KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science and Engineering

First cycle, 15 credits

Reinforcement learning for improved local search

Applied to the graph coloring problem

ADRIAN SALAMON
KLARA SANDSTRÖM

Stockholm, Sweden, 2023

Reinforcement learning for improved local search

Applied to the graph coloring problem

ADRIAN SALAMON
KLARA SANDSTRÖM

Degree Project in Computer Science, DA150X
Date: June 9, 2023
Supervisor: Stefano Markidis
Examiner: Pawel Herman
School of Electrical Engineering and Computer Science
Swedish title: Förstärkningsinlärning för att förbättra lokalsökning
Swedish subtitle: Applicerat på graffärgningsproblemet

Abstract

The graph coloring problem (GCP) is an important combinatorial optimization problem (COP) with various applications and a simple formulation: to assign colors to vertices in a graph such that no adjacent vertices share a color. The GCP is NP-hard, and in order to solve it within a reasonable time frame, heuristic local search (LS) based algorithms are commonly used. This study evaluates to what extent a LS algorithm for solving the GCP can be improved by using reinforcement learning (RL). This was achieved by designing and implementing an algorithm, named RLTCol, that combines the popular LS based TabuCol algorithm with RL. Our algorithm was evaluated against several state-of-the-art GCP algorithms as well as a variant of the algorithm that only uses LS. The results show that RL improved the performance of the LS algorithm, and that the RLTCol competed favorably against other LS based methods. Despite the simple RL policy that was used, the RL agent managed to generalize well and was able to solve many simple instances of the GCP on its own. This shows promise in the usefulness and ability of RL in solving complex COPs.

Sammanfattning

På grund av dess många tillämpningar är graffärgning ett viktigt kombinatoriskt optimeringsproblem. Problemet består i att tilldela färger till noder i en graf så att inga närliggande noder har samma färg. Graffärgning är NP-svårt och därför har olika heuristiska lokalsökningsalgoritmer utvecklats för att kunna lösa problemet inom rimlig tid. I den här studien undersöks i vilken utsträckning en lokalsökningsalgoritm för att lösa graffärgningsproblemet kan förbättras med hjälp av förstärkningsinlärning. I detta syfte designades och implementerades en ny algoritm vid namn RLTCol. Algoritmen kombinerar den populära lokalsökningsalgoritmen TabuCol med förstärkningsinlärning. RLTCol jämfördes med flera av de bästa algoritmerna för att lösa graffärgningsproblemet, samt med en version av algoritmen utan förstärkningsinlärning. Resultatet visade att förstärkningsinlärning förbättrade lokalsökningsalgoritmens prestanda, och höll samma standard som andra lokalsökningsbaserade algoritmer i litteraturen. Trots modellens enkla utformning lyckades förstärkningsinlärningsagenten lösa många enkla probleminstanser på egen hand och generaliserades dessutom bra. Detta visar på potentialen hos förstärkningsinlärning för att lösa komplexa kombinatoriska optimeringsproblem.

Contents

Nomenclature

Acronyms

COP	Combinatorial optimization problem
GCP	Graph coloring problem
LS	Local search
MDP	Markov decision process
MLP	Multilayer perceptron
ML	Machine learning
PPO	Proximal policy optimization
RLTCol	Reinforcement learning driven TabuCol
RL	Reinforcement learning
SA	Simulated annealing
TSP	Traveling salesman problem
TS	Tabu search

Chapter 1

Introduction

Combinatorial optimization problems (COPs) are a category of problems where the aim is to find a minima or maxima of an objective function over a discrete domain. Examples of well known COPs are the traveling salesman problem (TSP), the knapsack problem, and the graph coloring problem (GCP) [1]. COPs have applications in a broad range of fields, spanning from network design, task scheduling, and transportation system planning. COPs are in general NP-hard, which means that it is infeasible to find optimal solutions. Instead, heuristic approaches for finding good but not necessarily optimal solutions are used to solve large instances of complex COPs. One common heuristic approach is local search (LS). LS generally works by starting with a suboptimal solution that is iteratively improved through perturbation in order to find locally optimal solutions. LS has been employed successfully for many problems [2]. However, LS often gets trapped in a local optima as opposed to finding the global optima. To combat this, meta-heuristic algorithms are often used.

Meta-heuristic algorithms build on top of simpler heuristics such as LS and usually work by trying to encourage more exploration in the search process. Examples of meta-heuristic algorithms are simulated annealing (SA), tabu search (TS), ant colony optimization, and population-based memetic approaches [3]. One interesting area of research is trying to combine traditional meta-heuristic approaches with techniques from machine learning (ML). Some examples include learning an evaluation function to predict the outcome of a local search [4], and building a model that can generate better initial solutions to seed the algorithm with [5]. In addition, using techniques from reinforcement learning (RL) in conjunction with traditional meta-heuristics has showed promising results for solving COPs [5–8].

RL is one of the main sub-fields of ML, and broadly consists of making agents learn by interacting with an environment. By giving an RL agent rewards based on the actions that they take, the agent will try to learn what actions to take in order to maximise the cumulative reward it receives. Some general frameworks for solving COPs using learning methods have been proposed. Zhou et al. developed a learning-based method that uses learning automata for solving grouping problems [8]. Correia et al. developed a method called neural simulated annealing (Neural SA) which integrates an RL agent as part of the traditional SA algorithm [9]. Cai et al. devised a general framework called reinforcement learning based heuristic optimizaiton (RLHO) and used it to solve the TSP and bin-packing problem [5].

One particular COP that has seen extensive study is the GCP. The GCP can be used to model many real-world problems, such as scheduling and register allocation. The GCP consists of assigning colors to vertices in a graph such that no adjacent vertices share the same color. Many approaches for solving the GCP have been studied and proposed. They include TS [10], SA [11], population-based memetic algorithms [12], quantum annealing [13], and many more. Many of the most effective state-of-the-art methods incorporate a LS procedure as part of the algorithm. More specifically, the TabuCol algorithm [10] has proved to be a very effective LS procedure and is used extensively as a sub-procedure in many algorithms [2]. Limited research of using RL methods for solving the GCP has been conducted. A method using probabilistic learning based on a simple learning automata in combination with TS has been developed [3].

In this thesis, we present an algorithm named RLTCol for solving the GCP that combines RL with a heuristic optimizer. It is based on the RLHO framework [5] and is inspired by the Neural SA framework [9]. We compare the performance of RLTCol on several of the DIMACS benchmark graphs against state-of-the-art algorithms [14]. We also compare the performance of our algorithm to a variant of the algorithm with the RL part of the algorithm disabled, in order to determine the usefulness of the RL component.

1.1 Research Question

To what extent does incorporating RL in a LS based heuristic algorithm improve the performance of the algorithm for solving the GCP?

Performance is defined as the quality of solutions obtained (least amount of colors used) within a given time-frame.

Chapter 2

Background

In this section we introduce the GCP and a LS based method for solving it, as well as present the basic principles of RL. We also provide a general overview of existing approaches for solving the GCP and previous research on using RL for solving COPs.

