradient of the global score with respect to the selected color for this node. Red edges corresponds to conflicting edges. Blue circles highlight the nodes changing their color from one step to another. Several vertices can change their colors during the same iteration. Better seen in color.

## *5.2 ECP toy example*

The equitable coloring of this same graph (myciel4.col) is now considered with again $k = 5$. We use the same algorithm and the same parameters, but the equitable constraint is taken into account in the gradient evaluation: $\nabla_S f(S) = A \cdot S + \nabla_S e(S)$, with for $i, j \in [\![1, 21]\!] \times [\![1, 5]\!]$:

$$\frac{\partial e(S)}{\partial s_{i,j}} = \begin{cases} 0 & \text{if } \#V_j = 4 \textit{ or } \#V_j = 5 \\ \\ +1 & \text{if } \#V_j > 5 \\ \\ -1 & \text{if } \#V_j < 4 \end{cases} , \qquad (35)$$

where $\#V_j = \sum_{i=1}^{21} s_{i,j}$ is the total number of nodes with color $j$.

Figure 4 displays the last three steps (over 19) of $\text{TensCol}_0$ to find an equitable coloring for myciel4.col with 5 colors and random seed $r = 0$. The number on each node indicates the gradient of the global ECP score with respect to the selected color for this node. Therefore this number is equal for each node to the total number of conflicting edges related to this node (red edges) plus one if the group is in surplus and minus one if the group is in deficit. Blue circles highlight the nodes changing their color from one step to another. First we see that the algorithm alternates between legal coloring without any conflicting edges (Figure (a) and (c)) and illegal coloring with red conflicting edges (Figure (b)).

One can notice that many nodes (and even more than required) change from pink to red from Figure (a) to Figure (b) and the other way around from red to pink (and blue) from Figure (b) to Figure (c). This is due to the fact that the color updates are done independently for each node when using a first order gradient descent. It highlights that other update rules that can learn dependencies between nodes could be employed in order to *coordinate* the multiple node changes and avoid this kind of oscillation phenomenon. This is out of the scope of this work and constitutes an interesting research perspective.

## 6 Computational results on GCP and ECP benchmark instances

This section is dedicated to an assessment of the TensCol algorithm for solving the GCP and ECP. We first present the experimental setting of TensCol for both problems. Then, we show the computational results obtained on the GCP, followed by the results on the ECP.

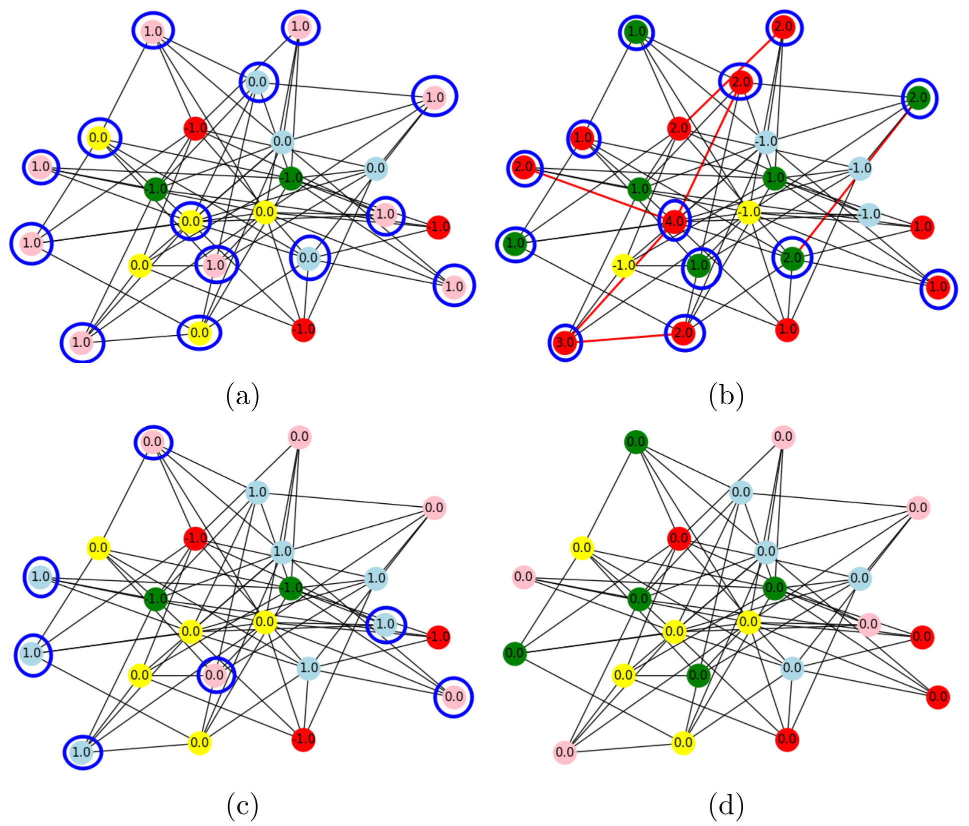

Fig. 4. **Last three steps of TensCol$_0$** to find an equitable coloring for myciel4.col with 5 colors (optimal value). The number on each node indicates the gradient of the total number of conflicts plus the gradient of the ECP constraint with respect to the selected color for this node. Red edges corresponds to conflicting edges. Blue circles highlight the nodes changing their color from one step to another. Better seen in color.

### *6.1 Benchmark instances and comparison criterion*

For the GCP we test our algorithm on the 36 most widely used benchmark instances from the second DIMACS competition [3] for assessing the performance of graph coloring algorithms in recent studies [45,35]. For the ECP we test our algorithm on the same set of 73 benchmark instances used in [41,43] from the second DIMACS and COLOR02 competitions [4] .

Following the common practice for reporting comparative results in the coloring literature, we focus on the best solution found by each algorithm corresponding to the smallest number $k$ of colors needed to color a graph.

It is worth mentioning that for the GCP, no single algorithm including the

[3] Publicly available at `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/`

[4] Publicly available at `https://mat.gsia.cmu.edu/COLOR02/`

most recent algorithms can attain the best-known results for all 36 tested DIMACS instances. Indeed, even the best performing algorithms miss at least two best-known results. This is understandable given that these instances have been studied for a long time (for some 30 years) and some best-known results have been achieved under specific and relaxed conditions (e.g., extended run time up to one month) and only by very few algorithms. Moreover, one notices that for these benchmark graphs, when $k$ is close to the chromatic number of the given graph $G$ or to the current best-known (upper) bound $k^*$, finding a legal $k$-coloring is a difficult task. As such, an algorithm able to attain the best-known results for a majority of the benchmark instances can be qualified competitive with respect to the current state-of-the-art on graph coloring.

