A deep learning guided memetic framework for graph coloring problems

Olivier Goudet[a], Cyril Grelier[a], Jin-Kao Hao[a,*]

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

Knowledge-Based Systems, https://doi.org/10.1016/j.knosys.2022.10998

Abstract

Given an undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E, a graph coloring problem involves finding a partition of the vertices into different independent sets. In this paper we present a new framework that combines a deep neural network with the best tools of classical heuristics for graph coloring. The proposed method is evaluated on two popular graph coloring problems (vertex coloring and weighted coloring). Computational experiments on well-known benchmark graphs show that the proposed approach is able to obtain highly competitive results for both problems. A study of the contribution of deep learning in the method highlights that it is possible to learn relevant patterns useful to obtain better solutions to graph coloring problems.

Keywords: Population-based search; GPU-based parallel search; deep learning; heuristics; graph coloring.

1. Introduction

Graph coloring involves assigning colors to the vertices of a graph subject to certain constraints and optimization objective. The popular vertex coloring problem (COL) is the most representative example and can be stated as follows. Given an undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E, the COL is to color the vertices of V so that two adjacent vertices receive different colors and the number of colors used is minimized (this number is called the chromatic number of G, denoted by $\chi(G)$). This problem can also be seen as finding a partition of the vertex set V into a minimum number of color groups (also called independent sets or color classes) so that two vertices linked by an edge belong to different color groups. In some variants of this conventional coloring problem, one aims to find a legal coloring of the graph while considering an alternative optimization objective.

The typical search space of a graph coloring problem is composed of the partitions of vertices V into k color groups:

$$\mathcal{S} = \{\{V_1, V_2, \ldots, V_k\} : \cup_{i=1}^{k} V_i = V, V_i \cap V_j = \emptyset\}, \qquad (1)$$

where $i \neq j, 1 \leq i, j \leq k, 1 \leq k \leq |V|$. This search space $\mathcal{S}$ is huge in general and finding an optimal solution S^* is usually intractable unless P=NP, as most graph coloring problems are NP-hard.

Graph coloring problems have been studied very intensively in the past decades and many coloring methods have been proposed in the literature. A first category of methods are based on local search (also called neighborhood search). Starting from an initial solution typically constructed using a greedy heuristic, a local search algorithm improves the current solution by considering the best moves in a given neighborhood. To escape local optima traps, local search algorithms usually incorporate dedicated mechanisms such as tabu lists [2, 17] or perturbation strategies [22, 33]. However for very difficult instances of graph coloring, the single trajectory local search approach is not powerful enough to locate very high quality solutions mainly due to its limited diversification capacity. To overcome this difficulty, hybrid algorithms have been proposed, in particular relying on the population-based memetic framework that combines local searches and crossovers [32]. The memetic framework has been very successful in solving several graph coloring problems [11, 23, 27, 31, 35]. These hybrid algorithms combine the benefits of local search for intensification with a population of high-quality solutions offering diversification possibilities.

The memetic algorithms proposed in the literature for graph coloring typically use a small population with no more than 100 individuals. At each generation, one offspring solution is usually created by a crossover (or recombination) operator applied to two or more randomly selected individuals from the population. One of the most popular crossovers used for graph coloring problems is the Greedy Partition Crossover (GPX) introduced in the hybrid evolutionary algorithm (HEA) [11]. The GPX operator produces offspring by inheriting alternatively the largest color classes in the parent solutions. The resulting offspring is then improved by a local search procedure such as TabuCol [17].

The crossover operator within a memetic algorithm enables the creation of new restarting points for the local search procedure, which are expected to be more promising and better than a pure random initialization. However, when using such mechanisms, there is usually no way of knowing in advance whether the new restarting point really indicates a promising area that is worth being further examined by the local search procedure. Indeed, sometimes, the use of a crossover can bring the search process back to an already visited region of the search space without any chance of further im-

*Corresponding author.

Email addresses: olivier.goudet@univ-angers.fr (Olivier Goudet), Cyril.Grelier@univ-angers.fr (Cyril Grelier), jin-kao.hao@univ-angers.fr (Jin-Kao Hao)

provement, or to a new region far from the global optimum. Moreover, hybrid algorithms do not have a specific memory to store information about past searches and thus can hardly discover useful patterns that may exist among the solutions encountered during the search trajectories (though inheriting color classes with crossovers can be seen as some sort of "learning" of good patterns).

On the other hand, numerous algorithms have been proposed since decades from the machine learning community to leverage statistical learning methods for solving difficult combinatorial search problems (see the recent survey of [1] on this topic). These attempts have been given a new lease of life, with the emergence of deep learning techniques for combinatorial optimization problems [7, 46], inspired from the great success of the AlphaZero algorithm for combinatorial games [40]. In particular, some recent works have applied reinforcement learning and deep learning to solve graph coloring problems [20, 26]. Nevertheless, these studies rarely exploit specific knowledge of the problem, which makes these learning approaches more general but may limit their performance. Indeed, the results obtained by this type of approach are for the moment far from the results obtained by state of the art algorithms on graph coloring problems such as hybrid algorithms [27, 29, 31] and simulated annealing algorithms [44]. We can mention however recent studies which try to take advantage of efficient local search algorithms and machine learning techniques [14, 49] with promising results for graph coloring problems.

In this work, we aim to push further the integration of machine learning and combinatorial optimization, by proposing a new framework which combines deep neural networks with the best tools of "classical" metaheuristics for graph coloring, so as to solve very difficult graph coloring problems which still resist the best current methods. In order to achieve this integration, we revisit an idea proposed in [4] twenty years ago. In [4], Boyan and Moore remarked that the performance of a local search procedure depends on the state from which the search starts and therefore proposed to use a regression algorithm to predict the results of a local search algorithm. Once learned, this predictive model can help to select new good starting points for the local search and to accelerate the search process. We exploit this idea with the use of modern deep learning techniques to better select promising crossovers among those possible ones in each generation of a memetic algorithm. We design a specific neural network architecture for graph coloring problems inspired by deep set networks [48, 28], in order to make it invariant by permutation of the color classes. Furthermore, as training a deep neural network requires a large amount of data, we follow the recent work [15] to adopt a large population P ($|P| >\approx 10^4$) for the underlying memetic algorithm, whose individuals evolve in parallel in the search space. In order to learn the neural network and to compute all the local searches in parallel for all the individuals of the population, we leverage GPU (Graphic Processing Units) computation.

As a proof of concept, we apply this approach to solve the weighted vertex coloring problem (WVCP) and the vertex coloring problem (COL). The classical COL is well known and has been studied for a long time. The WVCP has recently attracted a lot of interest in the literature [16, 33, 38, 42, 47]. In the WVCP, a strictly positive weight w_v is associated to each vertex v. The goal of the problem is to find a legal coloring minimizing the global score

$$f(S) = \sum_{i=1}^{k} \max_{j \in V_i} w_j, \tag{2}$$

where V_i ($1 \leq i \leq k$) is a color class including all the vertices receiving color i and $\max_{j \in V_i} w_j$ is the largest weight of color class V_i.

One observes that the COL is a special case of the WVCP when the vertex weight w_v is equal to one for all the vertices. In this case, minimizing the function f (Eq. (2)) is equivalent to the minimization of the number of colors. In the presentation that follows, we focus on the WVCP. However, for computational assessments, we present experimental results for both the WVCP and the COL.

The WVCP has a number of practical applications in different fields such as matrix decomposition problems [38], batch scheduling [13] and manufacturing [19]. In addition to heuristic algorithms [16, 33, 38, 42, 47], it has been addressed by exact methods [6, 9, 30].

2. General framework - revisiting the STAGE algorithm with deep learning and memetic algorithm

Given a problem whose goal is to find an optimal solution with respect to a minimization objective f, the expected search outcome of a stochastic local search algorithm A can be defined as

$$\mathbb{E}[f_A(S)] = \sum_{S' \in \mathcal{S}} P(S \xrightarrow{\text{A}} S') f(S'), \tag{3}$$

where $P(S \xrightarrow{\text{A}} S')$ is the probability that the search starting from S will terminate in state S'. $\mathbb{E}[f_A(S)]$ evaluates the potential interest of S as a starting state for the algorithm A.

The main idea of the STAGE algorithm [4] was to approximate the expectation $\mathbb{E}[f_A(S)]$ by a regression approximation model $\hat{f}_A : \mathbb{R}^d \rightarrow \mathbb{R}$, taking as input the encoded real-valued feature vector $F(S)$ (with d features) of a state S. This function $\hat{f}_A$ can be a linear regression model or a more complex non linear model such as a neural network.

Starting from a first random initial solution S, and the function approximator $\hat{f}_A$, the STAGE algorithm evolves in three steps:

1. **Optimize f using A.** From S, it runs the local search algorithm A, producing a search trajectory that ends at a local optimum S'.
2. **Train $\hat{f}_A$.** For each point S_i on the search trajectory, use $\{(F(S_i), f(S')\}$ as a new training pair for the function approximator.
3. **Optimize $\hat{f}_A$ using hillclimbing.** Continuing from S', perform a hillclimbing search on the learned objective function $\hat{f}_A$. This results in a new state S which should be a new good starting point for A.

We revisit this idea with an adaptation for each of the three steps of the STAGE algorithm.

1. First, regarding the first step, we run in parallel p local searches with algorithm A starting from different states to generate p different search trajectories (instead of a single one). This

makes it possible to build a training dataset with a high diversity of examples.

2. Secondly, regarding step 2, we do not use any prior mapping F from states to features. Following the current trend in deep learning, the embedding of the state can be directly learned in an end-to-end pipeline with a deep neural network, denoted as $f_\theta : \mathcal{S} \to \mathbb{R}$. We make this neural network invariant by permutation of the group of colors V_j in the coloring $S \in \mathcal{S}$, which is a very important feature of all graph coloring problems, by adapting the deep set network architecture proposed in [48] (see Section 2.3).
3. Thirdly, in step 3 of the original STAGE algorithm, a hill-climbing algorithm is used to optimize the current solution guided by the objective function $\hat{f}_A$. However, as we address a very complex problem and we use a complex non-convex and non-linear function (deep neural network), it is difficult to optimize it using a hillclimbing algorithm. We tried to use more complex algorithms such as tabu search to optimize it, but there is a deeper problem, which is the question of generalization. Indeed, if a state S is too different from the states already seen before in the training dataset, and in particular if the color groups V_j that composed it are too different from the color groups already seen before by the neural network, we expect that $f_\theta(S)$ can be very inaccurate for the estimation of $\mathbb{E}[f_A(S)]$. Therefore, we propose to replace this hillclimbing procedure by a crossover operation between different members of a population of candidate solutions. By recombining the different color groups already seen before by the learning algorithm we expect the approximation of $\mathbb{E}[f_A(S)]$, given by $f_\theta(S)$, to be more precise.

The pseudo-code of the proposed new deep learning guided memetic framework for graph coloring (DLMCOL) is shown in Algorithm 1.

The algorithm takes a graph G as input and tries to find a legal coloring S with the minimum score $f(S)$. At the beginning, all the individuals of the population are initialized in parallel using a greedy random algorithm (cf. Section 2.1) and the neural network f_θ is initialized with random weights. Then, the algorithm repeats a loop (generation) until a stopping criterion (e.g., a cutoff time limit or a maximum number of generations) is met. Each generation t involves the execution of five components:

1. The p offspring individuals of the current population are simultaneously improved by running in parallel p local searches on the GPU to find new legal solutions with a minimum score f (cf. Section 2.2). For each of the p improved individuals from step 1, we record S'_i, the legal state with the lowest score $f(S'_i)$ obtained during each local search trajectory.
2. From these p local search trajectories, a supervised learning training dataset $\mathcal{D} = \{(X_i, y_i)\}_{i=1}^{p}$ is built with $X_i = S_i^O$ and $y_i = f(S'_i)$ for $1 \leq i \leq p$ and the neural network f_θ is trained on this dataset during N epochs (cf. Section 2.3).
3. The distances between all pairs of the existing individuals $\{S_1, \ldots, S_p\}$ and new individuals $\{S'_1, \ldots, S'_p\}$ are computed in parallel (cf. Section 2.4).
4. Then the population updating procedure (cf. Section 2.5) merges the $2p$ existing and new individuals to create a new population of p individuals, by taking into account the fitness f of each individual and the distances between individuals in order to maintain some diversity in the population.
5. Finally each individual is matched with its K nearest neighbors in the population. For each individual, K offspring solutions are generated and the one with the best expected score evaluated with the neural network f_θ is selected (cf. Section 2.6). After this selection procedure, p offspring individuals are selected and become the p new starting points $\{S_1^O, \ldots, S_p^O\}$ which are improved in parallel by the local search procedure during the next generation $(t + 1)$.

