Monte Carlo Tree Search with Adaptive Simulation: a Case Study on Weighted Vertex Coloring

Cyril Grelier, Olivier Goudet, and Jin-Kao Hao*

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
{cyril.grelier,olivier.goudet,jin-kao.hao}@univ-angers.fr

Abstract. This work presents a hyper-heuristic approach with online learning, which combines Monte Carlo Tree Search with multiple local search operators selected on the fly during the search. The impacts of different operator policies, including proportional bias, one-armed bandit, and deep learning methods, are investigated. Experiments on well-known benchmarks of the Weighted Vertex Coloring Problem are conducted to highlight the advantages and limitations of each dynamic selection strategy.

Keywords: Monte Carlo Tree Search · Local Search · Hyper-heuristic · Weighted Vertex Coloring · Learning-driven optimization.

1 Introduction

Given a graph $G = (V, E, w)$ with vertex set V, edge set $E = \{\{u, v\}, u, v \in V\}$, and a weight function $w : V \rightarrow \mathbb{R}^+$, assigning a positive weight $w(v)$ to each node in V, the goal of the Weighted Vertex Coloring Problem (WVCP) is to find a partition $S = \{V_1, \ldots, V_k\}$ of the vertex set V, into k independent subsets V_i (also called color classes or groups) such that the following function is minimized:

$$f(S) = \sum_{i=1}^{k} \underset{v \in V_i}{max}\ w(v). \tag{1}$$

A set V_i is an independent set if and only if $\forall u, v \in V_i, \{u, v\} \notin E$.

The WVCP generalizes the popular NP-hard graph coloring problem, which corresponds to solving this problem when all weights $w(v)$ are equal for $v \in V$, and is therefore itself NP-hard. The WVCP belongs to the larger family of grouping problems whose main task is to partition a set of elements into mutually disjoint subsets such that additional constraints and/or optimization objectives are satisfied. The WVCP has many practical applications such as matrix decomposition [23] and batch job scheduling in a multiprocessor environment [22].

In addition to the exact methods for the WVCP, such as [19,10,5], dedicated heuristics have been proposed in the literature recently, including local search

* Corresponding author

algorithms [20,23,26,28], and memetic algorithms [12]. These heuristics can provide approximate solutions of good quality for medium and large instances, which cannot be solved in a reasonable time by exact methods. However, given the difficulty of the WVCP, no simple heuristic is able to obtain the best-known results for all instances of the literature. This can be explained by the fact that these instances have different characteristics (average degree, degree distribution, weight distribution). Even by fine-tuning the hyperparameters or using an adaptive selection of these parameters (e.g. the algorithm AFISA [26]), it is difficult for a single heuristic to provide enough flexibility to solve all instances successfully.

One possible way to overcome this difficulty is to find on the fly the best heuristic to use during the search for a given instance, hyper-heuristics have been proposed in the literature for this purpose. Hyper-heuristics are search methods that explore the space formed by low-level heuristics, instead of the space of direct solutions [3]. We refer the reader to [21,3,7] for an overview of existing hyper-heuristics proposed in the literature to solve various combinatorial optimization problems. Specifically, hyper-heuristics have already been used to solve grouping problems, with applications to scheduling and graph coloring [24,9], but never to the WVCP to our knowledge.

Most hyper-heuristics used for grouping problems are single-point hyper-heuristics in the sense that they apply a chosen low-level heuristic to the current solution at each step of the search before deciding to accept or reject the newly created solution [9]. However, these iterative hyper-heuristics are prone to get stuck in local optima, especially when tackling graph coloring problems that are generally characterized by a rough fitness landscape [2]. This occurs when the hyper-heuristic does not incorporate efficient low-level components that can be chosen to diversify the search into another area of the search space. Moreover, most of the hyper-heuristics proposed in the literature use a high-level strategy taking into account the gain in fitness obtained by the different low-level components (e.g. [11]) but do not take into account the current state of the iteratively improved solution.

To overcome these two limitations, we propose a hyper-heuristic approach with online learning, which combines the Monte Carlo tree search (MCTS) framework proposed in [13] and a high-level learning heuristic selector that takes into account the raw state of the current solution to be improved during the simulation phase of MCTS. This MCTS framework can continuously learn new promising starting points for a local search heuristic, which could help an iterative hyper-heuristic to avoid local minima traps.

Note that two previous works [25,1] have also used an MCTS in a hyper-heuristic for combinatorial optimization problems. In [25], the search space of low-level heuristics is explored by an MCTS, which searches for the best sequence of low-level heuristics to apply to a given solution, but without guarantee of optimality. Unlike this approach, the MCTS used in this work is directly constructed to explore the tree of legal solutions. Therefore, it has the property of being able to provide a theoretical guarantee that an optimal solution can be found if enough time is given to the algorithm. In [1], an MCTS was used

to create initial sequences, which are improved in a separate second step by a hyper-heuristic using a wide variety of local moves, selected by a high-level strategy. In this work, we aim to push further the coupling between the selection by the MCTS and the choice of the low-level heuristic.

We summarize the contributions of the present paper as follows. First, we propose a continuous integration between the MCTS, which is used to discover new starting points and the high-level operator selection strategy, which takes as input this starting point to make its choice. The score obtained by a given low-level heuristic is used to update both the MCTS learning strategy and the operator selector. Secondly, we investigate the impact of different online learning strategies, including a neural network taking into account not only the past scores of the low-level components as in [6], but also the raw state of the initiating solution to be improved by a local search heuristic. This deep neural network is made invariant by permuting the color groups with the deep set architecture [29], which is a desirable property for handling the WVCP.

2 Related works on the WVCP

Most of the best heuristics for the WVCP are local search procedures and constructive algorithms. This section presents a review of these methods.