2.1 Combinatorial Optimization Problems

A COP consists of finding the minima (or maxima) of an objective function over a discrete domain. Some examples include the TSP, GCP, and the bin-packing problem. Because of their various applications COPs have been studied extensively [1]. In general, COPs are NP-hard which means that there is no known algorithm to solve these problems in polynomial time in the number of bits of the problem instance.

2.1.1 Graph Coloring Problem

The GCP is a known NP-hard COP with many applications, including but not limited to: scheduling, register allocation, pattern matching, timetabling, and frequency assignment. The problem can be formulated as assigning colors to vertices in a graph such that no adjacent vertices share the same color. In general we aim to solve the decision version of the problem, which entails determining if it is possible to color a graph G with k colors.

The problem can be mathematically defined as following: given a graph

$G = (V, E)$ and a k, find a partition of V into k subsets $g_1, g_2, \ldots, g_k$ so that:

$$\sum_{\{u,v\} \in E} \delta(u,v) = 0 \tag{2.1}$$

Where $\delta(u,v) = 1$ if $u, v \in g_i$, and otherwise $\delta(u,v) = 0$ [1]. The minimum number of colors needed for a proper coloring of a graph is called the chromatic number, which is usually represented by χ. The sum in the left-hand side of Equation 2.1 can be seen as the objective function to be minimized in the COP formulation.

2.2 Local Search

The GCP is known to be NP-hard, and it is therefore infeasible to find optimal solutions. Because of this, a range of algorithms have been developed in order to find acceptable solutions in a reasonable amount of time. One heuristic approach to this is LS, where solutions are iteratively perturbed in order to find locally optimal solutions.

More formally LS works by having a solution s, and then perturbing it into a new solution $s' \in N(s)$, where $N(s)$ defines the set of neighboring solutions to s. In the context of the GCP, we define $N(s)$ as the set of solutions obtained by changing the color of one vertex v to a new color g_i. This is known as the 1-opt neighborhood, as it only performs a single move. If the objective function $f(s')$ is better than $f(s)$, we move to the new solution. For the GCP, we use the left hand side of Equation 2.1 as the function f to be minimized.

LS has the disadvantage of getting stuck in a local minima, when $f(s) < f(s')$ for all $s' \in N(s)$. In this case there is no guarantee that s is the optimal solution. Different variations of LS have been developed in order to escape local minima, such as SA and TS. TS has proved particularly effective for solving the GCP [2].

2.2.1 Tabu Search

TS is a LS algorithm that utilizes a tabu list to avoid getting stuck in local minima. The tabu list contains recent moves $s \rightarrow s'$ that have been made. If a move is in the tabu list, the algorithm is not allowed to make that move again for some number of iterations. This is in order to encourage exploration and avoid getting stuck. As applied to the GCP, the classic TabuCol algorithm from 1987 has been developed [10].

2.3 Reinforcement Learning

RL is one out of three major sub fields within ML. As opposed to supervised and unsupervised learning which involve classifying and structuring data, RL aims to teach an agent to make good decisions under uncertainty. An RL agent interacts with an environment by taking actions. Based on these actions, the agent receives varying rewards. By recording what rewards previous actions yielded, the agent can learn. The goal of the agent is to maximise the cumulative reward that it can obtain. As such, a good RL agent will not only take an action which gives a large immediate reward, but will also consider actions with lower immediate rewards and greater future rewards. This balance between current and future rewards is known as the exploration/exploitation trade-off. The RL framework is very general, and the agent itself does not know anything intrinsic about the problem it is trying to solve. [15]

2.3.1 Markov Decision Processes

RL problems are mathematically formalized as Markov decision processes (MDPs). An MDP is defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, P, \gamma)$ and consists of states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, an immediate reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, a transition kernel $P : \mathcal{S} \times \mathcal{A} \to \mathbb{P}(\mathcal{S})$, and a discount factor $\gamma \in [0,1]$. We also define a stochastic policy $\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})$.

At each time step t in RL, the *agent* is presented with a state $s_t \in \mathcal{S}$ from the *environment*. Based on the state s_t, the agent computes a probability of choosing each action according to its policy π. Based on the transition kernel P, an action $a_t \in \mathcal{A}$ is chosen and performed which produces a new state s_{t+1} and yields a reward $r_t \in \mathbb{R}$. This process is illustrated in Figure 2.1.

The agent will interact with the environment for a sequence of K time steps resulting in a trajectory $\tau = s_0, a_0, s_1, a_1, \ldots, s_K$. This trajectory is a sample from the distribution $P(\tau|\pi) = p_0(s_0) \prod_{k=0}^{K-1} P(s_{k+1}|s_k, a_k)\pi(a_k|s_k)$,

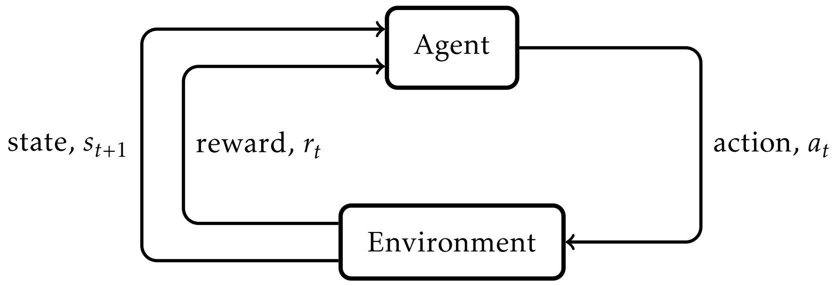

Figure 2.1: Markov Decision Processes. An agent interacts with an environment by taking actions which yield rewards and produce new states.

where $s_0 \sim p_0$ is sampled from the start distribution p_0. From this we can define the *discounted return* over a trajectory $R(\tau) = \sum_{k=0}^{K-1} \gamma^k r_k$, where $r_k = R(s_k, a_k, s_{k+1})$. The goal is to find an optimal policy π^* that maximises the expected return $\mathbb{E}_{\tau \sim P(\tau|\pi)}[R(\tau)]$.

2.3.2 Policy Gradient Methods

Policy gradient methods use a parameterized policy $\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})$. The parameters of the policy are updated directly based on the gradient of some performance measure. In general, neural networks are used as policy functions. Many different policy gradient methods have been conceived. In this paper we use proximal policy optimization (PPO), which employs several sophisticated methods that aim to reduce variance and improve stability [16].

2.4 Related Work

The following section contains an overview of previous work both in regards to methods for solving the GCP, and research into RL methods for solving COPs.

2.4.1 Graph Coloring Problem

Commonly used heuristic methods for solving the GCP broadly fall into four different categories: constructive approaches, LS, population-based hybrid approaches, and "reduce-and-solve" approaches. There are also some methods that do not fall into these categories.