Algorithm 1 Deep learning guided memetic framework

Input: Graph $G = (V, E)$, population size p.
Output: The best legal coloring S^* found so far
$P = \{S_1, \ldots, S_p\} \leftarrow$ population_initialization /* Section 2.1
Initialize the neural network f_θ with random weights.
$S^* = \emptyset$ and $f(S^*) = \infty$
$\{S_1^O, \ldots, S_p^O\} \leftarrow \{S_1, \ldots, S_p\}$
repeat
  for $i = \{1, \ldots, p\}$, **in parallel do**
    $S'_i \leftarrow$ local_search(S_i^O) /* Section 2.2
  end for
  $S'^* = \text{argmin}\{f(S'_i), i = 1, \ldots, p\}$
  if $f(S'^*) < f(S^*)$ **then**
    $S^* \leftarrow S'^*$
  end if
  Build supervised learning training dataset $\mathcal{D} = \{(S_i^O, f(S'_i))\}_{i=1}^{p}$ and train the neural network f_θ on it. /* Section 2.3.
  $D \leftarrow$ distance_computation$(S_1, \ldots, S_p, S'_1, \ldots, S'_p)$ /* Section 2.4
  $\{S_1, \ldots, S_p\} \leftarrow$ pop_update$(S_1, \ldots, S_p, S'_1, \ldots, S'_p, D)$ /* Section 2.5
  $\{S_1^O, \ldots, S_p^O\} \leftarrow$ build_and_select_offsprings$(S_1, \ldots, S_p, f_\theta, D)$ /* Section 2.6
until stopping condition met
return S^*

The algorithm stops when a predefined condition is reached and returns the best recorded solution S^*. The subsequent subsections present the five components of this deep learning guided memetic framework applied to the WVCP. In order to show some generality of the proposed framework, an application of this approach to the vertex coloring problem is presented in Section 2.7.

2.1. Initialization with a greedy random algorithm for the WVCP

In order to initialize the individuals of the population, we use a randomized greedy procedure which is known to be very effective for the WVCP [33, 42].

First all the vertices are sorted in descending order of the weights and then in descending order of the degrees. Then a color is assigned to each vertex without creating conflicts by randomly choosing a color in the set of the already used color. If no color is available for the vertex i with weight w_i, a new color is created (and the score of the current solution is increased by w_i).

Notice that for the WVCP, the number of used colors to find a legal coloring S minimizing the global score $f(S)$ is unknown in advance. However, it is at least strictly greater than the chromatic number of the graph G. In our case, we use a predefined maximum number of colors k in order to specify the size of the layers of the neural network and to allocate memory for the local searches on the GPU. Specifically, we set k to be the maximum number of colors used in the initial solutions generated by the randomized greedy procedure. The new search space $\mathcal{S}$ restricted with the k available colors is composed of the partitions of vertices V into k color groups:

$$\mathcal{S}_k = \{\{V_1, V_2, \ldots, V_k\} : \cup_{i=1}^{k} V_i = V, V_i \cap V_j = \emptyset, i \neq j, 1 \leq i, j \leq k\}.$$

Note that the best solution found for each benchmark graph of the WVCP presented in Section 3 typically requires significantly less colors than k.

2.2. Parallel iterated feasible and infeasible Tabu Search

For local optimization, we employ a parallel iterated tabu search algorithm to simultaneously improve the individuals of the current population. It relies on the adaptive feasible and infeasible tabu search procedure (AFISA) proposed in [42], with some slight modifications. AFISA is a sequential procedure that improves a starting legal or illegal coloring by optimizing the fitness function g given by:

$$g(S) = f(s) + \phi \times c(S), \tag{4}$$

where $\phi \in \mathbb{R}$ is an adaptive coefficient for the penalty function $c(S) = \sum_{\{u,v\} \in E} \delta_{uv}$ with:

$$\delta_{uv} = \begin{cases} 1 & \text{if } u \in V_i, v \in V_j \text{ and } i = j \text{ and } i \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

AFISA improves the current coloring by successively changing the color of a vertex in the search space $\mathcal{S}_k$ (with a maximum of k colors). Such a change is called an one-move. To prevent the search from revisiting already visited colorings, a vertex cannot change its color for the next tt (called tabu tenure) iterations[1]. The tabu tenure is set to be $L + 0.2 \times |V|$, where L is a random integer from $[0; 9]$ and $|V|$ is the number of vertices in the graph.

Like in the AFISA algorithm, we perform successive searches by changing dynamically the value of the parameter ϕ in order to navigate in the space of legal and illegal colorings. The maximum number of successive local searches is set to $maxLS\,Iters = 10$. At the beginning the parameter is set to the value $k/(2|V|) * \max_{i=1,\ldots,|V|} w_i$ for each individual j of the population. At the end of each successive tabu search, if the best current solution S_j found by the tabu search procedure is legal ($c(S_j) = 0$), then ϕ is divided by 2 (in order to increase the chance of visiting infeasible solutions); otherwise ϕ is multiplied by 2 (in order to guide the search toward feasible regions)[2]. For the last iteration of this iterative tabu search algorithm, we set $\phi = 2 \times \max_{i=1,\ldots,|V|} w_i$ to make sure that each tabu search is forced at least one time toward a legal solution.

The pseudo code of the parallel iterative tabu search is shown in Algorithm 2, which runs on the GPU to raise the quality of the current population in parallel. All the data structures required during the search are stored in each local thread memory running tabu search except the information of the graph which is stored in the global memory and shared by all tabu search runs.

Algorithm 2 Parallel iterated tabu search with feasible and infeasible solutions

1: **Input:** Population $P = \{S_1^O, \ldots, S_p^O\}$, depth of tabu search $nbIter_{TS}$, maximum number of successive local searches $maxLS\,Iters$, weights w_i of each vertex i.
2: **Output:** Improved population $P' = \{S'_1, \ldots, S'_p\}$.
3: **for** $i = \{1, \ldots, p\}$, **in parallel do**
4: $S'_i = \emptyset$ and $f(S'_i) = \infty$ /* Records the best solution found so far on each local thread.
5: **end for**
6: $iter = 0$
7: **while** $iter < maxLS\,Iters$ **do**
8: **for** $i = \{1, \ldots, p\}$, **in parallel do**
9: $S_i \leftarrow$ feasible_and_infeasible_tabu_search($S_i, nbIter_{TS}, \phi_i$) /* Improve the solutions by running the tabu search procedure to minimize the extended fitness function g with search depth $nbIter_{TS}$ and penalization coefficient ϕ_i.
10: **if** $f(S_i) < f(S'_i)$ and $c(S_i) = 0$ **then**
11: $S'_i \leftarrow S_i$
12: **end if**
13: **if** $iter < maxLS\,Iters - 1$ **then**
14: **if** $c(S_i) = 0$ **then**
15: $\phi_i = \phi_i / 2$
16: **else**
17: $\phi_i = \phi_i \times 2$
18: **end if**
19: **else**
20: $\phi_i = 2 \times \max_{i \in 1,\ldots,|V|} w_i$
21: **end if**
22: **end for**
23: $iter = iter + 1$
24: **end while**
25: **return** $P' = \{S'_1, \ldots, S'_p\}$

2.3. Deep neural network training

Once all the parallel tabu searches are done, we collect the starting states S_i^O and the best score $f(S'_i)$ found on each local thread. These data are then used to build a supervised training dataset

[1] In the original AFISA algorithm, the tabu tenure concerns past moves (as in the original TabuCol algorithm [17]) instead of completely freezing a vertex. However we empirically observed that it is more effective to freeze a vertex that has just changed its color in order to avoid too much color changes of the same vertex without any improvement of the score (plateau).

[2] In the original AFISA algorithm, ϕ is initially set to 1 and cannot be lower than 1. The adaptive mechanism only increases or decreases its value by 1. We empirically observed that dividing or multiplying its value by 2 (instead of simply changing its value by 1) appears to be more effective for faster adjustments, especially for graphs with heavy weights.

$\mathcal{D} = \{S_i^O, f(S_i')\}_{i=1}^p$ with p examples whose entries are the S_i^O in $\mathcal{S}_k$ and the corresponding targets are real values $f(S_i')$.

A neural network $f_\theta : \mathcal{S}_k \to \mathbb{R}$, parametrized by a vector θ (initialized at random at the beginning), is successively trained on each new dataset $\mathcal{D}$ produced at each generation (online training) in order to be able to be more and more accurate at predicting the expected score obtained after the local search procedure for any new starting point $S \in \mathcal{S}_k$.

This neural network f_θ takes directly as input a coloring S as a set of k vectors V_j, $S = \{V_1, \ldots, V_k\}$, where each V_j is a binary vector of size n indicating if the vertex i belongs to the color group j. For such an entry S, the neural network outputs a real value noted $f_\theta(S) \in \mathbb{R}$.

For the WVCP, one important characteristic of our neural network f_θ is that it is invariant by permutation of the group of colors of any solution S given as input. It should be a function f_θ from $\mathcal{S}_k$ to $\mathbb{R}$ so that for any permutation σ of the input color groups

$$f_\theta(V_{\sigma(1)}, \ldots, V_{\sigma(k)}) = f_\theta(V_1, \ldots, V_k). \tag{6}$$

As indicated in [28, 48], such permutation invariant functions can be obtained by combining the treatments of each color group vector V_j with an additional "color-averaging" operation that performs an average of the features across the different color groups. It has notably been shown in [28] that such operations are sufficient to recover all invariant functions from $\mathcal{S}_k$ to $\mathbb{R}$.

Using the notations proposed in [28], for a coloring $S = \{V_1, \ldots, V_k\}$ the color group invariant network f_θ is defined as

$$f_\theta(S) = \frac{1}{k} \sum_{i=1}^{k} (\phi_{\theta_P} \circ \phi_{\theta_{P-1}} \circ \cdots \circ \phi_{\theta_0}(S))_i, \tag{7}$$

where each ϕ_{θ_j} is a permutation invariant function from $\mathbb{R}^{k \times l_{j-1}}$ to $\mathbb{R}^{k \times l_j}$, where l_j's are the layer sizes. Note that for the first layer $l_{j-1} = |V|$.

Each layer operation ϕ_θ with l input features and L output features includes a weight matrix $\Lambda \in \mathbb{R}^{l \times L}$ that treats each color group independently, a color-mixing weight matrix $\Gamma \in \mathbb{R}^{l \times L}$ and a bias vector $\beta \in \mathbb{R}^l$.

As in classical multi-layer feed-forward neural networks, Λ processes each color group vector of the solution S independently. Then, the weight matrix Γ processes the average vector $\rho(S)$ computed across the different k color groups for each feature given by

$$\rho(S) = \rho(V_1, \ldots, V_k) \tag{8}$$

$$= \frac{1}{k} \sum_{i=1}^{k} V_i. \tag{9}$$

The output of the layer ϕ_θ is a matrix in $\mathbb{R}^{k \times L}$, which is the concatenation of k output vectors of size L:

$$\phi_\theta(S) = (\phi_\theta(S)_1, \ldots, \phi_\theta(S)_k), \tag{10}$$

where for $1 \leq i \leq k$,

$$\phi_\theta(S)_i = \text{LeakyReLU}_{0.2}(\beta + V_i\Lambda + \rho(S)\Gamma). \tag{11}$$

$\text{LeakyReLU}_{0.2}$ is a non linear activation function defined as

$$\text{LeakyReLU}_{0.2}(x) = \max(0.2 \times x, x).$$

After each local search procedure the neural network f_θ is trained during N epochs on the new dataset $\mathcal{D}$ using Adam optimizer [24] with initial learning rate $lr = 0.001$ and batches of size $b = 100$ in order to minimize the mean square error loss (MSE) between the outputs and the targets. In order to speed up the training, prior to the non linearity we apply a batch normalization layer [21], adapted to keep the invariant property of the network. For the layer j of the network, the output of the invariant batch normalization layer is

$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \mu, \tag{12}$$

where x is the input of size (b, k, l_j). The mean and standard-deviation of x are vectors of size l_j calculated over the mini-batches and all the k colors (in order to keep the invariance property). γ and μ are learnable parameter in $\mathbb{R}$. $\epsilon = 10^{-5}$ is a value added to the denominator for numerical stability. As in the original work of [21], during training the batch normalization layer keeps running estimates of its computed mean and variance, which are then used for normalization during evaluation. The running estimates are kept with a default momentum of 0.1.