These comments remain valid for the ECP. Meanwhile, given that the ECP has been studies less intensively compared to the GCP, one could still hope to find improved best-known results (i.e., equitable $k$-colorings with $k$ smaller than the current best bound $k^*$). As we show in Section 6.4, this is indeed possible with the proposed TensCol algorithm, which achieves 8 new record results.

## *6.2 Experimental setting*

The TensCol algorithm was implemented in Python 3.5 with Pytorch 1.1 library for tensor calculation with Cuda 10.0 [5] . It is specifically designed to run on GPU devices. In this work we used a Nvidia RTX 2080Ti graphic card with 12 GB memory. A Nvidia Tesla P100 with 16 GB memory was used only for the two very large graphs (C2000.5.col and C2000.9.col instances).

For a small graph such as r250.5.col colored with 65 colors for an equitable coloring, when we run TensCol with 200 candidate solutions in parallel there are $250*65*200 = 3.2$ millions weights in the model updated at each iteration. On a Nvidia RTX 2080Ti graphic card, it takes 0.002 seconds per iteration (for 200 candidate solutions evaluated). For a large graph such as C2000.9.col colored with 431 colors for an equitable coloring, there are $2000*431*200 = 172$ millions weights $w_{i,j,d}$ updated at each iteration. On a Nvidia Tesla P100, it takes 0.37 seconds per iteration.

For our experiments, each instance was solved 10 times independently (with 10 different random seeds) following the common practice in the literature for graph coloring.

We used a baseline set of parameters, given in Table 1, for the GCP and ECP, except for the smoothing parameter $\rho$ which is very sensitive to the

[5] The source code of our algorithm will be made publicly available.

different structures of the GCP and ECP instances. Indeed we noticed that $\rho$ is a key parameter depending on the fitness landscape of the coloring problem and thus we chose it among a set of 5 different values. For random graphs such as DSJC500.5.col or DSJC1000.5.col with rough fitness landscapes, we had to set $\rho$ to 100 or 200 in order to frequently reset the learned tensor of weights $\mathbf{W}$ according to equation (10). However, for geometric graphs such as R1000.1.col or instances based on register allocation for variables in real codes such as fpsol2.i.1.col or mulsol.i.1.col, we set $\rho = 1$ meaning that we do not apply any reset process during the optimization process.

Table 1
Parameter setting in TensCol for GCP and ECP

| Parameter | Description | Value |
|---|---|---|
| $D$ | number of parallel candidate solutions | 200 |
| $\sigma_0$ | standard deviation of initial parameters | 0.01 |
| $\theta$ | exponential base in softmax approximation | 1 |
| $\eta$ | learning rate | 0.001 |
| $nb_{iter}$ | number of iterations between two smoothing procedures | 5 |
| $\rho$ | smoothing parameter | {1,2,10,100,200} |
| $\alpha$ | exponent for diversity penalization | 2.5 |
| $\lambda$ | increase factor for diversity penalization | $10^{-5}$ |
| $\beta$ (only for GCP) | exponent for similarity bonus | 1.2 |
| $\mu$ (only for GCP) | increase factor for similarity bonus | $10^{-6}$ |
| $\nu$ (only for ECP) | increase factor for equitable constraint | $10^{-5}$ |

For some particular cases we had to deviate from this baseline set of parameters. For random graphs with high density ($\geq 0.9$), we set $\alpha = 1.5$ instead of $\alpha = 2.5$ in order to prevent the diversity penalization term $\kappa(S)$ from becoming too important (cf. equation (27)). On the contrary, for random graphs with low density ($\leq 0.1$), we set $\mu = 10^{-7}$ for the GCP instead of $\mu = 10^{-6}$ to avoid that the bonus term $\varpi(S)$ (cf. equation (29)) becomes too preponderant in the global loss evaluation. For families of instances with a huge number of nodes (more than 2000) such as wap02a.col and wap03a.col of the COLOR02 challenge, we had to decrease the number of candidate solutions evaluated in parallel from 200 to 20 in order to limit the GPU memory required. For the very difficult graph R1000.5.col on the GCP, we achieved a best coloring of 235 with the baseline set of parameters, while a better coloring with 234 colors can be obtained by TensCol with a specific fine tuning ($\rho = 10$, $\alpha = 1.5$, $\beta = 1.1$ and $nb_{iter} = 10$).

For all these instances, we used as the stopping criterion, a maximum allowed number of iterations of $2 \times 10^6$, which corresponds to a maximum of $4 \times 10^8$ different candidate solutions evaluated (as there are 200 solutions evaluated at each iteration).

## *6.3 Computational results on the GCP*

This section is dedicated to an extensive experimental evaluation of TensCol on the GCP. We show a comparison of TensCol with 5 state-of-the-art coloring algorithms of the literature:

- the probability learning based local search algorithm (PLSCOL) [45], which is an improved algorithm of reinforcement learning search (RLS) [46].
- the two-phase evolutionary algorithm (MMT) [31].
- the distributed evolutionary quantum annealing algorithm (QA) [42]
- the *memetic* algorithm (MA) [30]
- the recent implementation of the *memetic* approach with two elite solutions in the population (HEAD) [35]

A more detailed presentation of each reference algorithm is given in the Section 7.1.

The results of TensCol and the reference algorithms are reported in Table 2: column 2 indicates the number of vertices in each instance, column 3 shows the chromatic number number $\chi(G)$ (if known)[6]. Column 4 gives the best-known results (best upper bound of $\chi(G)$) ever reported by an algorithm. Columns 5 to 9 show the best results ($k_{best}$) of PLSCOL, MMT, MA, QA and HEAD respectively. The remaining columns reports the results obtained by our TensCol algorithm: the best result ($k_{best}$), the success runs over 10 runs during which TensCol attained its best result, and for indicative purpose the average computation time (in seconds) of TensCol for a successful run.

As we observe on Table 2, TensCol always finds the best-known coloring in a small amount of time on the instances with less than 200 nodes such as DSJC125.1.col or R125.1. On medium graphs with up to 500 nodes such as DSJC250.5.col or DSJC500.5.col, Tensol is very competitive compared to all the other algorithms.

For the large DSJC1000.5.col or DSJC1000.9.col instances, TensCol performs worse than MA and HEAD, two best-performing *memetic* algorithms that rely on recombining large independent sets between solutions, a technique which has proven to be very powerful for this kind of large random graphs. However using this mechanism can be a drawback for large geometric graphs as shown by the poor results obtained by MA and HEAD on the R1000.5.col instance. Indeed, as noticed in [44], for this R1000.5.col instance, some large independent sets are not part of any optimal 234-coloring and extracting such

[6] For some instances such as flat1000_76_0 the chromatic number is known by construction of the graph (and equal to 76 in this case) but no heuristic has yet found a legal coloring with this chromatic number.

independent sets cannot help to decrease the number of colors needed for the whole graph. TensCol however can recover the optimal coloring for all the family of geometric graphs and in particular for this difficult R1000.5.col instance with an optimal 234-coloring. It is notable that this optimal coloring was only previously found with the MMT algorithm [31], which is a complex two-phase method mixing an effective tabu search, a specialized grouping crossover operator, and a post-optimization phase based on a set covering formulation of the GCP.