- **Constructive approaches** generally work by iteratively constructing the solution vertex by vertex. Common constructive approaches are DSATUR and RLF. These methods generally produce solutions very fast, but often of poor quality [17, 18].
- **LS** starts from an initial solution which is iteratively improved by perturbation. LS has been implemented for many COPs and has yielded satisfactory results within a reasonable time frame [2]. For the GCP in particular, the TS algorithm known as TabuCol has become one of the most popular LS methods [10]. TabuCol is often used as a subprocedure in more advanced hybrid algorithms. LS methods are limited by the fact that they are unable to exploit global information, and perform worse than hybrid population-based algorithms.
- **Population-based hybrid approaches** generally work by improving multiple solutions in parallel. The solutions are manipulated by selection and recombination operators. The algorithms usually include mechanisms to preserve population diversity among the solutions. Hybrid algorithms have reported the best results on many difficult GCP instances. However, they have the disadvantage of being more complex both in design and implementation [19].
- **"Reduce-and-solve" approaches** work by first trying to reduce and simplify the graph, and then finding a proper coloring of the simpler graph. This can for example be done by identifying and removing independent sets from the graph. This approach is well suited for very large graphs, but performs worse for small and medium-sized graphs [20].

To our knowledge, only one learning-based method for solving the GCP has been developed. This method, developed by Zhou et al., is called probability learning and is based on learning automata [3]. It works by combining TabuCol with an updating probability matrix, and involves three phases: generating a solution based on the probability matrix, improving the solution using TabuCol, and updating the probability matrix based on the outcome of the TabuCol procedure. Using the probability matrix improved the performance of the algorithm, while still being a conceptually simpler method compared to hybrid algorithms.

2.4.2 Reinforcement Learning for Solving COPs

RL has been used in multiple ways for solving COPs, often in combination with a heuristic approach. A comprehensive survey of existing research into using RL for solving COPs has been done by Mazyavkina et al. They found that RL has been employed for solving several NP-hard COPs: the TSP, maximum cut problem, bin packing problem, minimum vertex cover, and maximum independent set [6]. Many of the algorithms developed exploit problem-specific features, which reduces the generality of these methods. However, some general frameworks that use RL to solve COPs have been proposed.

Cai et al. developed a framework named RLHO for solving COPs. It consists of an RL agent which passes initial solutions to a heuristic optimizer which returns a new solution and reward [5]. The RL agent uses both immediate rewards and the outcome of the heuristic optimization step as a training signal for the RL agent. In the paper, the RL agent is implemented using PPO and uses SA as the heuristic optimizer. They apply the RLHO framework to solve the the NP-hard bin packing problem. The method produced better results than using a pure heuristic optimizer.

With the aim of improving SA, Correia et al. incorporated an RL agent directly into the SA algorithm [9]. It works by letting the RL agent decide what neighboring solution to consider in each iteration of the algorithm. They named the method Neural SA. This method was applied and evaluated on four different COPs: Rosenbrock's function, the knapsack problem, the bin packing problem, and the TSP. They showed that Neural SA outperformed regular SA for all four COPs, which is particularly impressive given the lightweight neural networks employed for the RL policy.

Very limited research on using RL methods for solving the GCP has been conducted. It is therefore interesting to explore the benefits of combining effective RL methods with the powerful heuristic LS algorithm TabuCol, which will be evaluated in this thesis. Our method, named RLTCol, is mainly based on the RLHO framework [5], with our network model inspired by Neural SA [9].

Chapter 3

Method

In the following section we present our method for solving the GCP, named RLTCol. The general algorithm is first described, after which each component is explained in detail.

The algorithm consists of two components: a policy-based RL agent which passes initial solutions to a heuristic optimizer, and a heuristic optimizer that returns a new solution and reward to the agent after running for fixed number of iterations. The framework is illustrated in Figure 3.1. This is directly inspired by the RLHO framework by Cai et al. [5]. The framework can with some modification be applied to any COP.

The algorithm works by first randomly generating a solution, and then iteratively improving it. In each iteration, the RL agent runs for x steps, after which the last solution is passed to TabuCol. TabuCol then runs

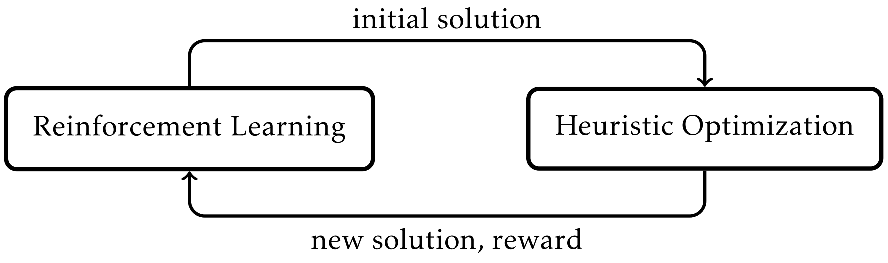

Figure 3.1: The RLTCol framework. An RL agent passes initial solutions to a heuristic optimizer that returns a new solution and reward.

for y iterations and returns either the best found or the last solution. The improvement of the solution obtained after TabuCol is also passed as an additional reward to the RL agent. The algorithm is described in pseudocode in Algorithm 1.

Algorithm 1: RLTCol

```
Input: A graph $G = (V, E)$, and a $k$
Output: The best solution $s^*$ found
1 $s, s^* \leftarrow$ random solution
2 for each iteration $1, 2, \ldots$ do
3     Run RL for $x$ steps, obtaining solution $s_x$
4     Pass $s_x$ to TabuCol, and run for $y$ steps to obtain $s_{x+y}$
5     Pass the reward $r_i$ as the difference in cost between $s_x$ and $s_{x+y}$
        to the RL agent.
6     if $f(s_{x+y}) \leq f(s^*)$ then
7         $s^* \leftarrow s_{x+y}$
8     end
9 end
```

3.1 Heuristic Optimizer

The heuristic optimization part of the framework consists of an implementation of TabuCol inspired by Dorne et al. [21]. The algorithm is detailed in Algorithm 2 and explained below.

In each iteration, the solution s is perturbed by a move $s \rightarrow s'$, where $s' \in N(s)$. As a heuristic, only vertices that are conflicting with at least one other vertex in its original class are considered when deciding which move to make. A move is defined by $\langle v, j \rangle \in V \times \{1, \ldots, k\}$, and at each iteration the best possible move is chosen. This is achieved using an incremental $|V|$-by-k matrix Δ, in which each entry $\Delta(x, j)$ represents the effect of changing the color of vertex x from i to j. This matrix is updated incrementally when a move is performed in $O(|V|k)$.

After a move is performed, the couple $\langle v, c(v) \rangle$ is classified as tabu for the next tl iterations, where $c(v)$ represents the color class of vertex v in s. This means that the vertex v cannot be assigned to its previous color class $c(v)$ during this period. A tabu move leading to a configuration

```
Algorithm 2: TabuCol
Input: A graph $G = (V, E)$, and initial solution $s_0$
Output: A solution
1  $s, s^* \leftarrow s_0$
2  for each iteration $1, 2, \ldots, y$ do
3      choose the best move $\langle v, j \rangle$
4      add $\langle v, c(v) \rangle$ in the Tabu list for $tl$ iterations
5      perform the move $\langle v, j \rangle$ in $s$
6      if $f(s) \leq f(s^*)$ then
7          $s^* \leftarrow s$
8      end
9  end
10 return $s^*$ with probability $1 - \beta$, otherwise $s$
```

better than the best configuration found so far is always accepted. The tabu tenure tl for a move is variable, and depends on the number of conflicting vertices in the current configuration, nb_{CFL}:

$$tl = Random(A) + \alpha \cdot nb_{CFL} \tag{3.1}$$

Where A and α are two parameters and the function $Random(A)$ returns a random integer in $\{0, \ldots, A-1\}$. We use parameters $\alpha = 1.2$ and $A = 12$, used previously by Zhou et al. [3]. In order to implement a tabu list, we can use a $V \times \{1 \ldots k\}$ table. Since we are running the algorithm many times in sequence, we found empirically that it was useful to introduce the parameter β that controls the probability of returning the best found solution s^* or the last found solution. We return s^* with probability $1 - \beta$, and the last solution s with probability β, where β is a parameter varied between $[0.05, 0.20]$ depending on the problem instance.