Once learned, this neural network will be used to select new crossovers for the next generation (see Section 2.6 below), but before performing crossovers, we must decide if the new legal colorings obtained after the parallel tabu search procedure can be inserted into the population. For this purpose, a distance-and-quality based pool update strategy is used to create a new population satisfying a minimum spacing among the individuals to ensure population diversity [37]. Maintaining this minimum spacing requires the computation of pairwise distances between the solutions, which is presented in the next subsection.

2.4. Distance computation

Following [14, 27, 35], for population updating, we use a $p \times p$ matrix to record all the distances between any two solutions of the population. This symmetric matrix is initialized with the $p \times (p-1)/2$ pairwise distances computed for each pair of individuals in the initial population, and then updated each time a new individual is inserted in the population.

To merge the p new solutions and the p existing solutions, we need to evaluate (i) $p \times p$ distances between each individual in the population $P = \{S_1, \ldots, S_p\}$ and each improved individual in $P' = \{S_1', \ldots, S_p'\}$ and (ii) $p \times (p-1)/2$ distances between all the pairs of individuals in P'. All the $p \times p + p \times (p-1)/2$ distance computations are independent from one another, and are performed in parallel on the GPU (one computation per thread).

Given two colorings S_i and S_j, we use the set-theoretic partition distance $D(S_i, S_j)$ to measure the dissimilarity between S_i and S_j, which corresponds to the minimum number of vertices that need to be displaced between color classes of S_i to transform S_i to S_j [36]. The exact partition distance between two solutions can be calculated with the Hungarian algorithm [25] in $O(|V| + n^3)$ time. However, given that we need to compute millions of distances at

each generation with the large population, we instead adopt the efficient approximation algorithm presented in [36], which scales in $O(|V|)$.

2.5. Population update

According to [35, 37], the population update procedure aims to keep the best individuals, but also to ensure a minimum spacing distance between the p individuals. The update procedure is sequential, as we need to compare one by one existing individuals in the population $P_t = \{S_1, \ldots, S_p\}$ at generation t and the tabu search improved offspring solutions in the population $P' = \{S'_1, \ldots, S'_p\}$.

We use the population update procedure proposed in [14]. This procedure greedily adds one by one the best individuals of $P^{all} = \{S_1, \ldots, S_p\} \cup \{S'_1, \ldots, S'_p\}$ in the population of the next generation P_{t+1} until P_{t+1} reaches p individuals, so that $D(S_i, S_j) > |V|/10$ ($|V|$ is the number of vertices), for any $S_i, S_j \in P_{t+1}$, $i \neq j$. Each $D(S_i, S_j)$ corresponds to the approximation of the set-theoretic partition distance which was precomputed in the last step of the algorithm.

2.6. Parent matching and selection of crossovers with the neural network

At each generation, each individual of the population is matched with its K nearest neighbors in the population (in the sense of the distance evaluated in subsection 2.4). We do not consider performing crossovers between individuals too far away in the search space as this may result in poor quality offsprings (cf. [31]).

For each individual i, K offspring solutions S_i^j ($1 \leq j \leq K$) are generated using the well-known GPX crossover [11, 31], where the individual i is taken as the first parent and its neighbor as the second parent (the GPX crossover is not symmetric).

For each individual i, among these K crossovers, we select the one with the best expected score evaluated with the neural network of Section 2.3:

$$S_i^0 = \underset{S_i^j, 1 \leq j \leq K}{\operatorname{argmin}} f_\theta(S_i^j). \tag{13}$$

After this selection procedure, p offspring solutions are identified that serve as the p new starting points $\{S_1^O, \ldots, S_p^O\}$ of the parallel tabu search procedure during the next generation $(t+1)$.

2.7. Adaptations of the algorithm for the vertex coloring problem

The vertex coloring problem COL aims at finding the smallest k for a given graph G (its chromatic number $\chi(G)$) so that G admits a legal coloring using k colors. Following the literature on graph coloring [10], we tackle this problem by solving a series of k-coloring problems (k-COL) with decreasing k values. Starting from an initial number of k colors, as soon as a legal k-coloring is found, k is decreased by one. This process is repeated until no legal solution with k colors can be found and the last k admitting a legal k-coloring defines an upper bound of the chromatic number of the graph. Let k be the given colors and $S = \{V_1, V_2, \ldots, V_k\}$ be a candidate k-coloring, the k-COL problem can be seen as the optimization problem that aims to minimize the number of conflicts given by $f(S)$ (until it reaches 0):

$$f(S) = \sum_{\{u,v\} \in E} \delta_{uv}, \tag{14}$$

where

$$\delta_{uv} = \begin{cases} 1 & \text{if } u \in V_i, v \in V_j \text{ and } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

For this minimization problem, we use the same deep learning guided memetic framework presented in the last subsections. The only parts of the general framework that require specific adaptations for the k-COL concern the initialization and the parallel local search. For the k-COL, we use a pure random initialization procedure. At the beginning, for each individual of the population, each node $v \in V$ receives a random color in $\{1, \ldots, k\}$. For the local search, we run in parallel the popular TabuCOL algorithm [17] (i.e., its efficient implementation presented in [11]) on the GPU to raise the quality of the current population during $128 \times |V|$ iterations, where $|V|$ is the order of the graph. TabuCOL is launched with its default parameters setting like in [31].

3. Experimental results

This section is dedicated to a computational assessment of the proposed deep learning memetic framework for solving the weighted vertex coloring problem and the conventional vertex coloring problem, by making comparisons with state-of-the-art methods.

3.1. Benchmark instances

We carried out extensive experiments on the WVCP benchmark graphs used in the recent studies [33, 41, 47]: the pxx, rxx, DIMACS/COLOR small, and DIMACS/COLOR large instances. The pxx and rxx instances are based on matrix-decomposition problems [38], while DIMACS/COLOR small [6, 9] and DIMACS/COLOR large [41] are based on DIMACS and COLOR competitions.

As indicated in [33, 47], for the WVCP, a preprocessing procedure can be applied to reduce a graph G with the set of weight W. For each clique C_l with l vertices, if we note w' the smallest weight of this set, all the vertices i in the graph with a degree equal to $l-1$ and a weight $w_i < w'$ can be removed from the graph without changing the optimal WVCP score of this instance. Enumerating all the cliques of the graph is a challenging problem. We used the *igraph python package*[3] with a timeout of ten seconds for all instances. For small instances it is enough to enumerate all the cliques of a graph. For our experimental evaluation, DLMCOL as well as all the competitors take these reduced graph as input.

For the vertex coloring problem COL, we conducted experiments on the classical DIMACS benchmark graphs used in most of the best coloring methods for this problem [31, 44]. These instances can be separated into two categories: *easy* instances and *difficult* instances. For the *easy* instances, most recent heuristics can reach the chromatic number (or its best known upper bound) in a short amount of time, while for the *difficult* instances, no single algorithm is able to reach the chromatic number or the best known

[3] `https://igraph.org/python/`

result for all these graphs. In this section, we only report the results for the most *difficult* instances. The results for the *easy* DIMACS instances are summarized in Appendix A.

For the WVCP and the COL, due to local memory limit on each thread of the GPU for the local searches, the DLMCOL algorithm was not runned on the biggest instances of the DIMACS/COLOR benchmarks (when $n \times k > 200000$): C2000.5, C2000.9, DSJC1000.9, r1000.5 and wap01-4a.

3.2. Implementation and parameter setting

The DLMCOL algorithm was coded in Python with the Numba 0.53.1 library for CUDA kernel implementation (local searches, distance computations, crossovers). The neural network was implemented in Pytorch 1.8.1. DLMCOL is specifically designed to run on GPUs. In this work we used a V100 Nvidia graphic card with 32 GB memory. The code of DLMCOL is publicly available at `https://github.com/GoudetOlivier/DLMCOL_WVCP`.

The population size p of DLMCOL is set to $p = 20480$, which is chosen as a multiple of the number of 64 threads per block. This large population size offers a good performance ratio on the Nvidia V100 graphics cards, while remaining reasonable for pairwise distance calculations in the population, as well as the memory occupation on the GPU for medium instances ($n \leq 500$). However for large instances ($n > 500$ and $k > 90$), we set $p = 8192$ in order to limit the global memory occupation on the device.

At each generation of DLMCOL, each of the p tabu searches is executed on a single GPU thread independently. For each tabu search, the neighboring solutions at the current iteration are evaluated efficiently with the incremental evaluation method of [11]. For fast memory access, a per-thread local memory is used to store specific local information such as the current solution being improved and the tabu tenure. The threads are grouped by block of size 64 and launched on the GPU grid. No per-block shared memory is used because the tabu searches are run independently on different block. However, a global memory is employed to store general information about the graph such as its adjacency matrix to avoid information duplication. All these p tabu searches are launched with a CUDA kernel function written in Numba and the best results obtained during each tabu search are transferred to the CPU after synchronization. Note that these tabu searches performed on the GPU could be replaced by CPU implementations (e.g. run on a multi-core environment). However, it was practical in our case to run the tabu searches on the same GPU device that is already used for training the neural network.

The number of tabu iterations $nbIter_{TS}$ depends on the size $|V|$ of the graph. The maximum number of iterated tabu searches launched at each generation, $LSIters$, is set to 10. The minimum spacing distance MS used for pool update is set to $\frac{|V|}{10}$.

For the neural network we implemented an architecture with 4 hidden layers of size $5|V|$, $2|V|$, $|V|$ and $|V|//2$ for the WVCP and 9 hidden layers of size $10|V|$, $5|V|$, $2|V|$, $2|V|$, $2|V|$, $2|V|$, $2|V|$, $|V|$ and $|V|//2$ for the k-COL problem. The neural network is trained at each generation with Adam optimizer [24] and initial learning rate $l_r = 0.001$.

Tables 1 and 2 summarize the parameter settings for the WVCP and the k-COL problems, which can be considered as the default and were used for all our experiments.

Table 1: Parameter setting in DLMCOL for the WVCP and the COL

Parameter	Description	Value
p	Population size	20480 (8192)
$maxLSIters$	Maximum number of successive local searches	10
$nbIter_{TS}$	Depth of tabu search	$10 \times \|V\|$
α	Tabu tenure parameter	0.2
MS	Minimum spacing between two individuals	$\frac{\|V\|}{10}$
l_r	Learning rate of the neural network	0.001
N	Number of epochs of the training	20
K	Number of considered neighbors for crossover selection	32

Table 2: Parameter setting in DLMCOL for the k-COL problem

Parameter	Description	Value
p	Population size	20480 (8192)
$nbIter_{TS}$	Depth of tabu search	$128 \times \|V\|$
α	Tabu tenure parameter	0.6
MS	Minimum spacing between two individuals	$\frac{\|V\|}{10}$
l_r	Learning rate of the neural network	0.001
N	Number of epochs of the training	5
K	Number of considered neighbors for crossover selection	16

3.3. Comparative results on weighted vertex coloring benchmarks

This section shows a comparative analysis on the pxx, rxx, DIMACS/COLOR small, and DIMACS/COLOR large instances with respect to the state-of-the-art methods [33, 41, 47]. The reference methods include the three best recent heuristics: AFISA [41], RedLS [47] and ILS-TS [33]. When they are available, we also include the optimal scores obtained with the exact algorithm MWSS [6] and extracted from [33].

Given the stochastic nature of the DLMCOL algorithm, each instance was independently solved 10 times. For small instances presented in Tables 3–4, a time limit of 1 hour was used. However for medium and large instances in Tables 5 and 6, as training the neural network and performing all the tabu searches with the large population is time consuming, a cutoff limit of 48 hours was retained.

For a fair comparison, we also launched the reference methods RedLS [47] and ILS-TS [33] during 48 hours on a computer with an Intel Xeon E5-2670 processor (2.5 GHz and 2 GB RAM), until no improvement was observed. As the available AFISA binary code does not allow setting a cutoff time, we only report its results mentioned in the original article [41]. However, we acknowledge that the comparison remains difficult in terms of computational time between DLMCOL and the competitors, as DLMCOL was run on GPUs while the other algorithms, AFISA, RedLS and ILS-TS used CPUs. Therefore the timing information is provided for indicative purposes only.