TensCol also obtains good results for the large latin square graph (latin_square_10), which is difficult in particular for the MMT algorithm, but also for the MA and HEAD *memetic* algorithms.

Finally, it is worth noting that given the particularity of the GCP benchmark instances as discussed in Section 6.1, it is not suitable to apply statistical tests.

## *6.4 Computational results on the ECP*

This section reports a comparison of TensCol with 5 recent state-of-the-art algorithms in the literature for the equitable coloring problem:

- the tabu search algorithm (TabuEqCol) [9]
- the backtracking based iterated tabu search (BITS) [26]
- the feasible and infeasible search algorithm (FISA) [41]
- the hybrid tabu search (HTS) [43]
- the memetic algorithm MAECP [40].

A more detailed presentation of each of these algorithms for the ECP is given in Section 7.2.

Tables 4 and 5 show the best results of the six compared algorithms in terms of the smallest number of colors used to color each graph with the equity constraint for the DIMACS and COLOR02 instances. In column 3 is displayed the overall best-known result $k^*$ of the ECP reported in the literature. The next five columns report the best results by the reference algorithms (TabuEqCol, BITS, FISA, HTS and MAECP). The last three columns show the results obtained by our Tensol algorithm: the best solution found $k_{best}$, the success run over 10 runs, and only for indicative purpose, the average computation time in second for a successful run.

The results in Tables 4 and 5 indicate that TensCol obtains comparable results with the state-of-the-art algorithms for the graphs with up to 500 nodes, except for DSJC500.9.col, le450_15a.col, le450_15b.col, le450_25c.col and le450_25d.col where TensCol performs worse than the reference algorithms, but it is better

Table 2
Comparative results of TensCol with state-of-the-art algorithms on the DIMACS graphs for the GCP. Numbers in bold indicate that the best result $k_{\text{best}}$ found by the algorithm is equal to the overall best-known result in the literature $k^*$ or the chromatic number.

| | | | | PLSCOL | MMT | MA | QA | HEAD | TensCol | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | $\|V\|$ | $\chi(G)$ | $k^*$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | SR | t(s) |
| DSJC125.1 | 125 | ? | 5 | **5** | **5** | **5** | **5** | **5** | **5** | 10/10 | 40 |
| DSJC125.5 | 125 | ? | 17 | **17** | **17** | **17** | **17** | **17** | **17** | 10/10 | 68 |
| DSJC125.9 | 125 | ? | 44 | **44** | **44** | **44** | **44** | **44** | **44** | 10/10 | 22 |
| DSJC250.1 | 250 | ? | 8 | **8** | **8** | **8** | **8** | **8** | **8** | 10/10 | 95 |
| DSJC250.5 | 250 | ? | 28 | **28** | **28** | **28** | **28** | **28** | **28** | 10/10 | 199 |
| DSJC250.9 | 250 | ? | 72 | **72** | **72** | **72** | **72** | **72** | **72** | 10/10 | 87 |
| DSJC500.1 | 500 | ? | 12 | **12** | **12** | **12** | **12** | **12** | **12** | 10/10 | 1098 |
| DSJC500.5 | 500 | ? | 47 | 48 | 48 | 48 | 48 | **47** | 48 | 5/10 | 7807 |
| DSJC500.9 | 500 | ? | 126 | **126** | 127 | **126** | **126** | **126** | **126** | 6/10 | 18433 |
| DSJC1000.1 | 1000 | ? | 20 | **20** | **20** | **20** | **20** | **20** | **20** | 10/10 | 10225 |
| DSJC1000.5 | 1000 | ? | 82 | 87 | 84 | 83 | 83 | **82** | 84 | 9/10 | 32495 |
| DSJC1000.9 | 1000 | ? | 222 | 223 | 225 | 223 | **222** | **222** | 224 | 6/10 | 58084 |
| DSJR500.1 | 500 | ? | 12 | **12** | **12** | **12** | **12** | * | **12** | 10/10 | 7 |
| DSJR500.1c | 500 | ? | 85 | **85** | **85** | **85** | **85** | **85** | **85** | 10/10 | 298 |
| DSJR500.5 | 500 | ? | 122 | **122** | **122** | **122** | **122** | * | **122** | 10/10 | 4310 |
| flat300_26_0 | 300 | 26 | 26 | **26** | **26** | **26** | * | * | **26** | 10/10 | 176 |
| flat300_28_0 | 300 | 28 | 28 | 30 | 31 | 29 | 31 | 31 | 31 | 10/10 | 586 |
| flat1000_76_0 | 1000 | 76 | 81 | 86 | 83 | 82 | 82 | **81** | 83 | 3/10 | 34349 |
| latin_square_10 | 900 | ? | 97 | 99 | 101 | 99 | **97** | * | 98 | 10/10 | 28925 |
| le450_15a | 450 | 15 | 15 | **15** | **15** | **15** | **15** | **15** | **15** | 10/10 | 333 |
| le450_15b | 450 | 15 | 15 | **15** | **15** | **15** | **15** | **15** | **15** | 10/10 | 333 |
| le450_15c | 450 | 15 | 15 | **15** | **15** | **15** | **15** | **15** | **15** | 10/10 | 507 |
| le450_15d | 450 | 15 | 15 | **15** | **15** | **15** | **15** | **15** | **15** | 10/10 | 301 |
| le450_25a | 450 | 25 | 25 | **25** | **25** | **25** | **25** | **25** | **25** | 10/10 | 87 |
| le450_25b | 450 | 25 | 25 | **25** | **25** | **25** | **25** | **25** | **25** | 10/10 | 12 |
| le450_25c | 450 | 25 | 25 | **25** | **25** | **25** | **25** | **25** | **25** | 10/10 | 19680 |
| le450_25d | 450 | 25 | 25 | **25** | **25** | **25** | **25** | **25** | **25** | 10/10 | 9549 |
| R125.1 | 125 | ? | 5 | **5** | **5** | **5** | **5** | * | **5** | 10/10 | 0.03 |
| R125.5 | 125 | ? | 36 | **36** | **36** | **36** | **36** | * | **36** | 10/10 | 6.2 |
| R250.1 | 250 | ? | 8 | **8** | **8** | **8** | **8** | * | **8** | 10/10 | 0.04 |
| R250.5 | 250 | ? | 65 | **65** | **65** | **65** | **65** | **65** | **65** | 10/10 | 33 |
| R1000.1 | 1000 | ? | 20 | **20** | **20** | **20** | **20** | * | **20** | 10/10 | 15 |
| R1000.1c | 1000 | ? | 98 | **98** | **98** | **98** | **98** | **98** | **98** | 10/10 | 4707 |
| R1000.5 | 1000 | 234 | 234 | 254 | **234** | 245 | 238 | 245 | **234** | 2/10 | 23692 |
| school1 | 385 | ? | 14 | **14** | **14** | **14** | **14** | **14** | **14** | 10/10 | 0.39 |
| school1_nsh | 352 | ? | 14 | **14** | **14** | **14** | **14** | **14** | **14** | 10/10 | 0.59 |

on the DSJC500.5.col instance.