3.2 MDP Formulation

The following section describes the MDP formulation used in the RL component of RLTCol. As part of the MDP formulation, the following needs to be defined: the state space, action space, reward function, transition probability function, and discount factor.

A state $s = (x, \psi) \in \mathcal{S}$ consists of a solution x and a problem instance ψ.

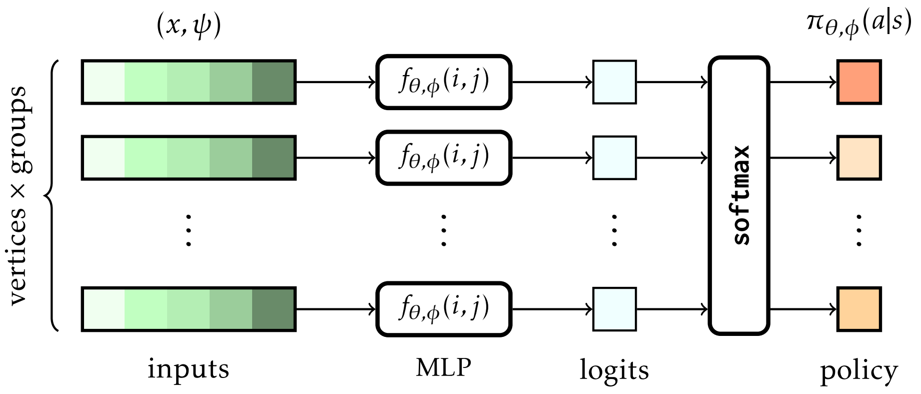

Figure 3.2: Policy network used for the RL agent. The same MLP is applied to all inputs independently. The logits are then made into a probability distribution via the softmax function.

The problem instance $\psi = (G, k)$ contains a graph G and the maximum number of colors k that can be used to color the graph. The solution $x = \{g_1, g_2, \ldots, g_k\}$ is a partitioning of the vertices in G into k groups, where each g_i is a set of vertices.

The action space $\mathcal{A}$ consists of actions $a \in \mathcal{A}$ which transition the current state into the next $s_t \longrightarrow s_{t+1}$. This is achieved by a perturbation of the solution $(x, \psi) \longrightarrow (x', \psi)$, where $x' \in \mathcal{N}(x)$ is a solution in the neighborhood of x. The neighborhood function $\mathcal{N}(x)$ returns all neighboring solutions to x where a single vertex in G is moved from from g_i to g_j.

As a reward function we use both the immediate reward and episodic rewards based on the outcome of the heuristic optimization. The immediate reward is given by $E((x_k, \psi)) - E((x_{k+1}, \psi))$ where $E((x, \psi))$ returns the number of conflicting edges in G given the solution x.

Since we use a stochastic policy, we can use a simple transition probability function that always accepts the sampled action. We use $\gamma = 0.99$ as our discount factor.

3.3 Policy

In addition to the MDP formulation, a stochastic policy $\pi : \mathcal{S} \longrightarrow \mathbb{P}(\mathcal{A})$ is defined. The policy is implemented as a simple neural network which is parameterized by θ and ϕ. The network is very small and has only 162 learnable parameters. A graphic overview of the architecture can be seen in Figure 3.2. For each vertex i and color j, we feed a feature representation of the current state $s = (x, \psi)$ into a feedforward multilayer perceptron (MLP). All vertices and colors are fed into identical MLPs to obtain logits that are then made into a probability distribution via the softmax function. This architecture can handle graphs of arbitrary size and number of colors, and each network evaluation can run in parallel.

The policy is divided into two parts: first a vertex i is selected, and then a group j to move the vertex to is chosen. The policy π can be written formally as:

$$\begin{aligned}
&\pi_{\theta,\phi}(a = (i,j)|s) = \pi_\theta(i|s)\pi_\phi(j|s,i), \text{where} \\
&\qquad\qquad \pi_\theta(i|s) = \text{softmax}(\mathbf{z}^{\text{vertex}}), \text{where } z_i^{\text{vertex}} = f_\theta(\langle \text{deg}_i, k_i, c_i \rangle), \\
&\qquad\quad \pi_\phi(j|s,i) = \text{softmax}(\mathbf{z}^{\text{group}}), \text{where } z_i^{\text{group}} = f_\phi(\langle n_i(g_j), n(g_j), |V| \rangle)
\end{aligned}$$

Where f_θ and f_ϕ are simple MLPs with one hidden layer. The MLPs have the structure $3 \rightarrow 16 \rightarrow 1$ and use ReLU activation functions. The input to f_θ is a tuple $\langle deg_i, k_i, c_i \rangle$ where the feature deg_i represents the degree of vertex i, k_i represents the number of conflicting edges that vertex i is connected to, and c_i represents number of different groups the adjacent vertices to vertex i belong to. The input to f_ϕ is a tuple $\langle n_i(g_j), n(g_j), |V| \rangle$ where $n_i(g_j)$ represents the number of vertices in group g_j adjacent to vertex i, $n(g_j)$ is the total number of vertices in group j, and $|V|$ is the total number of vertices in G.

Chapter 4

Results

This section is dedicated to experimental evaluation of the RLTCol algorithm presented previously. Firstly the learned policy is presented using simple examples. Then the performance of the algorithm is evaluated using the established DIMACS challenge instances [14]. Performance is defined as the quality of solution found, i.e. least amount of colors used, within a given time frame. The result is then compared against several established heuristic algorithms for solving the GCP. In order to evaluate the effectiveness of the RL component of RLTCol, a comparison is made to a variant of the algorithm named RLTCol$_0$ that does not use RL.