Columns 1, 2, and 3 of Tables 3–6 show the characteristics of each instance (i.e., name of the instance, number of vertices $|V|$, and optimal score reported in the literature if available). Columns 4-9 present the best and average scores obtained by the reference algorithms, as well as the average time in second required to obtain their best results. The results of the proposed DLMCOL algorithm are reported in columns 10 and 11. Boldfaced numbers show the dominating values while a star indicates a new upper bound[4].

[4]The certificates of the new best solutions from DLMCOL for the WVCP are available at `https://github.com/GoudetOlivier/DLMCOL_WVCP`.

The optima for the instances of Tables 3-5 are known except for four instances DSJC125 and two instances R100. As a result, no algorithm can further improve these bounds. For these graphs, the proposed algorithm and the latest ILS-TS algorithm report the same results and both algorithms perform better than AFISA and RedLS. However, the computation time required by DLMCOL to achieve its results is in general higher than the reference algorithms in particular when compared with ILT-TS. This is not really surprising given that the neural network training requires additional computation time in addition to the time needed by the optimization components.

For the larger instances reported in Table 6, DLMCOL obtains excellent results by reaching the best-known score for 31 over 49 instances. For 11 of them, DLMCOL even finds new upper bounds that had never been reported before. In particular, improvements are quite important for 3 instances with a high reduction of the best-known scores: DSJC500.5 from 707 to 685, flat1000_50_0 from 1184 to 924 and latin_square_10 from 1542 to 1480.

However, DLMCOL does not work well for large sparse graphs with low edge density such as DSJC1000.1, inithhx.i.2, inithhx.i.3 and wapXXa. For these graphs, it seems that it is very hard for the neural network to learn a common backbone of good solutions. An analysis of these negative results is proposed in Section 4.2.

For the largest graphs, we notice that the DLMCOL algorithm converges slowly, but continually. Even after 48 hours, DLMCOL still improves its solutions. This indicates that the algorithm is not trapped in local optima, which is a common problem for most existing WVCP algorithms. For the graphs DJSC1000.5, flat1000_60_0 and flat1000_76_0, DLMCOL was able to obtain still better new upper bounds of 1185, 1162, 1165, after 138, 98 and 95 hours, respectively, raising the total number of improved upper bounds to 14 for the WVCP.

For the large instances in Table 6, DLMCOL and the best competitors RedLS and ILS-TS have their own advantage respectively, while the proposed algorithm has the best overall success rate of 63% against AFISA (10%), RedLS (53%) and ILS-TS (49%).

3.4. Comparative results on vertex coloring benchmark

We show in this section the generality of the proposed approach by applying the approach to the vertex coloring problem. We present computational results on the 18 difficult DIMACS instances for the COL with respect to 12 state-of-the-art graph coloring methods. These instances are challenging because only a few algorithms can reach the best known results shown in Table 7 and only very few algorithms can attain the best known results for five graphs DSJC500.5, DSJC1000.5, flat_300_28_0, flat_1000_76_0, latin_sqr_10. Given that the COL is a special case of the WVCP when the vertex weight is equal to one, we also tested, for the first time, the two best WVCP algorithms, RedLS [47] and ILS-TS [33], on these difficult DIMACS instances[5]. Each algorithm was run 10 times to solve each instance under the relaxed condition of 48 hours per instance and per run. We observe that these local search based WVCP algorithms perform similarly compared to other popular local search coloring algorithms (see Table 8 below).

[5]AFISA [41] was not launched as the available binary code does not allow setting a cutoff time

Table 7 summarizes the computational results of our DLMCOL algorithm. Columns 2 and 3 give the features of the tested instance: the number of vertices $|V|$ and the density of the graph (dens.). Columns 4-6 present the chromatic number of the graph ($\chi(G)$, when known) and the best upper bound (k^*) reported so far in the literature with the references, including the two WVCP algorithms (RedLS [47] and ILS-TS [33]). In columns 7-9, the computational statistics of our DLMCOL algorithm are given, with the best (smallest) number of colors obtained to reach a legal solution (k_{best}), the associated success rate (SR) and the average time in seconds to reach the solution with the given k.

Following the common practice to report comparative results in the coloring literature, we also display in Table 8 the best solution found by each algorithm corresponding to the smallest number k of colors needed to reach a legal coloring for a graph. Column 2 corresponds to the best k found in the literature. Columns 3 displays the best k found by DLMCOL. Columns 4-9 and columns 10-17 respectively report the best k found by state-of-the art local search algorithms, including the two WVCP algorithms (RedLS [47] and ILS-TS [33]), and population based algorithms. Results corresponding to the best k found so far are displayed in boldface.

In Table 8, we regroup the three references [43], [44] and [45] in the same column (Column 16) as they correspond to the same QA-COL algorithm launched with different parameters and using different computing tools. We report the best k jointly reported in these three references.

It should be mentioned that these DIMACS instances have been studied for a long time (over thirty years) and some of the best known results have only been obtained by very few algorithms and sometimes with a very low success rate. Different computing tools have been used (such as multiple core servers with parallel computing) under specific and relaxed conditions (e.g., large run time from several days to one month, specific fine tuning of the hyperparameters for each given instance, etc.).

As displayed in Tables 7 and 8, DLMCOL can reach all the best results in the literature for these instances except for latin_sqr_10, for which a solution with $k = 97$ was only found once in [44]. In general, DLMCOL is very competitive for solving very difficult instances of medium size such as DSJC500.5 and flat300_28_0. Notably it finds a legal 47-coloring for DSJC500.5 with a success rate of 5/10 (only two reference algorithms QA-COL [45] and HEAD [31] can reach this result occasionally with specifically fine-tuned hyperparameters). DLMCOL also finds a solution with $k = 28$ colors for flat300_28_0, which is difficult for the two best competitors QA-COL [43, 44, 45] and HEAD [31]. For this instance, performing a strong exploration of the search space with a large population is very beneficial, as it seems that there exists only one legal solution with $k = 28$ colors. Indeed, dozens of solutions found by our algorithm are always the same up to color permutation. Finally, one notices that the DLMCOL algorithm is quite time consuming to solve large instances, given that it uses a very large population.

4. Analysis of important factors in the algorithm

In this section, we analyze the impacts of two important factors of the DLMCOL algorithm: (i) the very large population and (ii) the contributions of deep learning.

Table 3: Comparative results of DLMCOL with the state-of-the-art methods (AFISA, RedLS, ILS-TS) for DIMACS/COLOR small instances of the WVCP. Dominating results are indicated in boldface.

Instance			AFISA		RedLS		ILS-TS		DLMCOL	
Graph name	$\|V\|$	Opti.	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
DSJC125.1g	124	-	**23** (24)	3016	**23**	0.01	**23**	3	**23**	27
DSJC125.1gb	124	-	**90** (92.5)	402	91 (91.7)	696	**90**	15	**90**	28
DSJC125.5g	125	-	**71** (72.3)	216	72	32895	**71**	77	**71**	183
DSJC125.5gb	125	-	243 (250.2)	369	241 (241.3)	13528	**240**	219	**240**	40
DSJC125.9g	125	169	**169** (169.9)	16	**169**	3493	**169**	1.27	**169**	51
DSJC125.9gb	125	604	**604** (605.5)	444	**604**	17.62	**604**	59.8	**604**	51
GEOM100	100	65	**65** (65.0)	0.81	**65** (67.5)	0.01	**65**	0.03	**65**	27
GEOM100a	100	89	**89** (89.5)	110	90 (93.3)	0.01	**89**	0.65	**89**	30
GEOM100b	100	32	**32** (33.1)	59	**32**	0.6	**32**	0.02	**32**	18
GEOM110	110	68	**68** (68.0)	34	**68** (69.9)	0.03	**68**	0.05	**68**	19
GEOM110a	110	97	**97** (97.8)	177	**97** (99.4)	0.04	**97**	0.58	**97**	20
GEOM110b	110	37	**37** (37.9)	131	**37** (37.71)	22.1	**37**	0.25	**37**	27
GEOM120	120	72	**72**	33	**72** (73.1)	9.0	**72**	0.03	**72**	25
GEOM120a	120	105	**105** (106.3)	156	**105** (105.9)	1.96	**105**	0.77	**105**	30
GEOM120b	120	35	**35** (37.3)	67.7	**35** (35.25)	14.38	**35**	0.81	**35**	33
GEOM30b	30	12	**12**	0.02	**12**	0.01	**12**	0.01	**12**	20
GEOM40b	40	16	**16**	0.03	**16** (16.6)	0.01	**16**	0.01	**16**	20
GEOM50b	50	18	**18**	0.02	**18** (18.2)	62.02	**18**	0.01	**18**	15
GEOM60b	60	23	**23**	0.22	**23**	0.01	**23**	0.01	**23**	20
GEOM70	70	47	**47**	5	**47** (48.6)	0.01	**47**	0.02	**47**	19
GEOM70a	70	73	**73**	4	**73** (73.6)	0.25	**73**	0.03	**73**	20
GEOM70b	70	24	**24**	12	**24**	15.1	**24**	0.01	**24**	24
GEOM80	80	66	**66**	2	67 (67.4)	0.01	**66**	0.01	**66**	20
GEOM80a	80	76	**76** (76.1)	137	**76** (78.2)	1.3	**76**	0.04	**76**	23
GEOM80b	80	27	**27** (27.8)	67	**27**	24	**27**	0.06	**27**	12
GEOM90	90	61	**61** (61.2)	89	**61** (63.5)	1.74	**61**	0.15	**61**	21
GEOM90a	90	73	**73** (74)	512	**73** (74.1)	3.65	**73**	0.55	**73**	23
GEOM90b	90	30	**30** (30.1)	67	**30** (30.1)	0.17	**30**	0.02	**30**	25
R100_1g	100	21	**21** (22)	114	**21** (21.8)	508.0	**21**	7.14	**21**	300
R100_1gb	100	81	**81** (83.8)	3	**81** (81.4)	1279.7	**81**	1.98	**81**	27
R100_5g	100	-	**59** (60.1)	7	**59**	193	**59**	0.2	**59**	28
R100_5gb	100	-	221 (224.1)	187	**220** (222)	687	**220**	4	**220**	26
R100_9g	100	141	**141** (141.3)	21	**141**	0.62	**141**	40.8	**141**	36
R100_9gb	100	518	**518** (5449.3)	1152	**518**	8.17	**518** (518.3)	1066.1	**518**	35
R50_1g	50	14	**14**	0.14	**14** (14.1)	0.01	**14**	0.01	**14**	19
R50_1gb	50	53	**53** (53.0)	0.24	**53** (53.1)	0.07	**53**	0.01	**53**	19
R50_5g	50	37	**37** (37.0)	0.95	**37**	0.01	**37**	0.02	**37**	24
R50_5gb	50	135	**135** (135.3)	4	**135**	0.09	**135**	0.21	**135**	20
R50_9g	50	74	**74**	1	**74**	0.02	**74**	0.01	**74**	21
R50_9gb	50	262	**262**	13	**262**	0.01	**262**	1.4	**262**	22
R75_1g	75	18	**18** (18.4)	11	**18**	2.62	**18**	0.28	**18**	20
R75_1gb	75	70	**70** (70.1)	2	**70** (72.4)	0..35	**70**	0.23	**70**	20
R75_5g	75	51	**51** (51.4)	01	**51** (51.2)	598.6	**51**	0.39	**51**	22
R75_5gb	75	186	**186** (189)	19	**186**	51.1	**186**	2	**186**	23
R75_9g	75	110	**110**	3	**110**	0.08	**110**	0.1	**110**	27
R75_9gb	75	396	**396** (396.4)	146	**396**	0.42	**396**	3.9	**396**	26
Best rate			95%		89%		100%		100%	

Table 4: Comparative results of DLMCOL with the state-of-the-art methods (AFISA, RedLS, ILS-TS) for pxx instances of the WVCP. Dominating results are indicated in boldface.