For very large graphs (at least 2000 nodes) such as the instances of the wapXXa.col family, we had to limit the number of parallel solutions evaluated due to memory limitation on the gpu cards used for our experiments. As displayed on Table 5, the results obtained by TensCol are less good than the other algorithms for this family of graphs.

TensCol excels on large graphs between 900 and 2000 nodes. Remarkably, it found 8 new record solutions (summarized in Table 3) that improve on the best-known results published in the literature. As one can observe in Table 3, TensCol significantly improves the best-known upper bound for three large graphs with 37 colors less for C2000.9.col, 9 colors less for C2000.5.col and 8 colors less for R1000.5.col, while the improvement for the 6 other cases goes from -1 to -3 colors.

Table 3
New record coloring found by TensCol for 8 benchmark graphs. Some improvements are very important with 8, 9 and 37 colors less.

| Instance | $\|V\|$ | previous $k_{\text{best}}$ | new $k_{\text{best}}$ | Improvement |
|---|---|---|---|---|
| DSJR500.5.col | 500 | 124 | 122 | -2 |
| DSJC1000.5.col | 1000 | 95 | 92 | -3 |
| R1000.5.col | 1000 | 247 | 239 | -8 |
| flat1000_60.0.col | 1000 | 93 | 92 | -1 |
| flat1000_76.0.col | 1000 | 93 | 91 | -2 |
| latin_square_10.col | 900 | 103 | 102 | -1 |
| C2000.5.col | 2000 | 183 | 172 | -9 |
| C2000.9.col | 2000 | 468 | 431 | -37 |

## 7 Reference heuristics for the GCP and ECP

The GCP and ECP have been studied very intensively in the past. Both problems are known to be NP-hard in the general case [14,17]. Thus, assuming that $N \neq NP$, no algorithm can solve these problems in polynomial time except for specific instances. The best performing exact algorithms are generally not able to find an optimal solution for the GCP and ECP in a reasonable amount of time when the number of vertices is greater than 250 (in particular for random graphs with medium to high density of edges) [32,34].

Therefore, for large graphs one uses heuristics that explore partially the search space to find a proper $k$-coloring. Heuristics proceed in general by successively solving the $k$-coloring problem with decreasing values of $k$, without optimality guarantee [15].

A thorough review of existing heuristics for the GCP and ECP is out of the scope of this work. Instead, we provide below a brief description of the main

Table 4
Comparative results of TensCol with state-of-the-art algorithms on the 73 benchmark ECP instances (1/2). Numbers in bold indicate that the best result $k_{\text{best}}$ found by the algorithm is equal to the overall best-known result $k^*$ in the literature. A star in column $k_{\text{best}}$ for TensCol indicates that a new best coloring of the ECP has been found.

| | | | TabuEqCol | BITS | FISA | HTS | MAECP | TensCol | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | $\|V\|$ | $k^*$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | SR | t(s) |
| DSJC125.1 | 125 | 5 | **5** | **5** | **5** | **5** | **5** | **5** | 10/10 | 50 |
| DSJC125.5 | 125 | 9 | 17 | **17** | **17** | **17** | **17** | **17** | 10/10 | 124 |
| DSJC125.9 | 125 | 44 | 45 | **44** | **44** | **44** | **44** | **44** | 10/10 | 22 |
| DSJC250.1 | 250 | 8 | **8** | **8** | **8** | **8** | **8** | **8** | 10/10 | 97 |
| DSJC250.5 | 250 | 29 | 32 | **29** | **29** | **29** | **29** | **29** | 10/10 | 312 |
| DSJC250.9 | 250 | 72 | 83 | **72** | **72** | **72** | **72** | **72** | 9/10 | 1285 |
| DSJC500.1 | 500 | 13 | **13** | **13** | **13** | **13** | **13** | **13** | 10/10 | 168 |
| DSJC500.5 | 500 | 51 | 63 | 56 | 52 | 52 | **51** | **51** | 10/10 | 3793 |
| DSJC500.9 | 500 | 128 | 182 | 129 | 130 | 129 | **128** | 129 | 4/10 | 13537 |
| DSJR500.1 | 500 | 12 | **12** | **12** | **12** | **12** | **12** | **12** | 10/10 | 490 |
| DSJR500.5 | 500 | 124 | 133 | 126 | 126 | 125 | 124 | **122*** | 10/10 | 5021 |
| DSJC1000.1 | 1000 | 21 | 22 | **21** | **21** | **21** | **21** | **21** | 9/10 | 1757 |
| DSJC1000.5 | 1000 | 95 | 112 | 101 | 95 | 95 | 95 | **92*** | 10/10 | 12430 |
| DSJC1000.9 | 1000 | 251 | 329 | 252 | 252 | **251** | **251** | **251** | 10/10 | 20862 |
| R125.1 | 125 | 5 | - | **5** | **5** | **5** | **5** | **5** | 10/10 | 0.07 |
| R125.5 | 36 | 36 | - | **36** | **36** | **36** | **36** | **36** | 10/10 | 29 |
| R250.1 | 250 | 8 | - | **8** | **8** | **8** | **8** | **8** | 10/10 | 0.13 |
| R250.5 | 250 | 65 | - | 66 | 66 | **65** | **65** | **65** | 10/10 | 40 |
| R1000.1 | 1000 | 20 | - | **20** | **20** | **20** | **20** | **20** | 10/10 | 4426 |
| R1000.5 | 1000 | 247 | - | 250 | 250 | 249 | 247 | **239*** | 9/10 | 53187 |
| le450_5a | 450 | 5 | - | **5** | **5** | **5** | **5** | **5** | 10/10 | 15 |
| le450_5b | 450 | 5 | 7 | **5** | **5** | **5** | **5** | **5** | 10/10 | 22 |
| le450_5c | 450 | 5 | - | **5** | **5** | **5** | **5** | **5** | 10/10 | 4 |
| le450_5d | 450 | 5 | 8 | **5** | **5** | **5** | **5** | **5** | 10/10 | 3 |
| le450_15a | 450 | 15 | - | **15** | **15** | **15** | **15** | 18 | 10/10 | 4252 |
| le450_15b | 450 | 15 | **15** | **15** | **15** | **15** | **15** | 18 | 10/10 | 3346 |
| le450_15c | 450 | 15 | - | **15** | **15** | **15** | **15** | **15** | 10/10 | 120 |
| le450_15d | 450 | 15 | 16 | **15** | **15** | **15** | **15** | **15** | 10/10 | 178 |
| le450_25a | 450 | 25 | - | **25** | **25** | **25** | **25** | **25** | 10/10 | 6343 |
| le450_25b | 450 | 25 | **25** | **25** | **25** | **25** | **25** | **25** | 10/10 | 2411 |
| le450_25c | 450 | 26 | - | **26** | **26** | **26** | **26** | 31 | 2/10 | 46129 |
| le450_25d | 450 | 26 | 27 | **26** | **26** | **26** | **26** | 31 | 10/10 | 33419 |
| wap01a | 2368 | 42 | 46 | **42** | **42** | **42** | **42** | 46 | 2/10 | 1891 |
| wap02a | 2464 | 41 | 44 | **41** | **41** | **41** | **41** | 47 | 8/10 | 1657 |
| wap03a | 4730 | 44 | 50 | 45 | 45 | 45 | **44** | 51 | 2/10 | 5522 |
| wap04a | 5231 | 43 | - | 44 | 44 | 44 | **43** | 51 | 6/10 | 2376 |
| wap05a | 905 | 50 | - | **50** | **50** | **50** | **50** | **50** | 10/10 | 417 |