2.1 Local search algorithms

We described the four most effective local search heuristics for the WVCP: TW [13], AFISA [26], RedLS [28] and ILS-TS [20]. These four heuristics will constitute the set of low-level heuristics that we will manipulate in the hyper-heuristic proposed in this paper.

Legal tabu search. TabuWeight (TW) [13] is inspired by the classical TabuCol algorithm for the GCP [14]. TW explores the space of legal colorings. It uses the *one-move* operator (which displaces a vertex from its color group to another color group) without creating conflicts. The best move not forbidden by the tabu list is applied at each iteration. The move is then added to the tabu list and is forbidden for the next tt iterations, tt being a parameter called tabu tenure.

Adaptive feasible and infeasible tabu search. AFISA [26] is a tabu search algorithm using the *one-move* operator and explores both the legal and illegal search spaces. [1] AFISA uses an adaptive coefficient to oscillate between legal and illegal solutions during the search.

Local search with selection rules. RedLS [28] explores both the illegal and legal search spaces. It uses the configuration checking strategy [4] to avoid cycling in neighborhoods solutions with the *one-move* operator. When the solution is legal, RedLS moves all the heaviest vertices from a color group to other colors to lower the WVCP score. Then it resolves the conflicts with different selection rules.

[1] A solution is said illegal if two adjacent vertices share the same color.

Iterated local search with tabu search. ILSTS [20] explores both the legal and partial search spaces. From a complete solution, ILSTS iteratively performs two steps: (i) it removes the heaviest vertices from the several color groups and places them in the set of uncolored vertices; (ii) it minimizes the score $f(S)$ by applying different variants of the *one-move* operator and a *grenade* operator until the set of uncolored vertices becomes empty.

2.2 Constructive heuristics

Two constructive heuristics have been proposed for the WVCP. These heuristics start with an empty or a partial solution and then color the nodes one by one until a complete legal solution is obtained. They have been used in combination with local search heuristics.

Reactive Greedy Randomized Adaptive Search Procedure. RGRASP [23] is an algorithm that iterates over two steps. First, it initializes a solution with a greedy algorithm. Secondly, it improves the solution with a local search procedure. At each iteration, it randomly removes the color of a part of the vertices and starts again at the first step. The choice of the vertices to recolor is managed by an adaptive parameter that evolves according to the quality of the solution.

Monte Carlo Tree Search Algorithm. The MCTS algorithm proposed in [13] is a constructive heuristic for the WVCP, where a search tree is built to explore the partial and legal search space. The search tree is built incrementally and asymmetrically. For each iteration of the MCTS, a selection strategy balancing exploration and exploitation is used to construct a partial solution that is further completed and improved by a simulation strategy. To build this partial solution, the vertices of the graph are considered in a predefined order (they are sorted by decreasing order of their weight, then by degree) and each uncolored vertex $u \in U$ is assigned a particular color i. Such a move is denoted as $< u, U, V_i >$ and applying this move to a partial solution S being constructed gives a new solution $S \oplus < u, U, V_i >$. After the simulation phase, a legal solution is obtained and the search tree is updated with the WVCP score of this solution. In [13], the authors investigated the impact of various local search procedures used during this simulation phase of this MCTS algorithm, instead of the classical random simulation which performs badly for this problem. It appears from this study that the choice of the local search procedure has an important impact which depends on the type of instance considered.

To avoid catastrophic failure and to make the algorithm more robust to each specific instance, we propose to use this same MCTS framework, but to select dynamically the local search procedure during the search in a set $\mathcal{O} = \{o_1, \ldots, o_d\}$ of d different local search heuristics. In the proposed framework, the choice of the local search strategy may depend on the raw state of the initiating solution built by the MCTS at each iteration of the algorithm.

3 MCTS with adaptive simulation strategy

This section presents the general framework of the adaptive selection of local search operators combined with the MCTS algorithm presented in [13].

3.1 Main scheme

The proposed hyper-heuristic approach is detailed in Algorithm 1. It takes as input a weighted graph $G = (V, E, w)$, a set of local search operators $\mathcal{O} = \{o_1, \ldots, o_d\}$ and a high-level selection strategy $\pi_\theta : \Omega \rightarrow \mathcal{O}$ taking as input a complete and legal solution S in the feasible space Ω (Ω n'est pas defini, dont on precise ici sa nature.) and giving in output a local search operator $o \in \mathcal{O}$ to apply to S.

The selection strategy π_θ is parametrized by a vector of parameters θ initialized at random at the beginning of the search. Then the algorithm repeats a loop (iteration) until a cutoff time limit is met. Each iteration of the MCTS involves the execution of five steps:[2]

1. **Selection**: starting from the root node of the tree, the most promising children nodes are iteratively selected until a leaf node is reached. The selection of a child node at each level corresponds to the choice of a color for the next vertex of the graph[3]. The most promising node is selected based on an exploitation/exploration trade-off UCT (Upper Confidence bounds for Trees).
2. **Expansion**: the MCTS tree grows by adding a new child node to the leaf node reached during the selection phase.
3. **Adaptive Simulation**: the current partial solution is completed with greedy legal moves to obtain a complete solution S. Then the selected operator $o = \pi_\theta(S)$ is applied to the current solution S to improve it. This leads to a new solution S' and a new learning example (S, o, r), and the reward $r = -f(S')$ is stored in a database D. Every nb iterations, the selection strategy is learned online on this database D.
4. **Update**: after the simulation, the average score and the number of visits of each node on the explored branch are updated.
5. **Pruning**: if a new best score is found, some branches of the MCTS tree may be pruned if it is not possible to improve the best current score with it.

The fact that symmetries are cut in the search tree by restricting the set of legal moves considered during each step 1 and 2, and that pruning rules are applied in step 5, allows the whole tree to be explored in a reasonable time for small instances. This particularity of the algorithm allows to provide proof of optimality for such instances. We refer the reader to [13] for more details on

[2] One notices that compared to the MCTS method from [13], only the simulation phase (step 3) changes.