4.1 Policy

An illustration of the policy that was learned can be seen in Figure 4.1. The figure shows two RL iterations using the learned policy. The probability of choosing a vertex is displayed as the thickness of the border around the vertex, and the probabilities of choosing each color is shown for the sampled vertex. In this small example the agent manages to reduce the number of conflicts in the solution from 4 to 1. Here the policy is applied to a very simple graph, but from the illustration it is apparent that the RL agent was able to correctly identify good vertices to pick, as well as what colors to assign in order to find a proper coloring. In fact, the RL agent is able to find proper colorings of Mycielski graphs with $\chi = 4$ (such as in the figure) in 9 iterations on

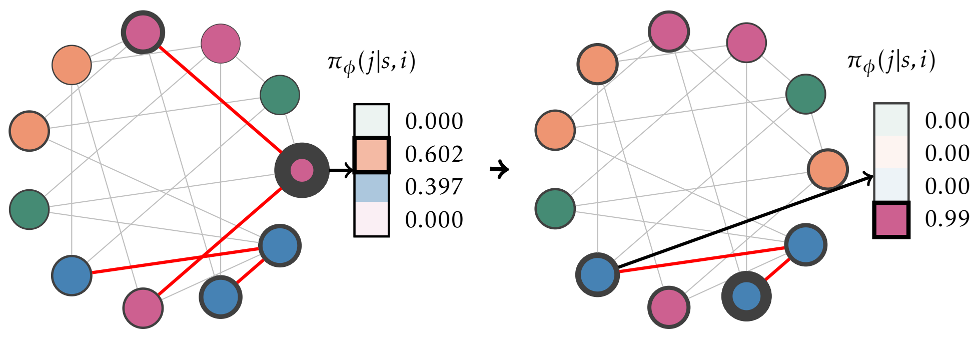

Figure 4.1: Illustration of two iterations of the learned policy, as applied to a random Mycielski graph with $\chi = 4$. Conflicting edges are representing by red colors. The thickness of the border around the vertices represents the probability $\pi_\theta(i|s)$ for each vertex i. The probability of choosing a color based on the vertex that was sampled, ie. $\pi_\phi(j|s,i)$, is shown in the bar right of each graph. The intensity of the color is scaled by $\pi_\phi(j|s,i)$.

average.

This shows that the RL agent is able to color simple graphs on its own. For example, it is able to find colorings of the graph `DSJC125.5` with $k = 18$, which is already better than constructive approaches such as DSATUR and RLF which yield colorings with 22 and 23 colors, respectively. This is particularly impressive since the agent itself has no knowledge of the structure of a graph or the problem it is trying to solve.

4.2 Benchmarking

The performance of RLTCol is assessed experimentally using the DIMACS challenge instances. These have been widely used in previous research for evaluating graph coloring algorithms. The particular instances on which the algorithm is tested belong to four different types:

- *Standard random graphs* named `DSJCn.x`, where n denotes the number of vertices and x is the density of of the graph. The density gives the probability of two vertices having a connecting edge.
- *Geometric random graphs* named `DSJRn.x`. They are generated by randomly choosing n points in a unit square which define the

vertices of the graphs. Two vertices are joined with an edge if the distance between them is less than x. Instances with the letter c denotes the complement of the geometric graph.

- *Leighton graphs* named `len_`χx have a density less than 0.25, have n vertices and have the chromatic number χ. The letter $x \in \{a, b, c, d\}$ indicates different graphs generated with different parameters.
- *"Quasi-random" flat graphs* named `flatn_`χ`_`δ. Here n refers to the number of vertices, χ is the chromatic number, and δ is a parameter for flatness, that gives the maximal allowed difference between the degrees of two vertices.

These graph instances can be crudely divided into "easy" and "difficult" graphs, according to the classification by Galiner et al. [22]. The graphs are said to be easy instances if they can be colored with k^* colors by a basic constructive coloring algorithm such as DSATUR [17], where k^* represents the number of colors in the best known solution. If only advanced algorithms are able to achieve a k^*-coloring, the instances are classified as difficult. Since the easy graphs are trivially solved by RLTCol, we only evaluate the performance of the algorithm on the difficult instances.

4.2.1 Experimental Settings

The RLTCol algorithm was implemented using a mix of Python and Rust. The RL portion of the algorithm was implemented using the Tianshou library [23] which is based on PyTorch, and the TabuCol algorithm was implemented in Rust. Our implementation of RLTCol is publicly available GitHub[1].

Due to time constraints the benchmarks were run on multiple computers with different specifications, albeit with similar performance. The majority of the tests were run on a system with an AMD Ryzen 7 3700X CPU running at 3.6 GHz, 32GB of RAM, and an Nvidia RTX 3060 Ti with 8GB of VRAM. The rest were run on Google Cloud instance with 8 CPU cores at 2.3 GHz, 32GB of RAM, and an NVIDIA T4 with 16GB of VRAM. The benchmarks for RLTCol$_0$ (without RL) were run on an equivalent computer without a GPU. Our benchmarks were run with a

[1]`https://github.com/adriansalamon/RLTCol`

Table 4.1: Comparative results of RLTCol and 9 state-of-the-art algorithms on difficult DIMACS graphs.

Instance	χ/k^*	Local search algorithms				Population-based algorithms					
		RLTCol	VSS	Partial	PLSCOL	HEA	MMT	Evo-Div	MA	QA	HEAD
		k_{best}	2008	2008	2018	1999	2008	2010	2010	2011	2015
DSJC250.5	?/28	**28**	*	*	**28**	*	**28**	*	**28**	**28**	**28**
DSJC500.1	?/12	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
DSJC500.5	?/47	48	48	48	48	48	48	48	48	48	**47**
DSJC500.9	?/126	**126**	**126**	127	**126**	**126**	127	**126**	**126**	**126**	**126**
DSJR500.1c	?/85	**85**	**85**	**85**	**85**	*	**85**	**85**	**85**	**85**	**85**
le450_15c	15/15	**15**	**15**	**15**	**15**	**15**	**15**	*	**15**	**15**	*
le450_15d	15/15	16	**15**	**15**	**15**	**15**	**15**	*	**15**	**15**	*
le450_25c	25/25	**25**	**25**	**25**	**25**	26	**25**	**25**	**25**	**25**	**25**
le450_25d	25/25	**25**	**25**	**25**	**25**	26	**25**	**25**	**25**	**25**	**25**
flat300_26_0	26/26	**26**	*	*	**26**	*	**26**	*	**26**	*	*
flat300_28_0	28/28	30	**28**	**28**	30	31	31	31	29	31	31
flat1000_76_0	76/81	86	85	87	86	83	83	82	82	82	**81**

5 hour time limit. In order to save time, we ran multiple tests on the same graph instance in parallel. This may marginally negatively affect the performance. Each instance was ran 10 times and was terminated when either a legal k-coloring was found, or a time limit of 5 hours was exceeded.

4.3 Comparison with state-of-the-art Algorithms

In the following section the RLTCol algorithm is compared to 9 state-of-the-art algorithms from literature. For completeness, we include both established LS and population-based hybrid algorithms. The algorithms from the literature were tested on different computer architectures and with different time constraints. Description of the algorithms, computer architectures, and what time limits were used to obtain the results are presented in Appendix A. Most of the algorithms were evaluated with time limits greater than 5 hours.

The best found colorings by all algorithms on the DIMACS instances is shown in Table 4.1. The left-most column contains the name of the graph instance and the second column shows the corresponding chromatic number χ (if known) and best known solution k^*. The third column contains the number of colors used in the best solution obtained by RLTCol over all runs, while the following columns show the results

Table 4.2: Comparative results of RLTCol and RLTCol$_0$ on difficult DIMACS graphs.