Instance			AFISA		RedLS		ILS-TS		DLMCOL	
Graph name	$\|V\|$	Opti.	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
P06	16	565	**565**	0.0	**565**	0.01	**565**	0.01	**565**	21
P07	24	3771	**3771**	0.0	**3771** (3773.5)	0.01	**3771**	0.01	**3771**	18
P08	24	4049	**4049**	0.2	**4049**	0.03	**4049**	0.21	**4049**	18
P09	25	3388	**3388** (3388.2)	1	**3388**	0.01	**3388**	0.01	**3388**	20
P10	16	3983	**3983**	0.7	**3983**	0.01	**3983**	0.01	**3983**	18
P11	18	3380	**3380**	0.0	**3380**	0.01	**3380**	0.01	**3380**	18
P12	26	657	**657**	0.0	**657**	0.01	**657**	0.01	**657**	19
P13	26	3220	**3220** (3221.1)	0.7	**3220** (3229)	0.01	**3220**	0.09	**3220**	20
P14	31	3157	**3157**	0.0	**3157**	0.01	**3157**	0.01	**3157**	19
P15	34	341	**341**	1.8	**341** (343.1)	0.01	**341**	0.01	**341**	24
P16	34	2343	**2343**	0.7	**2343** (2383.3)	0.01	**2343**	0.01	**2343**	21
P17	37	3281	**3281** (3322.2)	2.7	**3281** (3282.6)	0.05	**3281**	0.01	**3281**	21
P18	35	3228	**3228**	0.1	**3228**	0.01	**3228**	0.01	**3228**	22
P19	36	3710	**3710**	0.4	**3710**	0.01	**3710**	0.01	**3710**	22
P20	37	1830	**1830** (1841)	4.9	**1830** (1844)	0.05	**1830**	0.38	**1830**	22
P21	38	3660	**3660** (3660.5)	0.8	**3660** (3707.0)	0.01	**3660**	0.01	**3660**	23
P22	38	1912	**1912** (1912.2)	0.3	**1912** (1946)	0.21	**1912**	0.01	**1912**	19
P23	44	3770	**3770** (3793.0)	0.3	**3770** (3804)	0.04	**3770**	0.01	**3770**	19
P24	34	661	**661**	0.0	**661** (667.1)	0.19	**661**	0.01	**661**	18
P25	36	504	**504**	0.3	**504**	0.01	**504**	0.01	**504**	19
P26	37	520	**520**	0.1	**520**	0.01	**520**	0.01	**520**	19
P27	44	216	**216**	0.1	**216** (219)	0.01	**216**	0.07	**216**	20
P28	44	1729	**1729** (1735.1)	2.6	**1729**	0.01	**1729**	0.01	**1729**	19
P29	53	3470	**3470**	0.1	**3470**	0.01	**3470**	0.01	**3470**	19
P30	60	4891	**4891**	54	**4891** (4901)	0.01	**4891**	0.01	**4891**	20
P31	47	620	**620**	3.7	**620**	0.01	**620**	0.01	**620**	18
P32	51	2480	**2480**	0.4	**2480**	0.01	**2480**	0.01	**2480**	20
P33	56	3018	**3018** (3029.7)	0.4	**3018** (3096)	0.02	**3018**	0.01	**3018**	22
P34	74	1980	**1980** (1980.5)	3.0	**1980** (1994)	0.01	**1980**	0.03	**1980**	26
P35	86	2140	**2140** (2145.0)	4.5	**2140** (2161)	0.01	**2140**	0.02	**2140**	23
P36	101	7210	**7210** (7385)	0.1	**7210**	0.01	**7210**	0.01	**7210**	27
P38	87	2130	**2130** (2139.5)	9.5	2140 (2161)	0.01	**2130**	0.29	**2130**	25
P40	86	4984	**4984** (5016.6)	5.1	5005 (5082.7)	0.01	**4984**	0.33	**4984**	25
P41	116	2688	**2688** (2688.1)	0.1	**2688** (2785.8)	0.04	**2688**	0.32	**2688**	453
P42	138	2466	**2466** (2671.2)	931.0	2482 (2539.9)	0.02	**2466**	9.02	**2466**	35
Best rate				100%		91%		100 %		100%

4.1. Sensitivity to the population size

We perform a sensitivity analysis of the algorithm with respect to the population size p. For this, we launched the DLMCOL algorithm with p taking nine different values in the range $[100, 50000]$ to solve the instance DSJC500.5 of the WVCP with the same total number of tabu search iterations. Figure 1 displays the sensitivity of the average results over 10 runs to the population size p.

We observe that for the same total number of tabu search itera-

Table 5: Comparative results of DLMCOL with the state-of-the-art methods (AFISA, RedLS, ILS-TS) for rxx instances of the WVCP. Dominating results are indicated in boldface.

Instance			AFISA		RedLS		ILS-TS		DLMCOL	
Graph name	$\|V\|$	Opti.	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
r01	144	6724	**6724** (6727.8)	49.5	6732 (6769.2)	0.01	**6724**	0.96	**6724**	36
r02	142	6771	**6771** (6780.6)	85.3	6774 (6818.6)	0.01	**6771**	0.25	**6771**	35
r03	139	6473	**6473** (6490.8)	190.2	6505 (6597.7)	233.5	**6473**	4.53	**6473**	33
r04	151	6342	**6342** (6403.2)	467.3	6349 (6427.7)	0.42	**6342**	0.83	**6342**	38
r05	142	6408	**6408** (6466.3)	71.7	6411 (65010.3)	0.42	**6408**	2.57	**6408**	38
r06	148	7550	**7550** (7555.9)	29.2	**7550** (7558.9)	0.01	**7550**	0.01	**7550**	38
r07	141	6889	**6889** (7555.9)	34.8	6910 (6974.2)	954	**6889**	7.29	**6889**	37
r08	138	6057	**6057** (6080.3)	311.7	6071 (6147.4)	0.04	**6057**	1.6	**6057**	234
r09	129	6358	**6358** (6393.8)	395.2	6390 (6451.9)	66.13	**6358**	1.84	**6358**	35
r10	150	6508	**6508** (6519.3)	461.1	6518 (65078.6)	0.05	**6508**	2.46	**6508**	39
r11	208	7654	**7654** (7710.6)	9542.2	7691 (7739.5)	489.45	**7654**	5.27	**7654**	432
r12	199	7690	7691 (7710.4)	9542.2	7694 (7730.2)	2.61	**7690**	4.33	**7690**	58
r13	217	7500	7521 (7558.3)	619.5	7524 (7566.7)	0.04	**7500**	5.8	**7500**	66
r14	214	8254	**8254** (8283.9)	8044.1	8288 (8371.4)	0.87	**8254**	4.78	**8254**	60
r15	198	8021	**8021** (8126.8)	2559.1	**8021** (8024.0)	0.01	**8021**	0.01	**8021**	54
r16	188	7755	**7755** (7789.2)	195.5	7764 (7809.4)	0.01	**7755**	11.22	**7755**	51
r17	213	7979	**7979** (8030.3)	855.4	8011 (8064.3)	0.86	**7979**	4.39	**7979**	242
r18	200	7232	**7232** (7278.9)	868.2	7240 (7295.3)	11.53	**7232**	26.4	**7232**	4374
r19	185	6826	6840 (6868.1)	395.5	**6826** (6850.5)	39.15	**6826**	2.09	**6826**	189
r20	217	8023	**8023** (8102.0)	1028.5	8031 (8138.3)	1.68	**8023**	13.08	**8023**	3027
r21	281	9284	**9284** (9384.5)	4588.7	9294 (9320.1)	0.01	**9284**	9.15	**9284**	6103
r22	285	8887	**8887** (8959.3)	12911	8924 (9030.6)	0.01	**8887**	63.42	**8887**	1521
r23	288	9136	**9136** (9267.9)	3252.0	9145 (9222.0)	0.05	**9136**	42.77	**9136** (9137.7)	25716
r24	269	8464	**8464** (8572.9)	13142.6	8468 (8534.5)	0.01	**8464**	0.51	**8464**	797
r25	266	8426	8468 (8560.8)	874.8	8579 (8649.6)	0.01	**8426**	36.03	**8426**	7088
r26	284	8819	**8819** (8927.9)	14225.1	8937 (9035.3)	0.01	**8819**	82.2	**8819**	22861
r27	259	7975	**7975** (8019.7)	14074.9	**7975** (7997.3)	1.71	**7975**	9.86	**7975**	102
r28	288	9407	**9407** (9599.4)	8691.0	9409 (9475.4)	0.01	**9407**	0.44	**9407**	1891
r29	281	8693	**8693** (8743.7)	7613.1	8701 (8743.7)	0.03	**8693**	4.54	**8693**	3429
r30	301	9816	**9816** (10003.2)	8838.6	9820 (9877.1)	0.01	**9816**	1.36	**9816**	147
Best rate				87%		13.3%		100 %		100%

Table 6: Comparative results of DLMCOL with the state-of-the-art methods (MWSS, AFISA, RedLS, ILS-TS) for DIMACS/COLOR large instances of the WVCP. Dominating results are indicated in boldface. New upper bounds are displayed with a star.

Instance			AFISA		RedLS		ILS-TS		DLMCOL	
Graph name	$\|V\|$	Opti.	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
C2000.5	2000	-	2400 (2425.1)	3134.0	**2151** (2162.4)	11827	2250 (2266.2)	11030	-	-
C2000.9	2000	-	6228 (6284.0)	2798.3	**5486** (5507.9)	166246	5808 (5849.3)	161980	-	-
DSJC1000.1	1000	-	359 (362.9)	430.5	**300** (302.6)	98115	305 (305.9)	97025	342.0 (344.5)	1105
DSJC1000.5	1000	-	1357 (1430.9)	371.7	**1220** (1228.4)	234	1242 (1270.4)	1929	1230.0 (1260.0)	167639
DSJC1000.9	1000	-	3166 (3231.0)	490.2	**2864** (2875.7)	48298	2975 (2997.8)	101238	-	-
DSJC250.1	250	-	140 (141.9)	48.9	130 (132)	1	**127** (127)	2576	**127** (127)	1353
DSJC250.5	250	-	415 (428.1)	269.2	404 (407.7)	43579	393 (393.3)	58615	**392**	9226
DSJC250.9	250	934	939 (943.2)	926	940 (943.11)	801.22	**934** (936.4)	7670.05	**934**	4722
DSJC500.1	500	-	210 (215.6)	426	188 (189.8)	50	**184** (185.4)	72845	**184** (184.9)	23049
DSJC500.5	500	-	778 (845.1)	159.3	726 (729.0)	2466	707 (716.9)	51407	**685*** (688.4)	47064
DSJC500.9	500	-	1790 (1854.5)	831.1	1681 (1685.3)	148914	1692 (1702.5)	136637	**1662*** (1664)	121518
DSJR500.1	500	-	**169** (175.4)	458.9	171 (173.4)	1	**169** (169)	0.56	**169** (171.7)	35389
flat1000_50_0	1000	-	1289 (1315.7)	981.8	1184 (1186.7)	93711	1218 (1226.4)	141135	**924***	82068
flat1000_60_0	1000	-	1338 (1354)	201.9	**1220** (1230.6)	264	1242 (1258.0)	60631	1224 (1231.6)	168937
flat1000_76_0	1000	-	1314 (1337.6)	2396.6	**1200** (1210.5)	316	1240 (1246.7)	141292	1210.0 (1220.6)	164934
inithx.i.1	864	-	587 (587.9)	527.5	**569** (571.4)	2	**569**	¡ 0.01	**569**	214
inithx.i.2	645	-	341 (341.6)	0.01	**329** (331.5)	1328	**329**	184	336 (337.0)	127
inithx.i.3	621	-	352 (355.6)	0.01	**337** (337.4)	5	**337**	1	350 (352.2)	142
latin_square_10	900	-	1690 (1900.0)	780.3	1548 (1561.8)	36451	1542 (1557.0)	142925	**1480*** (1486.2)	149656
le450_15a	450	-	241 (247.1)	288.4	217 (218.7)	1	**213** (213.8)	83054	**213** (216.5)	43854
le450_15b	450	-	239(245.1)	368.3	219 (225.3)	1	**217** (218.2)	2505	**217** (219.6)	71563
le450_15c	450	-	313 (320.8)	432.9	288 (292.2)	2	277 (280.3)	17392	**275*** (277.4)	46789
le450_15d	450	-	306 (314.1)	113.7	285 (288.9)	979	274 (275.8)	155735	**272*** (272.4)	22917
le450_25a	450	-	317 (329.9)	362.3	308 (310.4)	45732	**306** (306.0)	715	307 (308.6)	26228
le450_25b	450	-	318 (325.8)	285.9	308 (310.4)	45732	**307** (307.0)	18	**307** (308.8)	43456
le450_25c	450	-	378 (387.9)	359.4	360 (364.1)	2	349 (352.1)	82371	**342*** (343)	57924
le450_25d	450	-	375 (385.3)	254.8	342 (350.2)	4	339 (342.4)	16881	**330*** (330.1)	45128
miles1000	128	431	432 (444.7)	480.0	**431**	6.06	**431**	18.81	**431**	581
miles1500	128	797	**797**	1802	**797** (797.3)	0.03	**797**	1.68	**797**	76
miles250	128	102	**102** (102.7)	56.6	**102** (103.6)	0.7	**102**	0.18	**102**	21
miles500	128	260	**260** (261.3)	48.4	**260** (260.4)	1.39	**260**	0.18	**260**	30
queen10_10	100	-	166 (169.2)	68.4	**162** (165.3)	10400	**162** (162.0)	24	**162** (162)	19
queen11_11	121	-	178 (182.3)	55.2	179 (180.1)	909	**172** (172.7)	68208	**172** (172)	1753
queen12_12	144	-	194 (198.6)	92.7	188 (189.4)	133726	**185** (185.2)	23709	**185** (185)	1826
queen13_13	169	-	204 (207.5)	199.8	197 (200.9)	8100	**194** (194.8)	20802	**194** (194)	1150
queen14_14	196	-	224 (227.4)	360.1	217 (219.7)	2	216 (217.1)	469	**215*** (215.2)	16621
queen15_15	225	-	237 (241.2)	183.4	230 (233.5)	1	224 (226.7)	2685	**223*** (224.1)	9276
queen16_16	256	-	253 (256.3)	300.9	242 (246.0)	579	238 (239.0)	60187	**234*** (234.8)	14751
wap01a	2368	-	638 (653.1)	1133.5	**545** (558.6)	89184	549 (550.5)	62497	-	-
wap02a	2464	-	637 (638.1)	3270.46	**538** (547.1)	40143	541 (543.1)	46848	-	-
wap03a	4730	-	687 (707.5)	2901.5	**562** (566.9)	134923	577 (579.7)	55554	-	-
wap04a	5231	-	698 (709.0)	4.79	**563** (583.0)	52833	570 (573.0)	538455	-	-
wap05a	905	-	598 (610.9)	1574.5	**542** (548.1)	1348	**542** (542.8)	62323	587 (588.0)	1139
wap06a	947	-	599 (607.6)	65.3	**519** (529.4)	134	**519** (520.8)	46077	574 (587.0)	1244
wap07a	1809	-	680 (692.5)	384.82	**561** (563.8)	3433	567 (571.0)	11943	685 (686.0)	8294
wap08a	1870	-	663 (673.4)	2627.2	**529** (540.3)	37505	546 (549.7)	67847	663 (665)	7397
zeroin.i.1	211	511	518	0.01	**511**	0.08	**511**	1.7	**511**	38
zeroin.i.2	211	336	**336** (337.6)	440.8	**336**	0.22	**336**	0.01	**336**	33
zeroin.i.3	206	298	299 (301.7)	139.6	**298** (298.7)	2.25	**298**	10.67	**298**	31
Best rate				10.2%		53.1%		48.9 %		63.3%