Table 5
Comparative results of TensCol with state-of-the-art algorithms on the 73 benchmark ECP instances (2/2). Numbers in bold indicate that the best result $k_{\text{best}}$ found by the algorithm is equal to the overall best-known result $k^*$ in the literature. A star in column $k_{\text{best}}$ for TensCol indicates that a new best coloring of the ECP has been found.

| | | | TabuEqCol | BITS | FISA | HTS | MAECP | TensCol | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | $\|V\|$ | $k^*$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | $k_{\text{best}}$ | SR | t(s) |
| wap06a | 947 | 41 | - | **41** | **41** | **41** | **41** | 45 | 8/10 | 413 |
| wap07a | 1809 | 42 | - | 43 | 43 | **42** | **42** | 49 | 10/10 | 995 |
| wap08a | 1870 | 42 | - | 43 | 43 | **42** | **42** | 47 | 2/10 | 1171 |
| flat300_28.0 | 300 | 32 | 36 | 34 | **32** | 33 | **32** | **32** | 10/10 | 1583 |
| flat1000_50.0 | 1000 | 92 | - | 101 | 94 | **92** | 93 | **92** | 7/10 | 5566 |
| flat1000_60.0 | 1000 | 93 | - | 101 | 94 | 94 | 93 | **92*** | 10/10 | 5905 |
| flat1000_76.0 | 1000 | 93 | 112 | 102 | 94 | 93 | 93 | **91*** | 3/10 | 53553 |
| latin_square_10 | 900 | 103 | 130 | 113 | 104 | 107 | **103** | **103** | 8/10 | 37233 |
| | | | | | | | | **102*** | 1/10 | 14484 |
| C2000.5 | 2000 | 183 | - | 201 | 183 | 188 | 183 | **172*** | 8/10 | 111884 |
| C2000.9 | 2000 | 468 | - | 502 | 493 | 501 | 468 | **431*** | 4/10 | 109243 |
| mulsol.i.1 | 197 | 49 | 50 | **49** | **49** | **49** | **49** | **49** | 10/10 | 301 |
| mulsol.i.2 | 188 | 36 | 48 | **36** | **36** | **36** | **36** | **36** | 3/10 | 2877 |
| fpsol2.i.1 | 496 | 65 | 78 | **65** | **65** | **65** | **65** | **65** | 10/10 | 3033 |
| fpsol2.i.2 | 451 | 47 | 60 | **47** | **47** | **47** | **47** | **47** | 10/10 | 1725 |
| fpsol2.i.3 | 425 | 55 | 79 | **55** | **55** | **55** | **55** | **55** | 10/10 | 135 |
| inithx.i.1 | 864 | 54 | 66 | **54** | **54** | **54** | **54** | **54** | 9/10 | 11620 |
| inithx.i.2 | 645 | 35 | 93 | 36 | 36 | **35** | **35** | **35** | 10/10 | 3039 |
| inithx.i.3 | 621 | 36 | - | 37 | 37 | **36** | **36** | **36** | 10/10 | 4943 |
| zeroin.i.1 | 211 | 49 | 51 | **49** | **49** | **49** | **49** | **49** | 10/10 | 449 |
| zeroin.i.2 | 211 | 36 | 51 | **36** | **36** | **36** | **36** | **36** | 10/10 | 66 |
| zeroin.i.3 | 206 | 36 | 49 | **36** | **36** | **36** | **36** | **36** | 10/10 | 65 |
| myciel6 | 95 | 7 | **7** | **7** | **7** | **7** | **7** | **7** | 10/10 | 5 |
| myciel7 | 191 | 8 | **8** | **8** | **8** | **8** | **8** | **8** | 10/10 | 9 |
| 4_FullIns_3 | 114 | 7 | - | **7** | **7** | **7** | **7** | **7** | 10/10 | 10 |
| 4_FullIns_4 | 690 | 8 | **8** | **8** | **8** | **8** | **8** | **8** | 10/10 | 26 |
| 4_FullIns_5 | 4146 | 9 | **9** | **9** | **9** | **9** | **9** | **9** | 10/10 | 5191 |
| 1_Insertions_6 | 607 | 7 | **7** | **7** | **7** | **7** | **7** | **7** | 10/10 | 20 |
| 2_Insertions_5 | 597 | 6 | **6** | **6** | **6** | **6** | **6** | **6** | 10/10 | 17 |
| 3_Insertions_5 | 1406 | 9 | **6** | **6** | **6** | **6** | **6** | **6** | 10/10 | 641 |
| school1 | 385 | 15 | **15** | **15** | **15** | **15** | **15** | **15** | 10/10 | 16 |
| school1_nsh | 352 | 14 | **14** | **14** | **14** | **14** | **14** | **14** | 8/10 | 1181 |
| qg.order40 | 1600 | 40 | **40** | **40** | **40** | **40** | **40** | **40** | 10/10 | 44 |
| qg.order60 | 3600 | 60 | **60** | **60** | **60** | **60** | **60** | **60** | 10/10 | 655 |
| ash331GPIA | 662 | 4 | **4** | **4** | **4** | **4** | **4** | **4** | 10/10 | 554 |
| ash608GPIA | 1216 | 4 | **4** | **4** | **4** | **4** | **4** | **4** | 9/10 | 749 |
| ash958GPIA | 1916 | 4 | **4** | **4** | **4** | **4** | **4** | **4** | 4/10 | 2819 |

heuristics, in particular, the best-performing heuristics used in Section 6 as reference algorithms, while referring the reader to [15,33] for a comprehensive review for the GCP and the recent studies such as [26,41,40,43] for the ECP.