Instance	χ/k^*	RLTCol				RLTCol$_0$			
		k	#succ	#iters	time (s)	k	#succ	#iters	time (s)
DSJC250.5	?/28	**28**	10/10	$\underline{1.2\times10^6}$	54	**28**	10/10	1.8×10^6	97
DSJC500.1	?/12	**12**	10/10	$\underline{6.1\times10^7}$	1353	**12**	9/10	1.7×10^8	7429
DSJC500.5	?/47	**48**	5/10	1.7×10^8	12853	49	10/10	7.5×10^7	2006
DSJC500.9	?/126	**126**	1/10	4.3×10^7	5590	127	2/10	3.7×10^7	11743
le450_15c	15/15	**15**	1/10	9.3×10^6	696	16	10/10	6.3×10^6	250
le450_15d	15/15	16	10/10	6.5×10^6	259	**15**	1/10	2.3×10^6	953
le450_25c	25/25	**25**	1/10	3.4×10^7	10535	26	10/10	2.9×10^5	14
le450_25d	25/25	**25**	2/10	3.1×10^8	9046	**25**	1/10	$\underline{1.8\times10^8}$	13448
flat300_26_0	26/26	**26**	10/10	2.3×10^6	176	**26**	10/10	$\underline{1.4\times10^6}$	126
flat300_28_0	28/28	**30**	1/10	1.7×10^8	15437	**30**	2/10	$\underline{1.2\times10^8}$	6472
flat1000_76_0	76/81	**86**	2/10	3.4×10^7	10687	88	4/10	2.7×10^7	10723

of the reference algorithms. The lowest k for each graph is highlighted in bold.

As can be seen in Table 4.1 the RLTCol algorithm manages to find a solution equal to the best known coloring in 8 out of 12 of the tested instances. In the 4 other cases the algorithm found a coloring using just a few colors more than the best known solution. This means that the algorithm does not quite compete with all of the reference algorithms, but it holds the standard set by several of them and is not far behind the best known algorithms. RLTCol competes well against LS algorithms, but expectedly falls behind the state-of-the-art population-based algorithms. The algorithm is also somewhat inconsistent, in the sense that it only managed to find its best colorings on some of the instances in a few of the ten runs.

4.4 Effectiveness of RL Component

In order to evaluate the effectiveness of including RL in RLTCol, a comparison was made against the performance of an algorithm without RL, denoted RLTCol$_0$. RLTCol$_0$ is run in the same manner as RLTCol, except that it runs RL for 0 steps and instead repeatedly calls TabuCol. The two algorithms were compared both in terms of the number of colors used in the best solution k, and how long the experiment took in terms of number of iterations and time. Both algorithms were run 10 times with a 5 hour time limit for each graph instance.

The results are presented in Table 4.2. Here the two left-most columns describe the graph instance and the chromatic number χ (if known) and the best found solution k^*. For each algorithm, the following information is presented: the smallest number of colors used in the best run k, the number of runs that succeeded in coloring the graph with k colors (#succ), the average number of iterations for the successful runs (#iters), and the average time the successful runs took. The best found k between the two algorithm is presented in bold. If the algorithms managed to color the graph with the same k, the smallest number of iterations is underlined. Fewer iterations indicates a more effective algorithm.

As can be seen in Table 4.2, the RLTCol algorithm finds a better coloring for 5 out of 11 instances. These include the three largest instances that we tested on: `DSJC500.5`, `DSJC500.9`, and `flat1000_76_0`. The algorithms find equally good solutions for 5 instances, out of which 4 are in line with the best found k^*. RLTCol$_0$ only manages to find a better coloring for the graph `le450_15d`. On the instances where the algorithms find equally good solutions, the number of iterations used are similar. This indicates that both algorithms are similarly efficient.

For many instances, the best found solutions are obtained only in a few of the runs, and in this sense both algorithms are similarly inconsistent.

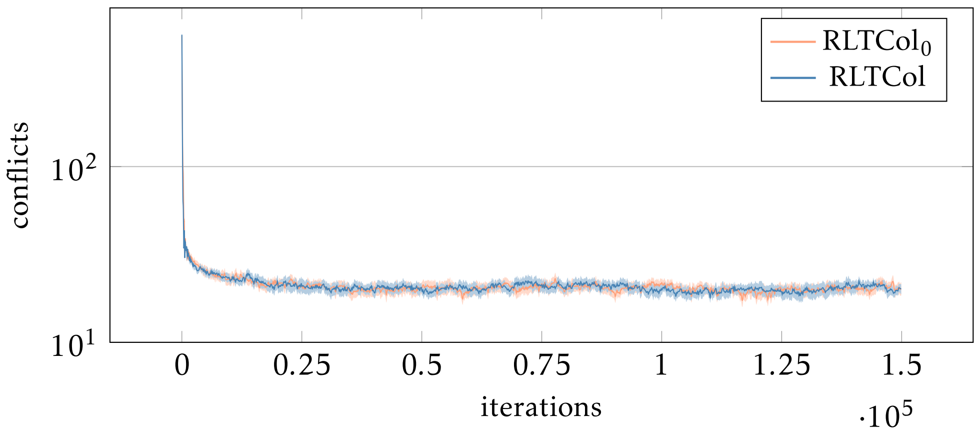

Figure 4.2: Convergence plot comparing RLTCol with RLTCol$_0$. Both algorithms were run trying to color the graph `DSJC250.5` with $k = 27$ (best known k is 28) for 150 000 iterations. The results are averaged over 25 runs for each algorithm.

This also shows that there is a high degree of variance in the performance of both algorithms. Even so, the extensive benchmarks indicates that RLTCol performs slightly better with the RL component enabled.

Figure 4.2 shows a convergence plot of RLTCol and RLTCol_0 on the graph `DSJC250.5` with $k = 27$ (the best known k is 28). The algorithms were both run 25 times for 150 000 iterations. The plot shows that both algorithms perform almost identically. The best runs produced equally good colorings with 6 conflicts. One thing to observe is that the most fluctuations in the number of conflicts can be attributed to the TabuCol procedure. The RL agent has the disadvantage of not having any notion of previous moves. As such, it will with a very low probability choose a move with a negative immediate reward, and therefore gets stuck in a local minima. In this case the agent either does not change the solution at all or flips back-and-forth between moves without affecting the number of conflicts in the solution.

Chapter 5

Discussion

This study has investigated to what extent RL can be used to improve the performance of a LS based heuristic algorithm. A framework for combining RL with the heuristic TabuCol algorithm was developed and evaluated. In this section the results and its implications will be discussed. Also, limitations in the methodology and potential future work that build upon the results from this thesis will be presented.

5.1 Results and Implications

The RL agent managed to learn a policy that was able to solve simple graph instances on its own, with better results than constructive algorithms. This is particularly impressive since the agent does not know anything intrinsic about the GCP. Notable is also the very simple policy network with only 162 learnable parameters that managed to achieve such good results. It is difficult to intuitively know if the learned policy is good at solving the given task, but for simple graphs such as in Figure 4.1 it is clear that the policy manages to prioritize between vertices and compute reasonable probabilities of colors to assign. Even if pure RL only manages to color simple graph instances, it is still impressive that an agent with a simple policy is able to solve such a complex problem.