tions, DLMCOL obtains better results with a large population size of 20000 than with other sizes. This can be explained by two reasons. First, a large population improves the population diversity, which favors the finding of promising areas in the search space and helps to better train the neural network at each generation. Second, a large population increases the chance for each individual to find a closer but different neighbor in the population for parent matching, which helps to generate promising offspring solutions. However, an excessively large population (such as $p = 50000$) is counterproductive, because the algorithm requires, in this case, much more

Table 7: Computational results of DLMCOL on the *difficult* DIMACS coloring challenge benchmarks for the COL problem

						DLMCOL		
Instance	$\|V\|$	dens.	$\chi(G)$	k^*	references	k_{best}	SR	t(s)
DSJC250.5	250	0.5	?	28	[3, 11, 12, 14, 18, 27, 29, 31, 34, 44, 33, 47]	**28**	10/10	219
DSJC500.1	500	0.1	?	12	[3, 5, 8, 11, 12, 14, 18, 27, 29, 31, 34, 44]	**12**	10/10	2396
DSJC500.5	500	0.5	?	47	[31, 45]	**47**	5/10	42513
DSJC500.9	500	0.9	?	126	[5, 12, 14, 18, 27, 31, 34, 44, 47]	**126**	10/10	17172
DSJC1000.1	1000	0.1	?	20	[3, 11, 12, 14, 18, 27, 29, 31, 34, 44]	**20**	10/10	32477
DSJC1000.5	1000	0.5	?	82	[31, 45]	**82**	4/10	167038
DSJR500.1c	500	0.97	85	85	[3, 11, 14, 18, 29, 31, 33, 34, 44, 47]	**85**	10/10	4395
DSJR500.5	500	0.47	122	122	[14, 27, 29, 31, 33, 39, 44, 47]	**122**	10/10	4310
r250.5	250	0.48	65	65	[14, 27, 29, 31, 33, 44, 33, 47]	**65**	10/10	19239
r1000.1c	1000	0.97	?	98	[3, 8, 14, 27, 29, 31, 33, 34, 44, 47]	**98**	10/10	2102
le450_25c	450	0.17	25	25	[3, 14, 18, 27, 29, 31, 34, 44]	**25**	10/10	48201
le450_25d	450	0.17	25	25	[3, 14, 18, 27, 29, 31, 34, 44]	**25**	10/10	45038
flat_300_26_0	300	0.48	26	26	[5, 8, 14, 27, 29, 33, 44, 47]	**26**	10/10	113
flat_300_28_0	450	0.17	28	28	[3, 18]	**28**	4/10	24651
flat_1000_50_0	1000	0.49	50	50	[3, 8, 12, 18, 27, 29, 31, 33, 34, 44, 47]	**50**	10/10	7160
flat_1000_60_0	1000	0.49	60	60	[8, 3, 12, 18, 27, 29, 31, 34, 44, 47]	**60**	10/10	12110
flat_1000_76_0	1000	0.49	76	81	[31, 45]	**81**	3/10	249165
latin_sqr_10	900	0.76	?	97	[43]	98	8/10	170518

Table 8: Comparison of DMLCOL with the state-of-the-art algorithms in terms of the best results on the *difficult* DIMACS coloring challenge benchmarks for the COL problem

			Local search algorithms						Population based algorithms							
Instance	k^*	DLMCOL	[8]	[5]	[18]	[3]	[47]	[33]	[11]	[12]	[29]	[34]	[27]	[42-44]	[31]	[14]
DSJC250.5	28	**28**	-	-	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**
DSJC500.1	12	**12**	**12**	**12**	**12**	**12**	13	13	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
DSJC500.5	47	**47**	50	49	48	48	50	50	48	48	48	48	48	**47**	**47**	48
DSJC500.9	126	**126**	127	**126**	**126**	127	**126**	127	-	**126**	127	**126**	**126**	**126**	**126**	**126**
DSJC1000.1	20	**20**	21	-	**20**	**20**	21	21	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
DSJC1000.5	82	**82**	90	89	86	89	91	91	83	84	83	83	83	**82**	**82**	84
DSJR500.1c	85	**85**	-	-	**85**	**85**	**85**	**85**	**85**	86	**85**	**85**	-	**85**	**85**	**85**
DSJR500.5	122	**122**	-	124	125	125	**122**	**122**	-	127	**122**	124	**122**	**122**	**122**	**122**
r250.5	65	**65**	-	-	66	66	**65**	**65**	-	-	**65**	-	**65**	**65**	**65**	**65**
r1000.1c	98	**98**	**98**	-	-	**98**	**98**	**98**	-	-	**98**	**98**	**98**	**98**	**98**	**98**
le450_25c	25	**25**	-	26	**25**	**25**	26	26	26	26	**25**	**25**	**25**	**25**	**25**	**25**
le450_25d	25	**25**	-	26	**25**	**25**	26	26	-	26	**25**	**25**	**25**	**25**	**25**	**25**
flat_300_26_0	26	**26**	**26**	**26**	-	-	**26**	**26**	-	**26**	**26**	-	**26**	**26**	**26**	**26**
flat_300_28_0	28	**28**	31	31	**28**	**28**	31	32	31	31	31	31	29	31	31	31
flat_1000_50_0	50	**50**	**50**	-	**50**	**50**	**50**	**50**	-	**50**	**50**	**50**	**50**	**50**	**50**	**50**
flat_1000_60_0	60	**60**	**60**	-	**60**	**60**	**60**	90	-	**60**	**60**	**60**	**60**	**60**	**60**	**60**
flat_1000_76_0	81	**81**	89	-	85	87	89	91	83	84	82	82	82	**81**	**81**	83
latin_sqr_10	97	98	-	99	-	-	101	100	-	104	101	-	99	**97**	-	98

time to converge toward solutions of good quality.

4.2. Analysis of the contributions of deep learning

We now analyze the interest of the neural network within the memetic framework. First, we study the general benefits of the crossover selection strategy driven by deep learning by performing an ablation study. Secondly, we compare the predicted results of local searches with the actual results, so as to shed lights on why this strategy is effective for some instances and less effective for others.

Benefits of the neural network based crossover selection. To assess the contributions of the neural network driven crossover selection strategy (see Section 2.6), we launched 10 replications of DLMCOL on 4 instances (DSJC500.1, DSJC500.5, le450_25c and le450_25d) of the WVCP with a cutoff time of 20 hours, with and without the neural network crossover selection. In the version without neural network, the second parent of the crossover is randomly chosen among the K nearest neighbors of each individual.

The average best score of the WVCP obtained at each generation is displayed in Figure 2. The green curve corresponds to the standard DLMCOL algorithm while the red curve corresponds to the version without deep learning. One first notices that the version without deep learning can perform more generations in the same amount of time because no time is spent on the neural network training and offspring evaluations. On the other hand, we observe that the green curve is always below the red curve and that better results are achieved in the same amount of time. This highlights that the neural network can really contribute to a better selection of promising crossovers for the memetic algorithm.

Quality of the predictions given by the neural network. Once trained, the neural network is used in DLMCOL to predict in advance the results of local searches in order to select the best promising crossovers for the next generation (see Section 2.6). Therefore, one can wonder if the fitness values predicted by the neural network for the p new starting points, $\{f_\theta(S_1^O), \ldots, f_\theta(S_p^O)\}$ at generation t are close

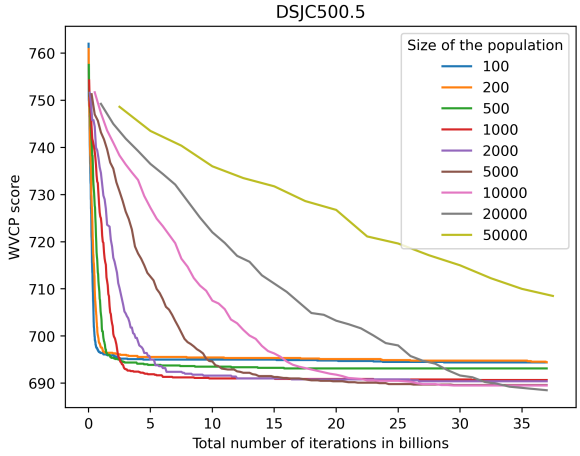

Figure 1: The impact of the population size p on the performance of DLMCOL. y-axis corresponds to the WVCP score (average best score over 10 runs at each generation) and x-axis corresponds to the number of generations.

to the actual fitness values $\{f(S'_1), \ldots, f_\theta(S'_p)\}$ obtained after the p local searches at the next generation $t+1$.

We recorded these predicted and actual fitness values at every generation of the search process for several instances of the WVCP (DSJC500.5.col, wap05a.col, DSJC1000.1.col, le450_25c.col and le450_25d.col). In Figures 3, and 4, we present two typical patterns of the evolution of the quality of the neural network prediction over the generations that we observed for these instances. For some graphs such as DSJC500.5.col, the neural network makes more and more precise predictions on average over generations, but for other graphs such as wap05a.col, the neural network does not really improve its predictions over time.