[3] The vertices are considered in the decreasing order of their weight then of their degree.

Algorithm 1 MCTS with adaptive simulation strategy

Input: Weighted graph $G = (V, E, w)$, $\mathcal{O}$ a set of local search operators and π_θ a selection strategy.
Output: The best legal coloring S^* found
$S^* = \emptyset$ and $f(S^*) = MaxInt$
$D = \emptyset$.

while stop condition is not met **do**
    $C \leftarrow R$ ▷ Current node corresponding to the root node of the tree
    $S \leftarrow \{V_1, U\}$ with $V_1 = \{v_1\}$ and $U = V \backslash V_1$ ▷ first vertex in first color group

    /* Step 1 - Selection */
    while C is not a leaf **do**
        $C \leftarrow$ select_best_child(C) with legal move $< u, U, V_i >$
        $S \leftarrow S \oplus < u, U, V_i >$
    end while

    /* Step 2 - Expansion */
    if C has a potential child, not yet open **then**
        $C \leftarrow$ open_first_child_not_open(C) with legal move $< u, U, V_i >$
        $S \leftarrow S \oplus < u, U, V_i >$
    end if

    /* Step 3 - Adaptive simulation strategy */
    $S \leftarrow$ greedy(S) ▷ **Complete current solution with a greedy algorithm**
    $o = \pi_\theta(S)$ ▷ **Select a local search operator**
    $S' \leftarrow o(S)$ ▷ **Improve the solution with the selected operator.**
    $D \leftarrow D \cup (S, o, -f(S'))$ ▷ **Store a learning example in the database.**
    $\theta \leftarrow learning(D, \theta)$ ▷ **Online learning of the adaptive selection strategy**

    /* Step 4 - Update */
    while $C \neq R$ **do**
        update(C,f(S'))
        $C \leftarrow$ parent(C)
    end while
    if $f(S') < f(S^*)$ **then**
        $S^* \leftarrow S'$
    end if

    /* Step 5 - Pruning */
    apply pruning rules

end while
return S^*

the different steps 1, 2, 4, and 5 of this algorithm. In what follows, only step 3, with the adaptive simulation strategy written in bold in Algorithm 1, will be

described in detail. The source code of the algorithm presented in this paper will be made publicly available at the url `https://github.com/xxxx`.

3.2 Adaptive simulation strategy framework

As shown in [13], no simulation strategy dominates the others for the WVCP, and some operators are more successful for some types of instances than others.

Dynamic operator selection. To choose the best possible local search operator during the search, a high-level strategy π_θ automatically selects an operator $o = \pi_\theta(S)$ in $\mathcal{O}$ to be applied to the current solution S. Note that the choice of this operator may depend on the solution S to be improved. The set $\mathcal{O}$ of low-level heuristic components considered for the simulation strategy are the four local search procedures presented in Section 2.1, namely TW, AFISA, RedLS, and ILS-TS. The different high-level strategies π_θ used in this work will be presented in Section 4.

Collecting learning examples. Even if some of these local search operators o make transitions between different search spaces, it always returns the best legal solution (with the smallest WVCP score) encountered during the search denoted as $S' = o(S)$. According to this new solution found, a new learning example (S, o, r), associated with the reward $r = -f(S')$, is stored in a database D. The reward r is negative because the WVCP is a minimization problem. One notices that in the literature on hyper-heuristics, the reward of a given operator is often proportional to $f(S) - f(S')$ (for a minimization problem), namely the difference between the score before and after the search (see for example [11] and [6]). However, in our algorithm, we experimentally found that is is better that this reward does not depend on the score of the solution S from which the local search starts as it may introduce some noise in the learning process. Indeed, the MCTS can produce solutions of different quality at each iteration and if the score of the solution S from which the local search starts is taken into account in the evaluation of the reward, operators starting from an already good solution (low $f(S)$ value) will be at a disadvantage compared to those starting from a bad solution (high $f(S)$ value), as it is comparatively easier to improve a solution of poor quality than a solution that is already of good quality.

Online learning of the high level operator selector. Every nb iterations of the MCTS, the policy π_θ is trained on the database D and the set of its parameters θ is updated. The database D is modeled as a queue of size N corresponding to the last N examples obtained during the past iterations. This queue of limited size avoids overloading the memory of the device, but also allows to better adapt to the potential variations of the operators' results, in case some operators are better at the beginning of the search than at the end.

4 Operator selectors

We investigate the impact of five different operator selection policies π_θ of different level of complexity: a deep neural network, three fitness-based criteria and one baseline random selector, where each operator has $\frac{1}{|\mathcal{O}|}$ chance to be selected.

4.1 Neural network selector

In this case, the function $\pi_\theta : \Omega \to \mathcal{O}$, is modeled by a deep neural network, parametrized by a vector θ (initialized at random at the beginning).

This neural network π_θ takes as input a coloring S as a set of k binary vectors $\mathbf{v_j}$ of size n, $S = \{\mathbf{v_1}, \ldots, \mathbf{v_k}\}$, where each $\mathbf{v_j}$ indicates the vertices belonging to the color group j. From such an entry the neural network outputs a vector with $|\mathcal{O}|$ values in $\mathbb{R}$ corresponding to the expected reward of each operator $o \in \mathcal{O}$. Then, with a 90% chance, the operator associated with the maximum expected reward is selected. We still keep a 10% chance to select an operator at random, to ensure a minimum of diversity in the operator selection (exploration).