Our algorithm RLTCol, which combines the RL component with TabuCol, achieved good results on several of the difficult DIMACS instances. The results also show that the RL component improves the performance of the TabuCol algorithm. This is illustrated by the fact that RLTCol managed to find better solutions than RLTCol$_0$ on 5 out of 11 of our benchmarks. First of all this shows that the RL component does not negatively affect the performance of the algorithm (which could be a possibility) but in fact slightly improves it. It is difficult to explain why this may be the case, but since the purpose of RL itself is to maximise rewards and optimally solve problems, it makes sense that RLTCol at least did not perform worse than the same algorithm without the RL component. The bulk of the work in RLTCol is done by the LS procedure, and as such it is not surprising that they perform very similarly. This can further be seen when investigating the convergence of the algorithms, as they appear almost identical.

RLTCol did not compete favorably with the reference state-of-the-art algorithms, in particular hybrid population-based algorithms. This can be attributed to several factors, which are laid out below.

Firstly the structure of both components of RLTCol are very similar, in the sense that they work on the same level. Just like in TabuCol, our RL agent only performs simple perturbations to the solution, and has no knowledge of solutions beyond the neighborhood of the current solution. A majority of the state-of-the-art hybrid population-based heuristic algorithms also consist of two distinct components, but the components work on different "levels". For example, in HEAD, two solutions are computed with TabuCol in parallel, and a higher level operator (specifically GPX) combines part of the solutions to produce two new solutions that are fed back to the next iteration [24]. A similar principle is used in PLSCOL, where the outcome of a LS procedure affects a global probability matrix that is used as a starting point for the next LS iteration [20]. Both these algorithms perform better than RLTCol.

Furthermore, the fact that the RL agent does not have any "memory" likely hinders its performance. Since the agent is trained to obtain immediate rewards, it learns to only perform moves that reduce the number of conflicts in the solution. As such, the policy will with a very low probability perform a move that leads to a worse solution now but has the possibility of leading to a better solution in the future. This means that the RL agent is very likely to get trapped in local minima. It could be possible to develop a more sophisticated RL model where the policy may use information about previous moves. Another approach may be to give some penalization to the agent for staying in a similar configuration for too long.

In summary, the results show that incorporating RL in a LS based heuristic can improve the performance of the LS algorithm, and that there is potential of this technique. This positively reaffirms the usefulness of RL in aiding in solving very complex problems.

5.2 Limitations

The framework for training the RL agent and evaluating the model is dependent on many parameters. Our limited testing suggests that the performance of RLTCol heavily depends on the parameters used during both training and evaluation. These include but are not limited to: the number of RL steps x and TabuCol iterations y in each episode, the number of training episodes and epochs, and the parameter β which decides the greediness of TabuCol. The PPO method for training the agent also has a large number of parameters that can be tuned. The value of the parameters for training and testing were decided based on literature and limited experimental testing. If given more time the parameters could be tuned more effectively and perhaps improve the results.

5.3 Future Work

The RL component of the RLTCol algorithm slightly improved the results of the heuristic TabuCol. It would be interesting to modify the framework and apply it to other COPs in order to further explore the benefits. Since the RL method used is conceptually simple and does not depend on problem specific features, the framework could easily be generalized to solve other problems than the GCP. If the framework proved useful in solving other COPs, it would further demonstrate the usefulness and applicability of RL in solving COPs.

Another area to explore would be trying to improve the performance of the RL component. In the MDP formulation used for the RLTCol algorithm, the agent makes decisions based on local information on the same level as the heuristic optimizer. The decisions are based only on features of individual vertices and color groups. The agent also has no notion of state or previous moves. An interesting path to explore would be embedding a more complex and higher level state and action space, as well as giving the RL agent some memory of previous moves. The agent might be able to learn a better policy if it also could base its decisions on structural information of the graph, such as symmetries and independent sets.

The best state-of-the art hybrid evolutionary algorithms use hand-crafted higher-level heuristics for combining solutions and preserving population diversity, and this could perhaps also be replaced with an RL agent. Another research direction that could be interesting and prove fruitful is to directly embed an RL component inside of the existing LS heuristic. Previous work by Correira et al. shows that this approach has potential [9].

Chapter 6

Conclusions

In this thesis we developed an algorithm, named RLTCol, for solving the GCP that involves a LS and a RL component. RLTCol works by iteratively running TabuCol and an RL policy. Our simple RL agent managed to find proper colorings on simple graph instances on its own, and performed well within our algorithm framework when combined with LS. We evaluated the performance of RLTCol on 12 of the difficult DIMACS graph instances, on which it achieved good results. Our algorithm did not manage to compete with state-of-the-art hybrid population based algorithms, but compared well with other LS based algorithms. The effectiveness of the RL component was evaluated by comparing RLTCol against a version of the algorithm with RL disabled. The results showed that the RL component slightly improved the performance of the algorithm, as it was able to find better or equal colorings on a 10 out of 11 of the graph instances that we tested on. This reinforces the potential of using RL as a technique for improving traditional heuristic algorithms for solving COPs such as the GCP. A more advanced RL agent that works on a higher level than in RLTCol might produce even better results, which could prove to be a promising research area.

Chapter 7

Bibliography

[1] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, en. John Wiley & Sons, Sep. 2011, ISBN: 978-1-118-03139-1.

[2] P. Galinier and A. Hertz, "A survey of local search methods for graph coloring", en, *Computers & Operations Research*, Part Special Issue: Anniversary Focused Issue of Computers & Operations Research on Tabu Search, vol. 33, no. 9, pp. 2547–2562, Sep. 2006, ISSN: 0305-0548, DOI: 10.1016/j.cor.2005.07.028.

[3] Y. Zhou, B. Duval, and J.-K. Hao, "Improving probability learning based local search for graph coloring", eng, *Applied soft computing*, vol. 65, pp. 542–553, 2018, Publisher: Elsevier B.V, ISSN: 1568-4946, DOI: 10.1016/j.asoc.2018.01.027.

[4] J. Boyan and A. W. Moore, "Learning evaluation functions to improve optimization by local search", *The Journal of Machine Learning Research*, vol. 1, pp. 77–112, Sep. 2001, ISSN: 1532-4435, DOI: 10.1162/15324430152733124.

[5] Q. Cai, W. Hang, A. Mirhoseini, G. Tucker, J. Wang, and W. Wei, *Reinforcement Learning Driven Heuristic Optimization*, Jun. 2019, DOI: 10.48550/arXiv.1906.06639.

[6] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, *Reinforcement Learning for Combinatorial Optimization: A Survey*, en, Dec. 2020, [Online]. Available: http://arxiv.org/abs/2003.03600 (visited on 02/07/2023).

[7] F. Kosanoglu, M. Atmis, and H. H. Turan, "A deep reinforcement learning assisted simulated annealing algorithm for a mainte-

nance planning problem", en, *Annals of Operations Research*, Mar. 2022, ISSN: 1572-9338, DOI: 10.1007/s10479-022-04612-8.