## *7.1 Heuristics for the GCP*

For the GCP, three main categories of heuristics have been proposed in the literature.

A first category of approaches are local search heuristics. They use iterative improvement procedures by choosing moves that decrease the cost function at each iteration until a local optimum is reached. In addition, advanced local search heuristics use different mechanisms to escape local minimums. One of the most popular local search heuristic for the graph coloring problem is the *TabuCol* algorithm [20]. The idea is to allow moves that can worsen the solution to escape local optima. However, reassigning to a vertex its former color is forbidden for a few iterations (with the help of the so-called tabu list), in order to avoid search cycling. One notices that *TabuCol* has been used as the key local optimization components of several state-of-the-art hybrid algorithms [16,30,31,35]. Recently, *TabuCol* has also been adopted as the local optimizer of the probability based learning method (PLSCOL) [45]. Finally, this single trajectory local search approach has been extended to multiple trajectory local search, which is exemplified by the Quantum Annealing algorithm (QA) [42]. In QA, $n$ candidate solutions are optimized by a simulating annealing algorithm, while the interactions between the solutions occur through a specific pairwise attraction process, encouraging the solutions to become similar to their neighbors and improving the spreading of good solution features among new solutions.

A second category of methods include hybrid algorithms, in particular, based on the *memetic* approach [19]. Indeed, these hybrid methods combine the benefits of local search methods for intensification with a population that maintains a diversity of the explored solutions. Different solutions evolve separately using a local search algorithm such as the above-mentioned *TabuCol*. And a crossover operator is used to create new candidate solutions from existing solutions. One of the most popular crossover operators is the Greedy Partition Crossover (GPX) introduced in the HEA algorithm [16] which follows the idea of grouping genetic algorithms of [12] and produces offspring by choosing alternatively the largest color class in the parent solutions. The MMT algorithm [31] completes a first phase of optimization with a *memetic* algorithm by a second phase of optimization based on the set covering formulation of the problem and using the best independent sets found so far. An extension of the HEA algorithm called MACOL (denoted as MA in this paper) has been proposed in [30]. It introduces a new adaptive multi-parent grouping crossover (AMPaX) and a new distance-and-quality based replacement mechanism used to maintain diversity of the solutions in the population. Recently, [35] proposed a new variation of the HEA approach called HEAD. HEAD maintains a population of only two solutions. In order to prevent premature convergence, HEAD introduces an innovative way for managing the diversification based on elite solutions. The idea is to keep in memory the best solutions (elites) obtained few generations before and to reintroduce them in the population when the two solutions becomes too similar. HEAD achieves new state-of-the-art solutions for several well-known instances of the GCP.

A third category of approaches is based on extracting large independent sets in order to split the original graph coloring problem into residual sub-graphs coloration problems. These methods are very effective to color large graphs with at least 1000 nodes [44].

One notices that the above-mentioned algorithms cover the current best-known results on the benchmark instances used in the last section for the CGP.

## *7.2 Heuristics for the ECP*

Most of the above-mentioned best performing methods for the GCP rely in one way or another on extraction of large independent sets or spreading of large building blocks between solutions. Thus, these methods cannot generally be applied as it stands for the ECP, because regrouping nodes in large sets may conflict with the equity constraint. Therefore different approaches have been proposed in the literature for the ECP.

One of the first method is the tabu search algorithm TabuEqCol [9], which is an adaptation of the *TabuCol* algorithm designed for the GCP [20] to the ECP. This algorithm has been improved in [26] by the backtracking based iterated tabu search (BITS), which embeds a backtracking scheme under the iterated local search framework. These two algorithms only consider feasible solutions with respect to the equity constraint, which may restrict too much the exploration of the space search.

A new category of methods relax the equity constraint and considers both equity-feasible and equity-infeasible solutions that facilitates the transition between visiting structurally different solutions. By enlarging their search spaces, these algorithms are able to improve on the results reported in [9,26]. The first work in this direction is the Feasible and Infeasible Search Algorithm (FISA) [41]. The hybrid tabu search (HTS) algorithm [43] is another important algorithm exploring both equity-feasible and equity-infeasible solutions and applying an additional novel cyclic exchange neighborhood. Finally, the memetic algorithm (MAECP) [40] is a population-based hybrid algorithm combining a backbone crossover operator, a two-phase feasible and infeasible local search and a quality-and-distance pool updating strategy. It is worth mentioning that FISA, HTS and MAECP cover the current best-known results on the benchmark instances used in the last section for the ECP.

## 8 Conclusion

In this paper, we presented GpGrad a new paradigm to solve grouping problems. The originality of our approach is to formulate the computation of a solution as a weight matrix optimization problem which is solved by first order gradient descent based weight learning. This framework is inspired from new techniques used in machine learning to optimize neural networks with hard threshold units.

We applied this general framework to solve the well-known and challenging graph coloring problem and the equitable coloring problem. A practical implementation, called TensCol, was proposed, which takes advantages offered by tensor calculus using parallel computing on GPU devices.

Experimental evaluations on popular DIMACS and COLOR02 challenge benchmark graphs showed that TensCol competes well with the best algorithms for both problems with an unified approach. Overall, even if TensCol does not find any new record result for the GCP (this rarely happened Since 2012), it provides good results in particular for large geometric graphs (such as R1000.5.col or latin_square_10.col) which are in general difficult to solve by the best performing *memetic* algorithms. For the ECP, our method finds 8 improved best results (new upper bounds) for the large geometric graphs and random graphs.

For future work, it would be interesting to apply the proposed approach to solve other grouping problems and applications. Furthermore, this work opens up two avenues for further research. A first perspective is to improve the optimization process of the learned weight matrix by using more sophisticated methods such as second order gradient descent with Hessian matrix, adaptive learning rate or momentum [5,38]. A second perspective could be to replace the weight matrix that handles the generative model of candidate solutions by a product of matrices or even a deep generative neural network [2] in order to learn more complex dependencies between the group assignments.

## References

[1] Amin Abbasi-Pooya and Ali Husseinzadeh Kashan. New mathematical models and a hybrid grouping evolution strategy algorithm for optimal helicopter routing and crew pickup and delivery. *Computers & Industrial Engineering*, 112:35–56, 2017.

[2] Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pages 226–234, 2014.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Una Benlic and Jin-Kao Hao. Hybrid metaheuristics for the graph partitioning problem. In *Hybrid Metaheuristics*, pages 157–185. 2013.

[5] Dimitri P Bertsekas. *Convex Optimization Algorithms.* Athena Scientific Belmont, 2015.