Using the *deep set* architecture [29,18,12], the output of this neural network is made invariant by the permutation of the color groups in S, so that the neural network selector will take the same decision for two input colors S and S_σ which are equivalent up to color group permutation σ. Specifically, for any permutation σ of the input color groups we have

$$\pi_\theta(\mathbf{v_{\sigma(1)}}, \ldots, \mathbf{v_{\sigma(k)}}) = \pi_\theta(\mathbf{v_1}, \ldots, \mathbf{v_k}). \tag{2}$$

For a coloring $S = \{\mathbf{v_1}, \ldots, \mathbf{v_k}\}$, the color group invariant network π_θ is defined as

$$\pi_\theta(S) = \frac{1}{k} \sum_{i=1}^{k} (\phi_{\theta_P} \circ \phi_{\theta_{P-1}} \circ \cdots \circ \phi_{\theta_0}(S))_i, \tag{3}$$

where each ϕ_{θ_j} is a permutation invariant function from $\mathbb{R}^{k \times l_{j-1}}$ to $\mathbb{R}^{k \times l_j}$ with l_j being the layer sizes. Note that we have $l_{j-1} = |V|$ for the first layer and $l_{j-1} = |V|$ and $l_j = |\mathcal{O}|$ for the last layer. See [18] for more details on the permutation invariant function ϕ_{θ_j}.

In this work, we use a neural network with two hidden layers of size $h1 = |V|$ and $h2 = \frac{|V|+|\mathcal{O}|}{2}$, and a non-linear activation function defined as $\mu(x) = \max(0.2 \times x, x)$ (LeakyReLU).

During this training phase, each learning example (S, o, r) of the database D is converted into a supervised learning example (X, y), with X an input matrix of size $k \times |V|$ corresponding to the set of k vectors $S = \{\mathbf{v_1}, \ldots, \mathbf{v_k}\}$, and y is a real vector of size $|\mathcal{O}|$ (number of candidate operators), such that y is initialized to the value $\pi_\theta(X)$ for each operator and then its value $y[o]$ for the chosen operator o is replaced by the expected reward r: $y[o] = r$.

Every $nb = 10$ iterations, this neural network is trained with the average mean square error loss on the $|\mathcal{O}|$ outputs (supervised learning) on the dataset D during 15 epochs with Adam optimizer [16] and learning rate 0.001.

4.2 Classic fitness-based selectors

We compare the neural network selector above with three *basic* selectors, which do not take as input the raw solution to make their choice. These three selectors are two criteria based on a proportional bias (roulette wheel and pursuit) and one criteria based on the one-arm bandit method. These three criteria have extensively been used in the literature of hyperheuristics (e.g. [27,17,15,8,11]).

For these three operators, the reward r of each training example (S, o, r) is normalized between 0 and 1 over the sliding window consisting of the last N examples stored in the database D, 0 being the worst value obtained during the N last iterations (corresponding to the highest value of the fitness $f(S')$) and 1 being the best value. This normalized average reward computed on this sliding window and associated to an operator o is written r_o. n_o denotes the number of times the operator o have been selected during the last N iterations.

Adaptive roulette wheel and pursuit For these two probability based strategies inspired from [11], π_θ is a random procedure driven by a vector of $|\mathcal{O}|$ parameters $\theta = [p_1, \ldots, p_{|\mathcal{O}|}]$, such that p_i corresponds to the probability that the operator i is chosen this turn. We have $\sum_{i=1}^{|\mathcal{O}|} p_i = 1$. In the beginning, all the probabilities are set to equal value. Therefore, $p_i = \frac{1}{|\mathcal{O}|}$, for $i = 1, \ldots, |\mathcal{O}|$.

The more an operator o achieves good rewards r_o, the more its associated probability p_i increases and the more it is chosen in the following iterations.

Adaptive roulette wheel. In this strategy, the probabilities p_o for $o = 1, \ldots, |\mathcal{O}|$ are updated at each iteration according to the formula

$$p_o = p_{min} + (1 - |\mathcal{O}| * p_{min}) * \frac{r_o}{\sum_{o'}^{|O|} r_{o'}},$$

where p_{min} is the minimum selection probability for each operator (which is set to a strictly positive value to ensure a minimum level of exploration). It is a hyperparameter of the methods set to the value of $\frac{1}{5 \times |\mathcal{O}|}$ in this work.

Pursuit. In this strategy, the probabilities are updated during the learning process as

$$\begin{cases} p_{o*} = p_{o*} + \beta(p_{max} - p_{o*}) \\ p_o = p_o + \beta(p_{min} - p_o), \end{cases} \tag{4}$$

where $o*$ is the index of the best operator on the sliding window, $p_{min} = \frac{1}{5 \times |\mathcal{O}|}$, $p_{max} = 1 - (|O| - 1) * p_{min}$ and $\beta = 0.7$ is a coefficient introduced to control this winner-take-all strategy.

Note that unlike the adaptive roulette strategy, which gives more balanced chances to all low-level operators, the Pursuit strategy is more elitist as it gives more chances to the best strategy applied in the previous iterations.

One-arm bandit This policy π_θ is based on the one-arm bandit method. At each iteration, according to the UCB (Upper Confidence Bound) formula, the operator with the highest score $s_o = r_o + c * \sqrt{\frac{2*\log|D|}{n_o+1}}$ is selected, where c is a hyperparameter set to the value of 1.

5 Experimentation

This section presents experiments to assess the different adaptive simulation strategies.

5.1 Experimental settings and benchmark instances

We consider the 188 WVCP benchmark instances in the literature. These instances come from various problems, 35 pxx instances and 30 rxx instances from matrix decomposition [23] and 123 from DIMACS/COLOR competitions [26].

