On Monte Carlo Tree Search for 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 the first study of using the popular Monte Carlo Tree Search (MCTS) method combined with dedicated heuristics for solving the Weighted Vertex Coloring Problem. Starting with the basic MCTS algorithm, we gradually introduce a number of algorithmic variants where MCTS is extended by various simulation strategies including greedy and local search heuristics. We conduct experiments on well-known benchmark instances to assess the value of each studied combination. We also provide empirical evidence to shed light on the advantages and limits of each strategy.

Keywords: Monte Carlo Tree Search · local search · graph coloring · weighted vertex coloring.

1 Introduction

The well-known Graph Coloring Problem (GCP) is to color the vertices of a graph using as few colors as possible such that no adjacent vertices share the same color (*legal* or *feasible* solution). The GCP can also be considered as partitioning the vertex set of the graph into a minimum number of color groups such that no vertices in each color group are adjacent. The GCP has numerous practical applications in various domains [14] and has been studied for a long time.

A variant of the GCP called the Weighted Vertex Coloring Problem (WVCP) has recently attracted a lot of interest in the literature [9,16,20,22]. In this problem, each vertex of the graph has a weight and the objective is to find a *legal* solution such that the sum of the weights of the heaviest vertex of each color group is minimized. Formally, given a weighted graph $G = (V, E)$ with vertex set V and edge set E, and let W be the set of weights $w(v)$ associated to each vertex v in V, the WVCP consists in finding a partition of the vertices in V into k color groups $S = \{V_1, \ldots, V_k\}$ $(1 \leq k \leq |V|)$ such that no adjacent vertices belong to the same color group and such that the score $\sum_{i=1}^{k} \max_{v \in V_i} w(v)$ is minimized. One can notice that when all the weights $w(v)$ $(v \in V)$ are equal to one, finding an optimal solution of this problem with a minimum score corresponds to solving the GCP. Therefore, the WVCP can be seen as a more general problem than the GCP and is NP-hard. This extension of the GCP is useful for a

---

⋆ Corresponding author

number of applications in multiple fields such as matrix decomposition problem [18], buffer size management, scheduling of jobs in a multiprocessor environment [17] or partitioning a set of jobs into batches [11].

Different methods have been proposed in the literature to solve the WVCP. First, this problem has been tackled with exact methods: a branch-and-price algorithm [7] and two ILP models proposed in [15] and [5], which transforms the WVCP into a maximum weight independent set problem. These exact methods are able to prove the optimality on small instances, but fail on large instances.

To handle large graphs, several heuristics have been introduced to solve the problem approximately [16,18,20,22]. The first category of heuristics is local search algorithms, which iteratively make transitions from the current solution to a neighbor solution. Three different approaches have been considered to explore the search space: legal, partial legal, or penalty strategies. The legal strategy starts from a *legal* solution and minimizes the score by performing only legal moves so that no color conflict is created in the new solution [18]. The partial legal strategy allows only legal coloring and keeps a set of uncolored vertices to avoid conflicts [16]. The penalty strategy considers both legal and illegal solutions in the search space [20,22], and uses a weighted evaluation function to minimize both the WVCP objective function and the number of conflicts in the illegal solutions. To escape local optima traps, these local search algorithms incorporate different mechanisms such as perturbation strategies [18,20], tabu list [16,20] and constraint reweighting schemes [22].

The second category of existing heuristics for the WVCP relies on the population-based memetic framework that combines local search with crossovers. A recent algorithm [9] of this category uses a deep neural network to learn an invariant by color permutation regression model, useful to select the most promising crossovers at each generation.

Research on combining such learning techniques and heuristics has received increasing attention in the past years [23,24]. In these new frameworks, useful information (e.g., relevant patterns) is learned from past search trajectories and used to guide a local search algorithm.

This study continues on this path and investigates the potential benefits of Monte Carlo Tree Search (MCTS) to improve the results of sequential coloring and local search algorithms. MCTS is a heuristic search algorithm that generated considerable interest due to its spectacular success for the game of Go [8], and in other domains (see a survey of [2] on this topic). It has been recently revisited in combination with modern deep learning techniques for difficult two-player games (cf. AlphaGo [19]). MCTS has also been applied to combinatorial optimization problems seen as a one-player game such as the traveling salesman problem [6] or the knapsack problem [10]. An algorithm based on MCTS has recently been implemented with some success for the GCP in [3]. In this work, we investigate for the first time the MCTS approach for solving the WVCP.

In MCTS, a tree is built incrementally and asymmetrically. For each iteration, a tree policy balancing exploration and exploitation is used to find the most critical node to expand. A simulation is then run from the expanded node and

the search tree is updated with the result of this simulation. Its incremental and asymmetric properties make MCTS a promising candidate for the WVCP because in this problem only the heaviest vertex of each color group has an impact on the objective score. Therefore learning to color the heaviest vertices of the graph before coloring the rest of the graph seems particularly relevant for this problem. The contributions of this work are summarized as follows.