Figure 3 displays three scatter plots at generation 1, 16 and 31 where, the x-axis and y-axis respectively correspond to the predicted WVCP scores (generation 0, 15 and 30) and the actual WVCP scores (generation 1, 16 and 31) obtained after the local search for the instance DSJC500.5.col for all the $p = 20480$ individuals of the whole population. In the bottom right corner is displayed a boxplot of the prediction error in percent for the $p = 20480$ local searches at generation 1, 16 and 31. One first observes that the neural network is quite inaccurate at generation 1 (in red), because the relative prediction error is quite high, around 8.7% and the Pearson correlation coefficient between the predicted and actual results is equal to 0.015. However, at generation 16 (in blue), the neural network can provide more accurate predictions of the WVCP score than can be obtained by the local searches. Indeed the relative error is below 1.4% and the Pearson correlation coefficient gets to the value of 0.42. At generation 31, we observe that the neural network makes huge mistakes in the prediction, but only for one part of the samples, which can be explained by the fact that the prediction of the correct fitness values becomes more and more difficult as the search for very good solutions becomes more unpredictable. However at generation 31, the median of the relative error is lower for the whole population.

Figure 4 displays the same comparison between expected and actual results of the local searches for the large graph wap05a.col with a low edge density for which we have shown in Section 3.3 that DLMCOL fails for this type of instance. An attempt to explain this failure can be seen in Figure 4, where we observe that the relative prediction error is always high (around 2 and 3 %), but more importantly, the Pearson correlation calculated between the predicted and actual scores is respectively equal to 0.13, 0.05 and 0.01 at generation 1, 16 and 31, which means that the neural network is not able to really distinguish promising new starting points for local search among all the possible ones.

It can be explained by the fact that good solutions for this type of instances are typically characterized by a low ratio of the number of color groups over the total number of vertices. As for the WVCP, only the maximum weight of each color group has an impact on the score, many different groupings of vertices are possible without impacting the score for these instances. It results in a very high average distance between the best solutions of the population. Therefore, the neural network fails to learn relevant patterns in this too diversified population.

Figure 5 confirms this intuition by showing that the average distance between the individuals in the population (red solid line) remains very high over the generations (around 680 for a maximum of 849) during the resolution of the instance wap05a.col. Conversely for the instance DSJC500.5.col, the average distance between individuals (green solid line) decreases over generations, showing that the best individuals retained in the population share backbones of good solutions.

The green and red areas in Figure 5 are delimited by the maximum and minimum values of the distances between all the individuals in the population for the instances wap05a.col (red) and DSJC500.5.col (green). We remark that a minimum distance of 50 is reached after 25 generations for the instance DSJC500.5.col and this distance does not drop below 50. It comes from the fact that a minimum spacing between the individuals is imposed during the insertion of new individuals in the population (cf. Section. 2.5).

5. Conclusion

A deep learning guided memetic framework for graph coloring problems was presented, as well as an implementation on GPU devices to solve the classical vertex k-coloring problem and the weighted vertex coloring problem. This approach uses the deep set architecture to learn an invariant by color permutation regression model, useful to select the most promising crossovers at each generation. Additionally, it can take advantage of GPU computations to perform massively parallel local optimization with a large population to ensure a high degree of search intensification while maintaining a suitable degree of population diversification.

The proposed approach was assessed on popular DIMACS and COLOR challenge benchmarks of the two studied coloring problems. The computational results show that the algorithm competes globally well with the best algorithms for both problems. For the vertex coloring problem, it can reach most of the best known results of the literature for difficult instances. For the weighted coloring problem, it can find 14 new upper bounds for very difficult instances and significantly improves the previous best results for three graphs. An analysis of the predicted and actual fitness values after local search shows that the neural network can help to some

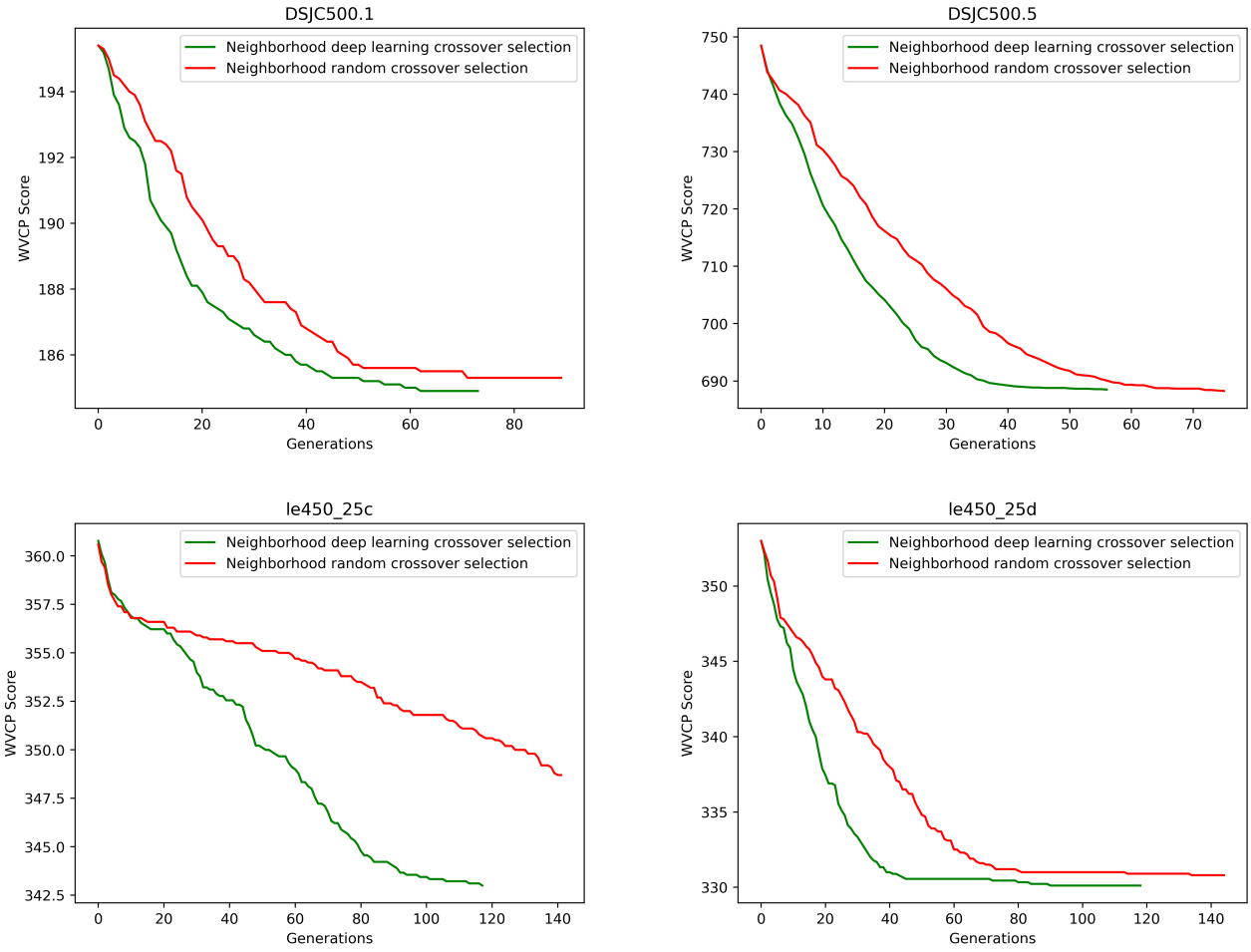

Figure 2: The impact of the deep learning driven crossover selection strategy on the algorithm. y-axis corresponds to the WVCP score (average best score over 10 runs at each generation) and x-axis corresponds to the number of generations.

extent in finding promising new good starting points at each generation, which eases the discovery of high quality solutions in the search space.

The achieved results reveal however three main limitations of the proposed approach. First, due to the memory capacity on the GPU devices we used, the DLMCOL algorithm has trouble to deal with very large instances ($|V| \geq 1000$). In particular, for the parallel local searches, the memory available on each thread of the GPU can be a huge limitation. Secondly, the algorithm has a slow convergence in comparison with sequential local search algorithms, due to its large population and the time spent to train the neural network at each generation. Thirdly, the algorithm fails for large instances with a low density (sparse graphs) for the WVCP, as for these instances the neural network has trouble to learn good patterns to effectively guide the selection of promising crossovers.

Other future works could be envisaged. In particular, it would be interesting to test the DLMCOL framework with the same type of neural network architecture to solve other graph coloring problems. Moreover, it could be worth applying deep learning techniques to learn a specific crossover for the weighted graph coloring problem instead of the classical GPX crossover used in this work. Finally, other neural network structures, such as graph convolutional neural networks, could be investigated to overcome the difficulty encountered on sparse graphs.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We would like to thank the reviewers for their useful comments. We also thank Dr. Wen Sun for sharing the binary code of the AFISA algorithm [41], Dr. Yiyuan Wang for sharing the source code of the RedLS algorithm [47] and Pr. Bruno Nogueira for sharing the source code of the ILS-TS algorithm [33]. We would like to thank Daniel Rolland for proofreading this article. This work was granted access to the HPC resources of IDRIS (Grant No. 2020-A0090611887) from GENCI, which is also acknowledged.

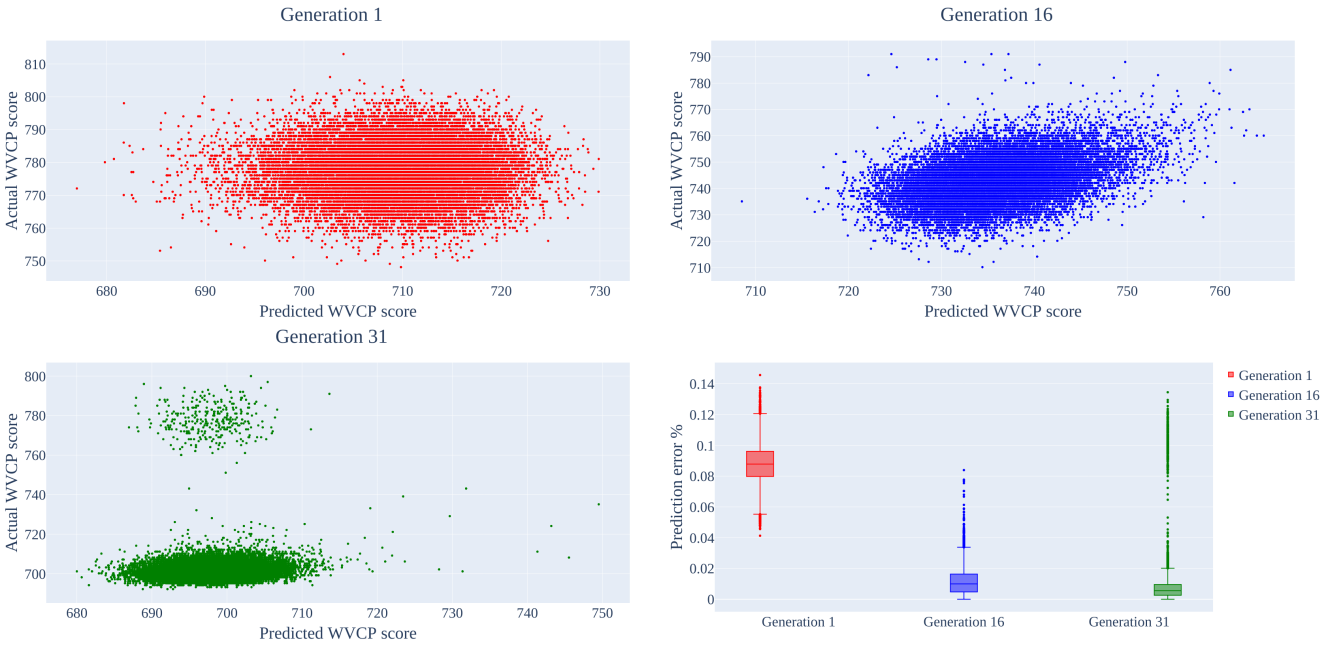

Figure 3: Comparison of the predicted and actual results of the local searches at generation 1, 16 and 31 for the instance DSJC500.5.col.