[6] John S Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in Neural Information Processing Systems*, pages 211–217, 1990.

[7] Jack Brimberg, Nenad Mladenovic, and Dragan Urosevic. Solving the maximally diverse grouping problem by skewed general variable neighborhood search. *Information Sciences*, 295:650–675, 2015.

[8] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering - Selected Results and Surveys*, pages 117–158. 2016.

[9] Isabel Méndez Díaz, Graciela Nasini, and Daniel Severín. A tabu search heuristic for the equitable coloring problem. In *International Symposium on Combinatorial Optimization*, pages 347–358. Springer, 2014.

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

[11] Emanuel Falkenauer. A grouping genetic algorithm for line balancing with resource dependent task times. In *Proceedings of the 1997 International Conference on Neural Information Processing and Intelligent Information Systems, Vol. I, Dunedin, New Zealand, 1997*, pages 464–468, 1997.

[12] Emanuel Falkenauer. *Genetic Algorithms and Grouping Problems.* John Wiley & Sons, Inc., 1998.

[13] Charles Fleurent and Jacques A Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63(3):437–461, 1996.

[14] Hanna Furmańczyk and Marek Kubale. The complexity of equitable vertex coloring of graphs. *Journal of Applied Computer Science*, 13(2):95–106, 2005.

[15] Philippe Galinier, Jean-Philippe Hamiez, Jin-Kao Hao, and Daniel Porumbel. Recent advances in graph vertex coloring. In *Handbook of Optimization*, pages 505–528. Springer, 2013.

[16] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[17] Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completness,*. WH Freeman, San Francisco, New York, USA, 1979.

[18] Steven Gold, Anand Rangarajan, et al. Softmax to softassign: Neural network algorithms for combinatorial optimization. *Journal of Artificial Neural Networks*, 2(4):381–399, 1996.

[19] Jin-Kao Hao. Memetic algorithms in discrete optimization. In *Handbook of Memetic Algorithms*, pages 73–94. Springer, 2012.

[20] Alain Hertz and Dominique de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[21] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[22] Diviyan Kalainathan, Olivier Goudet, Isabelle Guyon, David Lopez-Paz, and Michèle Sebag. Sam: Structural agnostic model, causal discovery and penalized adversarial learning. *arXiv preprint arXiv:1803.04929*, 2018.

[23] Ali Husseinzadeh Kashan, Mina Husseinzadeh Kashan, and Somayyeh Karimiyan. A particle swarm optimizer for grouping problems. *Information Sciences*, 252:81–95, 2013.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Xiangjing Lai and Jin-Kao Hao. Iterated maxima search for the maximally diverse grouping problem. *European Journal of Operational Research*, 254(3):780–800, 2016.

[26] Xiangjing Lai, Jin-Kao Hao, and Fred Glover. Backtracking based iterated tabu search for equitable coloring. *Engineering Applications of Artificial Intelligence*, 46:269–278, 2015.

[27] R. M. R. Lewis. *A Guide to Graph Colouring - Algorithms and Applications*. Springer, 2016.

[28] Rhydian Lewis and Ben Paechter. Finding feasible timetables using group-based operators. *IEEE Transactions on Evolutionary Computation*, 11(3):397–413, 2007.

[29] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through $l\_0$ regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[30] Zhipeng Lü and Jin-Kao Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250, 2010.

[31] Enrico Malaguti, Michele Monaci, and Paolo Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316, 2008.

[32] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

[33] Enrico Malaguti and Paolo Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 2010.

[34] Isabel Méndez-Díaz, Graciela Nasini, and Daniel Severín. A dsatur-based algorithm for the equitable coloring problem. *Computers & Operations Research*, 57:41–50, 2015.

[35] Laurent Moalic and Alexandre Gondran. Variations on memetic algorithms for graph coloring problems. *Journal of Heuristics*, 24(1):1–24, 2018.

[36] Carsten Peterson and Bo Söderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1(01):3–22, 1989.

[37] Arathi Ramani, Igor L Markov, Karem A Sakallah, and Fadi A Aloul. Breaking instance-independent symmetries in exact graph coloring. *Journal of Artificial Intelligence Research*, 26:289–322, 2006.

[38] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[39] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[40] Wen Sun and Jin-Kao Hao. Memetic search for the equitable coloring problem. *http://www.info.univ-angers.fr/pub/hao/papers/MAECPSunHaoJuly2019.pdf*, 2019.

[41] Wen Sun, Jin-Kao Hao, Xiangjing Lai, and Qinghua Wu. On feasible and infeasible search for equitable graph coloring. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 369–376. ACM, 2017.

[42] Olawale Titiloye and Alan Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376–384, 2011.

[43] Wenyu Wang, Jin-Kao Hao, and Qinghua Wu. Tabu search with feasible and infeasible searches for equitable coloring. *Engineering Applications of Artificial Intelligence*, 71:1–14, 2018.

[44] Qinghua Wu and Jin-Kao Hao. Coloring large graphs based on independent set extraction. *Computers & Operations Research*, 39(2):283–290, 2012.

[45] Yangming Zhou, Béatrice Duval, and Jin-Kao Hao. Improving probability learning based local search for graph coloring. *Applied Soft Computing*, 65:542–553, 2018.

[46] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. Reinforcement learning based local search for grouping problems: A case study on graph coloring. *Expert Systems with Applications*, 64:412–422, 2016.

## Appendices

### A Evaluation of $\frac{\partial p_{i,l}}{\partial w_{i,j}}$

For $i, j \in [\![1, n]\!] \times [\![1, k]\!]$,

$$p_{i,j} = \frac{e^{\theta w_{i,j}}}{\sum_{l=1}^{k} e^{\theta w_{i,l}}}. \tag{36}$$

Therefore, if $l = j$:

$$\begin{aligned}
\frac{\partial p_{i,l}}{\partial w_{i,j}} &= \frac{\partial \frac{e^{\theta w_{i,l}}}{\sum_{m=1}^{k} e^{\theta w_{i,m}}}}{\partial w_{i,j}} && (37)\\
&= \frac{\theta e^{w_{i,l}} \sum_{m=1}^{k} e^{\theta w_{i,m}} - \theta e^{\theta w_{i,l}} e^{\theta w_{i,j}}}{(\sum_{m=1}^{k} e^{\theta w_{i,m}})^2} && (38)\\
&= \theta \frac{e^{\theta w_{i,l}}}{\sum_{m=1}^{k} e^{\theta w_{i,m}}} \frac{\sum_{m=1}^{k} e^{\theta w_{i,m}} - e^{\theta w_{i,j}}}{\sum_{m=1}^{k} e^{\theta w_{i,m}}} && (39)\\
&= \theta p_{i,l}(1 - p_{i,j}) && (40)
\end{aligned}$$