Among these instances, 124 have a known optimal score. Therefore, when evaluating a given method on such an instance, we stop it when the optimal score is reached. We perform 20 independent runs of each method to solve each instance on a computer equipped with Intel Xeon ES 2630, 2,66 GHz CPU. The time limit for each local search algorithm was one hour. For the combined methods, MCTS with local search, with and without adaptive selection, a time limit of one hour was set for local search, in order to compare all methods with the same overall time spent in the local search solvers. At each iteration of the MCTS algorithm, the time allowed in a local search (during the simulation phase) is set to $|V| * 0.02$ seconds.

The algorithm is coded in C++, compiled and optimized with a g++ 12.1 compiler. The source code is available at `https://github.com/xxx`.

In the following, when a method is said to be better than another on a given instance, it means that the difference between the average scores computed over 20 runs is in favor of the first method, and that this difference is significant (t-test with a p-value of 0.001).

5.2 Adaptive operator selection during the search

Figure 1 shows the mean of the cumulative selection of each operator with error bars during the 20 runs on four difficult instances for each criterion: Random, Roulette wheel, Pursuit, One-armed Bandit (UCB), and Neural network. The four local search algorithms that can be selected at each iteration of the MCTS (see Section 2.1) are TW (dash-dotted line in light blue), ILST-TS (solid line in red), RedLS (dashed line in green) and AFISA (dotted line in purple).

We first observe that during the first 20 iterations of the algorithm the choices made by the various high-level strategies are almost random. It corresponds to the time spent collecting enough examples to learn.

Figure 1 shows that it is not always the same strategy that is favored for the different instances. For example, the method TW is preferred for the instance

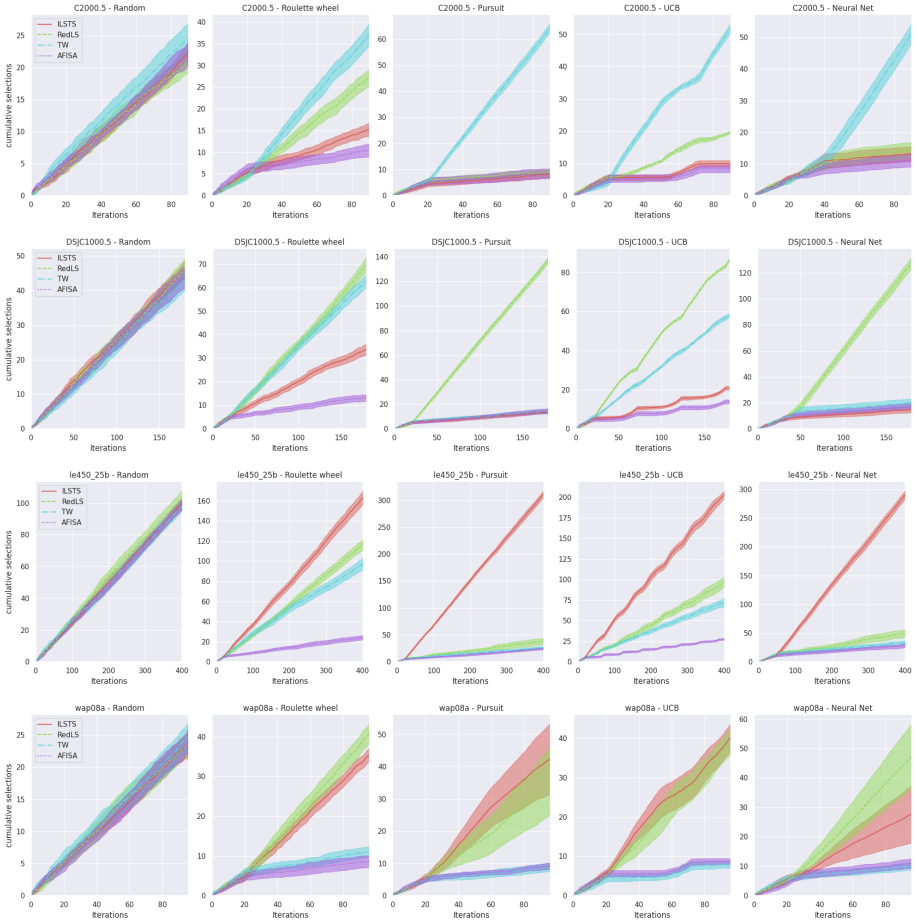

Fig. 1. Each plot represents the mean of the cumulative selection of operators for one criterion based on the 20 runs on one instance, with error bar. One line per instance (4 different instances), one column per adaptive method.

C2000.5, RedLS for DSJC1000.5 and ILSTS for le450_25b. The small error bars show that the selection is generally consistent over the different runs, except for wap08a for which the different strategies oscillate a lot between the ReldLS and ILSTS methods (which are the two best performing methods for this instance [28,20]) and thus for this instance the cumulative selection can vary a lot from one run to another. These curves highlight that the different operator selectors (except the Random selector) are in general able to identify the best performing method for each given instance.

One can observe different trends in the selection of operators depending on the policy. First, the random selection proceeds as expected and we observe that

the average cumulative selection numbers are almost equal for all operators at different time steps. Roulette Wheel still keeps a lot of diversity in the selection of the different operators but learns a bias toward the best methods during the search. The UCB strategy behaves much like the Roulette wheel strategy. Note that a lower exploration vs. exploitation coefficient would reduce the number of times the worst operators are selected. The Pursuit and Neural network strategies have similar behaviors: in general the best operators for a given instance are selected in priority, the other ones are picked randomly. These two criteria are more elitists than the others.

5.3 Performance comparisons on the different benchmark instances

Table 1 displays performance comparisons between each pair of methods on the 188 benchmark instances considered in this work. The different types of methods are (i) the four standalone local search solvers AFISA, TW, RedLS, and ILSTS; (ii) the MCTS versions with a fixed local search solver used for simulation: MCTS+AFISA, MCTS+TW, MCTS+RedLS, and MCTS+ILSTS; (iii) the MCTS versions with adaptive simulation strategies: Random, Roulette Wheel, Pursuit, UCB, and Neural network.