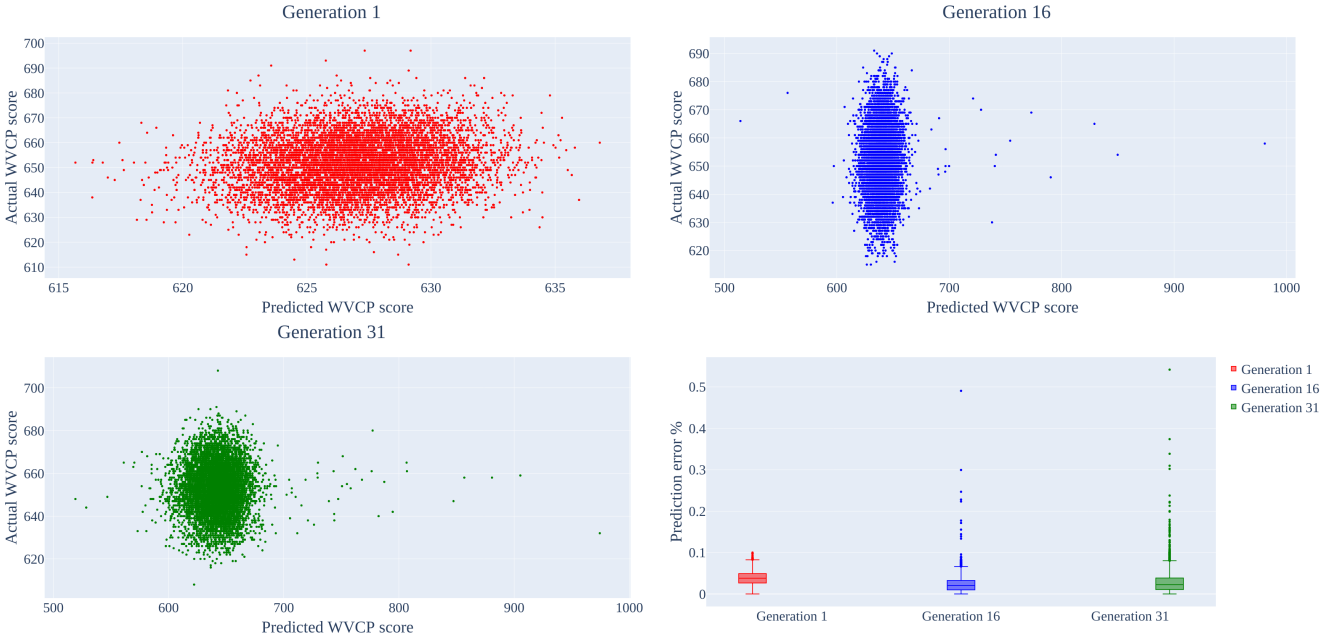

Figure 4: Comparison of the predicted and actual results of the local searches at generation 1, 16 and 31 for the instance wap05a.col.

References

[1] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 1996.

[2] Ivo Blöchliger and Nicolas Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975, 2008.

[3] Ivo Blöchliger and Nicolas Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975, 2008.

[4] Justin Boyan and Andrew W Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1(Nov):77–112, 2000.

[5] Marco Chiarandini, Thomas Stützle, et al. An application of iterated local search to graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, 2002.

[6] Denis Cornaz, Fabio Furini, and Enrico Malaguti. Solving vertex coloring problems as maximum weight stable set problems. *Discrete Applied Mathematics*, 217:151–162, 2017.

[7] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6351–6361, Red Hook, NY, USA, 2017. Curran Associates Inc.

[8] Raphaël Dorne and Jin-Kao Hao. Tabu search for graph coloring, t-colorings and set t-colorings. In *Meta-heuristics*, pages 77–92. Springer, 1999.

[9] Fabio Furini and Enrico Malaguti. Exact weighted vertex coloring via branch-

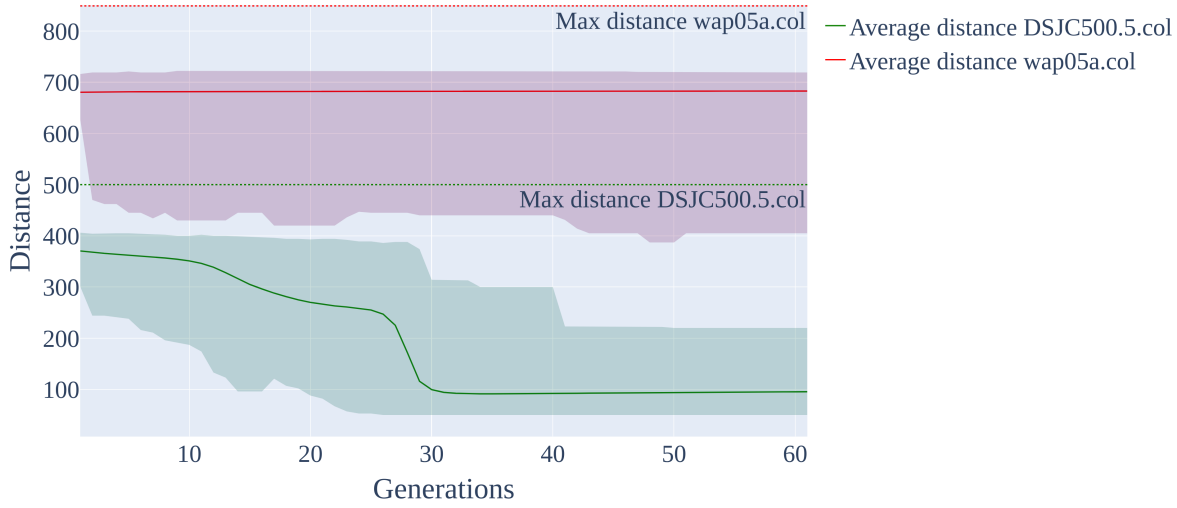

Figure 5: Red and green solid lines correspond to the average of the distance between the individuals of the population for the instances wap05a.col and DSJC500.5.col. Green and red areas are delimited by the maximum and minimum values of the distances between all the individuals in the population. The distance is an approximation of the set-theoretic partition distance (see Section 2.4).

and-price. *Discrete Optimization*, 9(2):130–136, 2012.

[10] 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.

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

[12] Philippe Galinier, Alain Hertz, and Nicolas Zufferey. An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2):267–279, 2008.

[13] Haris Gavranovic and Gerd Finke. Graph partitioning and set covering for the optimal design of a production system in the metal industry. *IFAC Proceedings Volumes*, 33(17):603–608, 2000.

[14] Olivier Goudet, Béatrice Duval, and Jin-Kao Hao. Population-based gradient descent weight learning for graph coloring problems. *Knowledge-Based Systems*, 212:106581, 2021.

[15] Olivier Goudet and Jin-Kao Hao. Massively parallel hybrid search for the partial latin square extension problem. *arXiv preprint arXiv:2103.10453*, 2021.

[16] Cyril Grelier, Olivier Goudet, and Jin-Kao Hao. On monte carlo tree search for weighted vertex coloring. In Leslie Pérez-Cáceres and Sébatien Verel, editors, *Evolutionary Computation in Combinatorial Optimization - 22nd European Conference, EvoCOP 2022, Madrid, Spain, April 20-22, 2022. Proceedings*, Lecture Notes in Computer Science, pages 1–16. Springer, 2022.

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

[18] Alain Hertz, Matthieu Plumettaz, and Nicolas Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008.

[19] Dorit S Hochbaum and Dan Landy. Scheduling semiconductor burn-in operations to minimize total flowtime. *Operations Research*, 45(6):874–885, 1997.

[20] Jiayi Huang, Mostofa Patwary, and Gregory Diamos. Coloring big graphs with AlphagoZero. *arXiv preprint arXiv:1902.10162*, 2019.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015.

[22] Yan Jin and Jin-Kao Hao. Solving the latin square completion problem by memetic graph coloring. *IEEE Transactions on Evolutionary Computation*, 23(6):1015–1028, 2019.

[23] Yan Jin, Jin-Kao Hao, and Jean-Philippe Hamiez. A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research*, 43:318–327, 2014.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations (Poster) 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

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

[26] Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 879–885. IEEE, 2019.

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

[28] Thomas Lucas, Corentin Tallec, Yann Ollivier, and Jakob Verbeek. Mixed batches and symmetric discriminators for gan training. In *International Conference on Machine Learning*, pages 2844–2853. PMLR, 2018.

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

[30] Enrico Malaguti, Michele Monaci, and Paolo Toth. Models and heuristic algorithms for a weighted vertex coloring problem. *Journal of Heuristics*, 15(5):503–526, 2009.

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

[32] Ferrante Neri, Carlos Cotta, and Pablo Moscato, editors. *Handbook of Memetic Algorithms*, volume 379 of *Studies in Computational Intelligence*. Springer, 2012.

[33] Bruno Nogueira, Eduardo Tavares, and Paulo Maciel. Iterated local search with tabu search for the weighted vertex coloring problem. *Computers & Operations Research*, 125:105087, 2021.

[34] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. Diversity control and multi-parent recombination for evolutionary graph coloring algorithms. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 121–132. Springer, 2009.

[35] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers & Operations Research*, 37(10):1822–1832, 2010.

[36] Daniel Cosmin Porumbel, Jin Kao Hao, and Pascale Kuntz. An efficient algorithm for computing the distance between close partitions. *Discrete Applied Mathematics*, 159(1):53–59, 2011.

[37] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. Spacing memetic algorithms. In Natalio Krasnogor and Pier Luca Lanzi, editors, *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 1061–1068. ACM, 2011.

[38] Marcelo Prais and Celso C Ribeiro. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing*, 12(3):164–176, 2000.

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

[40] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[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] Wen Sun, Jin-Kao Hao, Xiangjing Lai, and Qinghua Wu. Adaptive feasible and infeasible tabu search for weighted vertex coloring. *Information Sciences*, 466:203–219, 2018.

[43] Olawale Titiloye and Alan Crispin. Graph coloring with a distributed hybrid quantum annealing algorithm. In *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pages 553–562. Springer, 2011.

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

[45] Olawale Titiloye and Alan Crispin. Parameter tuning patterns for random graph coloring with quantum annealing. *PloS one*, 7(11):e50060, 2012.

[46] Qi Wang and Chunlei Tang. Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Knowledge-Based Systems*, page 107526, 2021.

[47] Yiyuan Wang, Shaowei Cai, Shiwei Pan, Ximing Li, and Monghao Yin. Reduction and local search for weighted graph coloring problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2433–2441, 2020.

[48] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[49] 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.

Appendix A. Results of DLMCOL on easy instances of vertex coloring problem

This appendix (Table A.9) reports the computational results reached by the DLMCOL algorithm on the easy set of the DIMACS challenge benchmark graphs. As Table A.9 shows, DLMCOL can consistently and easily reach the chromatic number $\chi(G)$ or the best known result k^* for each instance.

Table A.9: Computational results of DLMCOL on the *easy* DIMACS challenge benchmarks for the COL problem

					DLMCOL		
Instance	$\|V\|$	dens.	$\chi(G)$	k^*	k_{best}	SR	t(s)
DSJC125.1	125	0.1	5	5	**5**	10/10	19
DSJC125.5	125	0.5	17	17	**17**	10/10	27
DSJC125.9	125	0.5	44	44	**44**	10/10	33
DSJC250.1	250	0.1	?	8	**8**	10/10	36
DSJC250.9	250	0.9	72	72	**72**	10/10	313
r125.1	125	0.03	5	5	**5**	10/10	15
r125.1c	125	0.97	46	46	**46**	10/10	32
r125.5	125	0.5	36	36	**36**	10/10	322
r250.1	250	0.03	8	8	**8**	10/10	23
r250.1c	250	0.97	64	64	**64**	10/10	73
DSJR500.1	500	0.03	12	12	**12**	10/10	64
r1000.1	1000	0.03	20	20	**20**	10/10	258
le450_5a	450	0.06	5	5	**5**	10/10	69
le450_5b	450	0.06	5	5	**5**	10/10	76
le450_5c	450	0.10	5	5	**5**	10/10	76
le450_5d	450	0.10	5	5	**5**	10/10	69
le450_15a	450	0.08	15	15	**15**	10/10	95
le450_15b	450	0.08	15	15	**15**	10/10	93
le450_15c	450	0.17	15	15	**15**	10/10	143
le450_15d	450	0.17	15	15	**15**	10/10	314
le450_25a	450	0.08	25	25	**25**	10/10	58
le450_25b	450	0.08	25	25	**25**	10/10	55
school1	385	0.26	14	14	**14**	10/10	73
school1_nsh	352	0.24	14	14	**14**	10/10	59
flat300_20_0	300	0.48	20	20	**20**	10/10	56