If $l \neq j$, then,

$$\begin{aligned}
\frac{\partial p_{i,l}}{\partial w_{i,j}} &= \frac{-\theta e^{\theta w_{i,l}} e^{\theta w_{i,j}}}{(\sum_{m=1}^{k} e^{\theta w_{i,m}})^2} && (41)\\
&= -\theta \frac{e^{\theta w_{i,l}}}{\sum_{m=1}^{k} e^{\theta w_{i,m}}} \frac{e^{\theta w_{i,j}}}{\sum_{m=1}^{k} e^{\theta w_{i,m}}} && (42)\\
&= -\theta p_{i,l} p_{i,j} && (43)
\end{aligned}$$

The two cases, $l = j$ and $l \neq j$ corresponding to equations (40) and (43) can be summarized with a single expression as:

$$\frac{\partial p_{i,l}}{\partial w_{i,j}} = \theta p_{i,l}(\delta_{j,l} - p_{i,j}), \tag{44}$$

with $\delta_{j,l}$ the Kronecker symbol equal to 1 if $j = l$ and 0 otherwise.

## B Evaluation of the gradient of the GCP score with respect to a candidate solutions $S$

The GCP score for a solution $S$ is given by:

$$f(S) = \frac{1}{2} \sum_{i,j=1}^{n} a_{i,j} \sum_{l=1}^{k} s_{i,l} s_{j,l} \tag{45}$$

For $q, r \in [\![1, n]\!] \times [\![1, k]\!]$, the terms depending on $s_{q,r}$ in equation (45) are:

$$\frac{1}{2}(a_{q,q} s_{q,r}^2 + \sum_{i=1, i \neq q}^{n} a_{i,q} s_{i,r} s_{q,r} + \sum_{j=1, j \neq q}^{n} a_{q,j} s_{q,r} s_{j,r})$$

Therefore, for $q, r \in [\![1, n]\!] \times [\![1, k]\!]$, the partial derivative of $f(S)$ with respect to an element $s_{q,r}$ is:

$$\frac{\partial f(S)}{\partial s_{q,r}} = a_{q,q} s_{q,r} + \frac{1}{2} \sum_{i=1, i \neq q}^{n} a_{i,q} s_{i,r} + \frac{1}{2} \sum_{j=1, j \neq q}^{n} a_{q,j} s_{j,r} \tag{46}$$

As the matrix $A$ is symmetric, $a_{i,q} = a_{q,i}$, it gives:

$$\begin{aligned} \frac{\partial f(S)}{\partial s_{q,r}} &= a_{q,q} s_{q,r} + \frac{1}{2} \sum_{i=1, i \neq q}^{n} a_{q,i} s_{i,r} + \frac{1}{2} \sum_{j=1, j \neq q}^{n} a_{q,j} s_{j,r} && (47) \\ &= a_{q,q} s_{q,r} + \sum_{i=1, i \neq q}^{n} a_{q,i} s_{i,r} && (48) \\ &= \sum_{i=1}^{n} a_{q,i} s_{i,r} && (49) \end{aligned}$$

## C Evaluation of the total loss $\mathcal{L}(\mathbf{S})$ of TensCol with respect to a tensor of $D$ solutions $\mathbf{S}$

The total loss $\mathcal{L}(\mathbf{S})$ at time $t$ is given by:

$$\mathcal{L}(\mathbf{S})(t) = \sum_{d=1}^{D} f(S^d) + \lambda t \kappa(\mathbf{S}) - \mu t \varpi(\mathbf{S}) \tag{50}$$

For $p, q, r \in [\![1, D]\!] \times [\![1, n]\!] \times [\![1, k]\!]$, the partial derivative of $\mathcal{L}(\mathbf{S})(t)$ with respect to an element $s_{p,q,r}$ is:

$$\frac{\partial \mathcal{L}(\mathbf{S})(t)}{\partial s_{p,q,r}} = \frac{\partial \sum_{d=1}^{D} f(S^d)}{\partial s_{p,q,r}} + \lambda t \frac{\partial \kappa(\mathbf{S})}{\partial s_{p,q,r}} - \mu t \frac{\partial \varpi(\mathbf{S})}{\partial s_{p,q,r}} \tag{51}$$

Using the same evaluation as in the appendix B, we obtain:

$$\begin{aligned} \frac{\partial \sum_{d=1}^{D} f(S^d)}{\partial s_{p,q,r}} &= \frac{\partial f(S^p)}{\partial s_{p,q,r}} && (52) \\ &= \sum_{i=1}^{n} a_{q,i} s_{p,i,r} && (53) \end{aligned}$$

The partial derivative of $\kappa(\mathbf{S})$ with respect to an element $s_{p,q,r}$ is:

$$\begin{aligned} \frac{\partial \kappa(\mathbf{S})}{\partial s_{p,q,r}} &= \frac{\partial \sum_{i,j} a_{i,j} (\bar{v}_{i,j})^{\alpha}}{\partial s_{p,q,r}} && (54) \\ &= \alpha \sum_{i,j} a_{i,j} \frac{\partial \bar{v}_{i,j}}{\partial s_{p,q,r}} (\bar{v}_{i,j})^{\alpha-1} && (55) \\ &= \alpha \sum_{i,j} a_{i,j} \frac{\partial \sum_{d} \sum_{l=1}^{k} s_{d,i,l} s_{d,j,l}}{\partial s_{p,q,r}} (\bar{v}_{i,j})^{\alpha-1} && (56) \\ &= \alpha \sum_{i,j} a_{i,j} (\bar{v}_{i,j})^{\alpha-1} \frac{\partial \sum_{l=1}^{k} s_{p,i,l} s_{p,j,l}}{\partial s_{p,q,r}} && (57) \\ &= 2\alpha \sum_{j=1}^{n} a_{q,j} (\bar{v}_{q,j})^{\alpha-1} s_{p,j,r} && (58) \end{aligned}$$

An equivalent evaluation gives the partial derivative of $\varpi(\mathbf{S})$ with respect to an element $s_{p,q,r}$ as:

$$\frac{\partial \kappa(\mathbf{S})}{\partial s_{p,q,r}} = 2\beta \sum_{j=1}^{n} (1 - a_{q,j}) (\bar{v}_{q,j})^{\beta-1} s_{p,j,r} \tag{59}$$

Using equations (53), (58) and (59), the gradient of the total loss given by equation (51) can be rewritten with a single tensor operations as:

$$\nabla_{\mathbf{S}} \mathcal{L}(t) = A \cdot \mathbf{S} + 2\alpha\lambda t (A \odot \bar{V}^{\circ(\alpha-1)}) \cdot \mathbf{S} - 2\beta\mu t (\bar{A} \odot \bar{V}^{\circ(\beta-1)}) \cdot \mathbf{S} \tag{60}$$