The MCTS combined with a local search solver improves the results of the local search alone except for ILSTS. Among the different MCTS+LS versions, MCTS+RedLS is the most efficient.

Looking at the adaptive selection strategy, we observe that the random strategy always performed the worst compared to the other adaptive strategies. This highlights the relevance of dynamically learning the best strategy for each given instance during the search.

When comparing the different adaptive strategies, it appears that the methods Roulette wheel and UCB obtained almost the same results, and are significantly better than the random strategy. The best performing strategies among all the compared strategies are Pursuit and Neural network. These methods are the most elitist. This indicates that it is beneficial to identify rapidly which method is efficient for each given instance and thus favor more intensification in the choice of the low-level heuristics to use at each iteration of the algorithm.

In general, although adaptive selection with the neural network performs well, a comparative advantage over the more basic Pursuit method does not appear in these experiments. There seems to be no obvious advantage to choosing an operator depending on the raw state of the solution from which the search begins. A simple fitness-based selection strategy can be just as effective as with more information in this context.

Table 5.3 displays more detailed results for several methods tested in this work. Column 1 indicates the name of the instance. Due to space limitation, we show here the results for a set of 20 instances (over 188) of different types: size of the graph $|V|$ ranging from 125 (instance DSJC125.5gb) to 2000 (instance C2000.5), degree ranging from 0.1 (instance DSJC250.1) to 0.9 (instance DSJC250.9), as well as different weight and degree distributions. Column 2 shows the best-known score (BKS) for each given instance reported in the literature.

Table 1. Comparison of all local search solvers and MCTS variants with and without adaptive selection. Each value corresponds to the number of instances where the row method is significantly better than the column method over the 188 benchmark instances considered (t-test with a p-value of 0.001). A number is written in bold if the number of instances is higher for the method in the row.

	AFISA	MCTS+AFISA	TW	MCTS+TW	RedLS	MCTS+RedLS	ILSTS	MCTS+ILSTS	Random	Roulette Wheel	Pursuit	UCB	Neural network
AFISA	-	38	**45**	16	39	0	9	14	0	0	0	0	0
MCTS+AFISA	**45**	-	**59**	8	**55**	0	0	0	0	0	0	0	0
TabuWeight	43	41	-	25	31	2	18	22	2	2	2	2	2
MCTS+TabuWeight	**73**	**73**	**86**	-	**71**	5	10	19	0	0	0	0	0
RedLS	**46**	43	**52**	30	-	14	22	25	15	15	15	15	15
MCTS+RedLS	**104**	**84**	**111**	**72**	**95**	-	**38**	**42**	**20**	4	2	4	4
ILSTS	**90**	**83**	**93**	**60**	**82**	15	-	**31**	3	4	4	2	3
MCTS+ILSTS	**84**	**82**	**90**	**52**	**78**	13	0	-	1	1	0	0	0
Random	**105**	**84**	**109**	**76**	**93**	13	**35**	**40**	-	0	0	0	1
Roulette Wheel	**105**	**84**	**112**	**79**	**95**	**14**	**38**	**41**	**5**	-	0	0	0
Pursuit	**105**	**84**	**112**	**79**	**95**	**16**	**38**	**41**	**12**	0	-	1	0
UCB	**105**	**84**	**112**	**79**	**94**	**14**	**37**	**42**	**5**	0	1	-	0
Neural network	**105**	**84**	**112**	**79**	**95**	**15**	**38**	**41**	**12**	**2**	0	0	-

A score in bold in this column indicates that the score has been proven optimal. Note that some of these best-known scores have been obtained under specific and relax conditions (computation time of more than one day and parallel computation on GPU device), while the results reported here are reached on a single CPU and with a time limit of one hour of local search. Column 3-26 display the results for one part of the different methods compared in this work. For the sake of space, the results of the versions AFISA, TW, MCTS+AFISA, MCTS+TW, and MCTS+ILSTS are not reported here. For each method are displayed the best and the average scores obtained over 20 runs, and the time spent in seconds to obtain the best scores.

The detailed results for all the 188 instances reached by all the different methods will be available at `https://github.com/xxx`. The last lines of this Table 5.3 summarize the results obtained by all methods for all the 188 instances: (1) "# BKS" is the number of times a method finds the best know score of the literature on each given instance, (2) "# best average" is the number of times the method gets the best average score compared to the other methods. We first observe that the two best local search solvers RedLS and ILSTS perform well but not for the same type of instance. More specifically, the RedLS algorithm is better for large instances such as DSJC1000.5 and C2000.5, while ILSTS is better for smaller instances (DSJC250.1, DSJC250.5, and rxx instances). This

shows that these methods can be complementary and that it may be beneficial to choose the right one on the fly for a given instance.

Secondly, we observe that the combination of the MCTS framework with e.g. the RedLS method (MCTS + RedLS version) improves the average results comparison to the RedLS solver alone for many instances, but not for the largest ones like DSJC1000.5 and C2000.5. This can be explained by the fact that the search space is very large for these two instances, and in this case it seems more advantageous to favor intensification in a limited area of the search space to get good results, rather than favoring more diversity with multiple restarts but with less time spent in different areas of the search space.