[8] Y. Zhou, J.-K. Hao, and B. Duval, "Reinforcement learning based local search for grouping problems: A case study on graph coloring", en, *Expert Systems with Applications*, vol. 64, pp. 412–422, Dec. 2016, ISSN: 0957-4174, DOI: 10.1016/j.eswa.2016.07.047.

[9] A. H. C. Correia, D. E. Worrall, and R. Bondesan, *Neural Simulated Annealing*, en, Mar. 2022, [Online]. Available: http://arxiv.org/abs/2203.02201 (visited on 02/08/2023).

[10] A. Hertz and D. de Werra, "Using tabu search techniques for graph coloring", en, *Computing*, vol. 39, no. 4, pp. 345–351, Dec. 1987, ISSN: 0010-485X, 1436-5057, DOI: 10.1007/BF02239976.

[11] M. Chams, A. Hertz, and D. de Werra, "Some experiments with simulated annealing for coloring graphs", en, *European Journal of Operational Research*, Third EURO Summer Institute Special Issue Decision Making in an Uncertain World, vol. 32, no. 2, pp. 260–266, Nov. 1987, ISSN: 0377-2217, DOI: 10.1016/S0377-2217(87)80148-0.

[12] P. Galinier and J.-K. Hao, "Hybrid Evolutionary Algorithms for Graph Coloring", en, *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 379–397, Dec. 1999, ISSN: 1573-2886, DOI: 10.1023/A:1009823419804.

[13] O. Titiloye and A. Crispin, "Quantum annealing of the graph coloring problem", en, *Discrete Optimization*, vol. 8, no. 2, pp. 376–384, May 2011, ISSN: 1572-5286, DOI: 10.1016/j.disopt.2010.12.001.

[14] *DIMACS Graphs and Best Algorithms*, [Online]. Available: http://cedric.cnam.fr/~porumbed/graphs/ (visited on 02/07/2023).

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction* (Adaptive computation and machine learning), en. Cambridge, Mass: MIT Press, 1998, ISBN: 978-0-262-19398-6.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms*, Aug. 2017, DOI: 10.48550/arXiv.1707.06347.

[17] D. Br, "New methods to color the vertices of a graph", en, vol. 22, no. 4, 1979.

[18] F. Leighton, "A graph coloring algorithm for large scheduling problems", en, *Journal of Research of the National Bureau of Stan-*

dards, vol. 84, no. 6, p. 489, Nov. 1979, ISSN: 0160-1741, DOI: 10.6028/jres.084.024.

[19] Z. Lü and J.-K. Hao, "A memetic algorithm for graph coloring", en, *European Journal of Operational Research*, vol. 203, no. 1, pp. 241–250, May 2010, ISSN: 03772217, DOI: 10.1016/j.ejor.2009.07.016.

[20] J.-K. Hao and Q. Wu, "Improving the extraction and expansion method for large graph coloring", en, *Discrete Applied Mathematics*, vol. 160, no. 16, pp. 2397–2407, Nov. 2012, ISSN: 0166-218X, DOI: 10.1016/j.dam.2012.06.007.

[21] R. Dorne, J.-K. Hao, P. Scientifique, and G. Besse, "A New Genetic Local Search Algorithm for Graph Coloring", vol. 1498, Jan. 2000, ISSN: 978-3-540-65078-2, DOI: 10.1007/BFb0056916.

[22] P. Galinier, J.-P. Hamiez, J.-K. Hao, and D. Porumbel, "Recent Advances in Graph Vertex Coloring", en, in *Handbook of Optimization: From Classical to Modern Approach*, ser. Intelligent Systems Reference Library, I. Zelinka, V. Snášel, and A. Abraham, Eds., Berlin, Heidelberg: Springer, 2013, pp. 505–528, ISBN: 978-3-642-30504-7, DOI: 10.1007/978-3-642-30504-7_20.

[23] J. Weng *et al.*, *Tianshou: A Highly Modularized Deep Reinforcement Learning Library*, arXiv:2107.14171 [cs], Aug. 2022, DOI: 10.48550/arXiv.2107.14171.

[24] L. Moalic and A. Gondran, "The New Memetic Algorithm HEAD for Graph Coloring: An Easy Way for Managing Diversity", en, in *Evolutionary Computation in Combinatorial Optimization*, G. Ochoa and F. Chicano, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2015, pp. 173–183, ISBN: 978-3-319-16468-7, DOI: 10.1007/978-3-319-16468-7_15.

[25] A. Hertz, M. Plumettaz, and N. Zufferey, "Variable space search for graph coloring", en, *Discrete Applied Mathematics*, Fifth International Conference on Graphs and Optimization, vol. 156, no. 13, pp. 2551–2560, Jul. 2008, ISSN: 0166-218X, DOI: 10.1016/j.dam.2008.03.022.

[26] I. Blöchliger and N. Zufferey, "A graph coloring heuristic using partial solutions and a reactive tabu scheme", en, *Computers & Operations Research*, Part Special Issue: New Trends in Locational Analysis, vol. 35, no. 3, pp. 960–975, Mar. 2008, ISSN: 0305-0548, DOI: 10.1016/j.cor.2006.05.014.

[27] E. Malaguti, M. Monaci, and P. Toth, "A Metaheuristic Approach for the Vertex Coloring Problem", *INFORMS Journal on Computing*, vol. 20, no. 2, pp. 302–316, 2008, Publisher: INFORMS: Institute for Operations Research, ISSN: 10919856, DOI: 10.1287/ijoc.1070.0245.

[28] D. C. Porumbel, J.-K. Hao, and P. Kuntz, "An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring", en, *Computers & Operations Research*, vol. 37, no. 10, pp. 1822–1832, Oct. 2010, ISSN: 0305-0548, DOI: 10.1016/j.cor.2010.01.015.

Appendix A

Reference Algorithms

The algorithms used for comparison with RLTCol, as well as the computer architecture that they were evaluated on, and the time limit used are detailed below:

(1) Variable space search algorithm (VSS) [25] (a 2.0 GHz Pentium 4 processor and a cut off time of 10 h).
(2) Local search algorithm using partial solutions (Partial) [26] (a 2.0 GHz Pentium 4 and a time limit of 10 h together with a limit of 2×10^9 iterations without improvement).
(3) Probability learning based local search (PLSCOL) [3] (a 2.8 GHz Intel E5-2760 and a cut off time of 5 h).
(4) Hybrid evolutionary algorithm (HEA) [12] (the processor used is not available and the results were obtained with different parameter settings).
(5) Two-phase evolutionary algorithm (MMT) [27] (a 2.4 GHz Pentium processor and a cut off time of 6000 or 40,000 s).
(6) Evolutionary algorithm with diversity guarantee (Evo-Div) [28] (a 2.8 GHz Xeon processor and a cut off time of 12 h).
(7) Memetic algorithm (MA) [19] (a 3.4 GHz processor and a cut off time of 5 h).
(8) Distributed quantum annealing algorithm (QA) [13] (a 3.0 GHz Intel processor with 12 cores and a cut off time of 5 h).
(9) The newest parallel memetic algorithm (HEAD) [24] (a 3.1 GHz Intel Xeon processor with 4 cores and a cut off time of at least 3 h).

TRITA-EECS-EX- 2023:0000