Thirdly, it appears that the two MCTS versions with adaptive simulation, such as Pursuit and Neural network, reach the largest number of BKS over the whole set of 188 instances, with good results for a wide variety of instances (such as DSJC250_X, DSJC500_X, le450_xx, and rxx instances).

instance	BKS	RedLS			MCTS+RedLS			ILSTS			Random			Roulette Wheel			Pursuit			UCB			Neural network		
		best	avg	t(s)	best	avg	t(s)	best	avg	t(s)	best	avg	t(s)	best	avg	t(s)	best	avg	t(s)	best	avg	t(s)	best	avg	t(s)
DSJC125.5gb	240	245	260.5	0	241	241	906.8	**240**	241	1160	**240**	240.9	2477	**240**	240.7	2042.1	**240**	**240.6**	2073.4	**240**	240.9	1666.5	**240**	**240.6**	2120.1
DSJC250.1	127	130	131.6	4.2	**127**	**127.8**	2187	130	131.9	1361.3	**127**	128.5	1494	**127**	128	2154	**127**	128	1830	**127**	128.2	1598	**127**	**127.8**	2418.8
DSJC250.5	392	398	401.4	248	**396**	**398.5**	1980	398	408.4	3591	398	400.2	2046	397	399.1	4050	397	399.2	894	**396**	399.1	822	397	398.7	1238
DSJC250.9	**934**	**934**	**935.8**	1039	935	936.8	144	936	944.9	3214	937	938.4	2988	935	937.9	3234	936	937.2	822	937	937.9	1932	936	937.4	2728
DSJC500.1	184	**187**	202.2	1448	188	188.9	777.3	199	202.2	2174	189	189.7	1133	**187**	188.9	3124	188	**188.8**	1727	**187**	189.1	539	188	**188.8**	2672.6
DSJR500.1	169	171	184.5	0	**169**	**169**	222	**169**	**169**	0.9	**169**	**169**	19.2	**169**	**169**	18.8	**169**	**169**	13.8	**169**	**169**	21	**169**	**169**	15.9
DSJC1000.5	1185	**1201**	**1215.7**	709	1275	1283	924	1349	1377.1	2922	1280	1292.8	2457	1276	1286	3171	1272	1283.6	903	1276	1285.7	2688	1275	1282.7	3671
C2000.5	2144	**2175**	**2196.7**	1624	2494	2508.8	2665	2459	2501.6	3365	2456	2463.8	1640	2439	2453.3	3567	2439	2453.1	1148	2438	2450.8	574	2441	2452.1	4529.5
GEOM120	**72**	**72**	75.2	0	**72**	**72**	12.4	**72**	**72**	0.1	**72**	**72**	5.5	**72**	**72**	4.2	**72**	**72**	4.5	**72**	**72**	4.3	**72**	**72**	3.7
le450_15a	212	214	236.3	604	**213**	**214**	2578.5	222	225.9	967	214	214.8	1188	**213**	214.4	126	**213**	**214**	1278	**213**	214.4	2043	**213**	**214**	1095.7
le450_15b	216	218	225.6	105	**217**	217.9	2700	225	227.7	2341.5	218	218.1	967.3	**217**	217.9	1395	**217**	**217.8**	2106	**217**	217.9	1953	**217**	**217.8**	1743.3
le450_25a	306	**306**	307.4	223.2	**306**	**306**	220.2	**306**	307.6	1743.7	**306**	306.3	771.2	**306**	306.1	663.5	**306**	**306**	590.1	**306**	**306**	710.1	**306**	**306**	1012.8
le450_25b	307	**307**	313.4	130.3	**307**	**307**	235.6	**307**	307.1	1444.1	**307**	307.1	688.9	**307**	**307**	610.8	**307**	307.1	609.4	**307**	**307**	988	**307**	**307**	771.5
p42	**2466**	**2466**	2522.5	0	**2466**	**2466**	109.5	**2466**	**2466**	16.2	**2466**	**2466**	111.5	**2466**	**2466**	76.2	**2466**	**2466**	62	**2466**	**2466**	55	**2466**	**2466**	63
queen15_15	223	227	229.9	1110	**225**	226.6	460	233	237.1	2493	**225**	227.4	480	226	226.9	2241	**225**	**226.4**	2196.2	226	227.2	1812.5	226	226.9	2203.8
r28	**9407**	9410	9563	126	**9407**	9427	1917.5	**9407**	**9407**	55.8	**9407**	**9407**	172.9	**9407**	**9407**	155.9	**9407**	**9407**	101.5	**9407**	**9407**	133.4	**9407**	**9407**	223.4
r29	**8693**	8696	8817.6	2	**8693**	**8693**	972.9	**8693**	**8693**	206.8	**8693**	8693.2	585.5	**8693**	**8693**	714	**8693**	**8693**	403.1	**8693**	**8693**	459.4	**8693**	**8693**	590.4
r30	**9816**	9836	9988.2	4	**9816**	9823	2827	**9816**	**9816**	20.1	**9816**	**9816**	79.6	**9816**	**9816**	78.8	**9816**	**9816**	82.2	**9816**	**9816**	90.5	**9816**	**9816**	112.5
wap07a	555	729	745.7	0	**574**	644.9	175	627	636.8	1942	**574**	**629.8**	1575	640	644	3290	637	643.2	1435	584	640	490	578	639.2	1129
wap08a	529	**537**	614	2257	549	**551.3**	2376	600	614.2	2563	551	554.2	432	551	553.3	1332	551	556.6	756	551	553.8	2124	551	553.5	3956.2
# BKS		112/188			153/188			151/188			154/188			155/188			156/188			156/188			157/188		
# best average		46/188			144/188			141/188			139/188			142/188			155/188			146/188			152/188		

Table 2. Part of the table reporting detailed results of the different methods for each instance of the benchmarks. For the sake of space, the results of the methods AFISA, TW, MCTS+AFISA, MCTS+TabuWeight, and MCTS+ILSTS are not reported here, and only the results for 20 instances over 188 are shown.

6 Conclusions

A Monte Carlo Tree Search framework with adaptive simulation strategy was presented and tested on the Weighted Graph Coloring Problem. Different high-level operator selectors have been introduced in this work.

The results show that the MCTS versions with adaptive operator selection reach the highest number of best-known scores for the set of 188 benchmark instances of the literature compared to the state-of-the-art methods in one hour of computation time on a CPU (except compared to the DLMCOL algorithm [12] which uses a GPU and was run with an extended computation time).

Analysis of the operator selections during the search for each particular instance shows that, in general, the choice of operators does not change during

the search. In fact, once the best solver for each instance has been identified, it is usually still chosen for the rest of the search. This lack of variation during the search may be explained by the fact that for a given instance, there is usually a dominant solver and we do not observe complementarity in the use of the different operators during the search for this problem. This may explain why the neural network selector taking into account the raw state of the current solution does not bring better results than a more basic fitness-based selector such as the Pursuit strategy.

A future work would be to introduce this new neural network operator selection strategy into a memetic algorithm for graph coloring problems where the different operators selected can be more complementary during the search (e.g. local search and crossover operators).

Acknowledgment

We would like to thank Dr. Wen Sun [26], Dr. Yiyuan Wang, [28] and Pr. Bruno Nogueira [20] for sharing their codes. This work was granted access to the HPC resources of IDRIS (Grant No. 2020-A0090611887, 2022-A0130611887) from GENCI and the Centre Régional de Calcul Intensif des Pays de la Loire (CCIPL).

References

1. Asta, S., Karapetyan, D., Kheiri, A., Özcan, E., Parkes, A.J.: Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. Information Sciences **373**, 476–498 (2016)
2. Bouziri, H., Mellouli, K., Talbi, E.G.: The k-coloring fitness landscape. Journal of Combinatorial Optimization **21**(3), 306–329 (2011)
3. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society **64**(12), 1695–1724 (2013)
4. Cai, S., Su, K., Sattar, A.: Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artificial Intelligence **175**(9), 1672–1696 (2011)
5. Cornaz, D., Furini, F., Malaguti, E.: Solving vertex coloring problems as maximum weight stable set problems. Discrete Applied Mathematics **217**, 151–162 (Jan 2017)
6. Dantas, A., Rego, A.F.d., Pozo, A.: Using deep q-network for selection hyper-heuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 1488–1492 (2021)
7. Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K.: Recent advances in selection hyper-heuristics. European Journal of Operational Research **285**(2), 405–428 (2020)
8. Eiben, A.E., Smit, S.K.: Evolutionary algorithm parameters and methods to tune them. In: Autonomous Search, pp. 15–36. Springer
9. Elhag, A., Özcan, E.: A grouping hyper-heuristic framework: Application on graph colouring. Expert Systems with Applications **42**(13), 5491–5507 (2015)
10. Furini, F., Malaguti, E.: Exact weighted vertex coloring via branch-and-price. Discrete Optimization **9**(2), 130–136 (2012)

11. Goëffon, A., Lardeux, F., Saubion, F.: Simulating non-stationary operators in search algorithms. Applied Soft Computing **38**, 257–268 (Jan 2016)
12. Goudet, O., Grelier, C., Hao, J.K.: A deep learning guided memetic framework for graph coloring problems. Knowledge-Based Systems **258**, 109986 (2022)
13. Grelier, C., Goudet, O., Hao, J.K.: On monte carlo tree search for weighted vertex coloring. In: Evolutionary Computation in Combinatorial Optimization. p. 1–16. Lecture Notes in Computer Science, Springer International Publishing, Cham (2022)
14. Hertz, A., Werra, D.: Werra, d.: Using tabu search techniques for graph coloring. computing 39, 345-351. Computing **39** (Dec 1987)
15. Hoos, H.H.: Automated algorithm configuration and parameter tuning. In: Autonomous Search, pp. 37–71. Springer
16. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
17. Lobo, F., Lima, C.F., Michalewicz, Z.: Parameter setting in evolutionary algorithms, vol. 54. Springer Science & Business Media (2007)
18. Lucas, T., Tallec, C., Ollivier, Y., Verbeek, J.: Mixed batches and symmetric discriminators for gan training. In: International Conference on Machine Learning. pp. 2844–2853 (2018)
19. Malaguti, E., Monaci, M., Toth, P.: Models and heuristic algorithms for a weighted vertex coloring problem. Journal of Heuristics **15**(5), 503–526 (2009)
20. Nogueira, B., Tavares, E., Maciel, P.: Iterated local search with tabu search for the weighted vertex coloring problem. Computers & Operations Research **125**, 105087 (Jan 2021)
21. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intelligent data analysis **12**(1), 3–23 (2008)
22. Pemmaraju, S.V., Raman, R.: Approximation algorithms for the max-coloring problem. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Automata, Languages and Programming. p. 1064–1075. Lecture Notes in Computer Science (2005)
23. Prais, M., Ribeiro, C.C.: Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. INFORMS Journal on Computing **12**(3), 164–176 (Aug 2000)
24. Sabar, N.R., Ayob, M., Qu, R., Kendall, G.: A graph coloring constructive hyper-heuristic for examination timetabling problems. Applied Intelligence **37**(1), 1–11 (2012)
25. Sabar, N.R., Kendall, G.: Population based monte carlo tree search hyper-heuristic for combinatorial optimization problems. Information Sciences **314**, 225–239 (2015)
26. Sun, W., Hao, J.K., Lai, X., Wu, Q.: Adaptive feasible and infeasible tabu search for weighted vertex coloring. Information Sciences **466**, 203–219 (Oct 2018)
27. Thierens, D.: An adaptive pursuit strategy for allocating operator probabilities. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation. pp. 1539–1546 (2005)
28. Wang, Y., Cai, S., Pan, S., Li, X., Yin, M.: Reduction and local search for weighted graph coloring problem. Proceedings of the AAAI Conference on Artificial Intelligence **34**(0303), 2433–2441 (2020)
29. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep sets. Advances in neural information processing systems **30** (2017)