First, we present a new MCTS algorithm dedicated to the WVCP, which considers the problem from the perspective of a sequential coloring with a pre-defined vertex order. The exploration of the tree is accelerated with the use of specific pruning rules, which offer the possibility to explore the whole tree in a reasonable amount of time for small instances and to obtain optimality proofs. Secondly, for large instances, when obtaining an exact result is impossible in a reasonable time, we study how this MCTS algorithm can be tightly coupled with other heuristics. Specifically, we investigate the integration of different greedy coloring strategies and a local search procedure with the MCTS algorithm.

The rest of the paper is organized as follows. Section 2 introduces the weighted vertex coloring problem and the constructive approach with a tree. Section 3 describes the MCTS algorithm devised to tackle the problem. Section 4 presents the coupling of MCTS with a local search algorithm. Section 5 reports results of the different versions of MCTS. Section 6 discusses the contributions and presents research perspectives.

2 Constructive approach with a tree for the weighted graph coloring problem

This section presents a tree-based approach for the WVCP, which aims to explore the partial and legal search space of this problem.

2.1 Partial and legal search space

The search space Ω studied in our algorithm concerns legal, but potentially partial, k-colorings. A partial legal k-coloring S is a partition of the set of vertices V into k disjoint independent sets V_i ($1 \leq i \leq k$), and a set of uncolored vertices $U = V \setminus \bigcup_{i=1}^{k} V_i$. A independent set V_i is a set of mutually non adjacent vertices of the graph: $\forall u, v \in V_i, (u, v) \notin E$. For the WVCP, the number of colors k that can be used is not known in advance. Nevertheless, it is not lower than the chromatic number of the graph $\chi(G)$ and not greater than the number of vertices $|V|$ of the graph. A solution of the WVCP is denoted as partial if $U \neq \emptyset$ and complete otherwise. The objective of the WVCP is to find a complete solution S with a minimum score $f(S)$ given by: $f(S) = \sum_{i=1}^{k} \max_{v \in V_i} w(v)$.

2.2 Tree search for weighted vertex coloring

Backtracking based tree search is a popular approach for the graph coloring problem [1,12,14]. In our case, a tree search algorithm can be used to explore the partial and legal search space of the WVCP previously defined.

Starting from a solution where no vertex is colored (i.e., $U = V$) and that corresponds to the root node R of the tree, child nodes C are successively selected in the tree, consisting of coloring one new vertex at a time. This process is repeated until a terminal node T is reached (all the vertices are colored). A complete solution (i.e., a legal coloring) corresponds thus to a branch from the root node to a terminal node.

The selection of each child node corresponds to applying a move to the current partial solution being constructed. A move consists of assigning a particular color i to an uncolored vertex $u \in U$, denoted as $< u, U, V_i >$. Applying a move to the current partial solution S, results in a new solution $S \oplus < u, U, V_i >$. This tree search algorithm only considers legal moves to stay in the partial legal space. For a partial solution $S = \{V_1, ..., V_k, U\}$, a move $< u, U, V_i >$ is said legal if no vertex of V_i is adjacent to the vertex u. At each level of the tree, there is at least one possible legal move that applies to a vertex a new color that has never been used before (or putting this vertex in a new empty set V_i, with $k + 1 \leq i \leq |V|$).

Applying a succession of $|V|$ legal moves from the initial solution results in a legal coloring of the WVCP and reaches a terminal node of the tree. During this process, at the level t of the tree ($0 \leq t < |V|$), the current legal and partial solution $S = \{V_1, ..., V_k, U\}$ has already used k colors and t vertices have already received a color. Therefore $|U| = |V| - t$.

At this level, a first naive approach could be to consider all the possible legal moves, corresponding to choosing a vertex in the set U and assigning to the vertex a color i, with $1 \leq i \leq |V|$. This kind of choice can work with small graphs but with large graphs, the number of possible legal moves becomes huge. Indeed, at each level t, the number of possible legal moves can go up to $(|V| - t) \times |V|$. To reduce the set of move possibilities, we consider the vertices of the graph in a predefined order $(v_1, \ldots, v_{|V|})$. Moreover, to choose a color for the incoming vertex, we consider only the colors already used in the partial solution plus one color (creation of a new independent set). Thus for a current legal and partial solution $S = \{V_1, ..., V_k, U\}$, at most $k + 1$ moves are considered. The set of legal moves is:

$$\mathcal{L}(S) = \{< u, U, V_i >, 1 \leq i \leq k, \forall v \in V_i, (u, v) \notin E\} \cup < u, U, V_{k+1} > \quad (1)$$

This decision cuts the symmetries in the tree while reducing the number of branching factors at each level of the tree.

2.3   Predefined vertex order

We propose to consider a predefined ordering of the vertices, sorted by weight then by degree. Vertices with higher weights are placed first. If two vertices have the same weight, then the vertex with the higher degree is placed first. This order is intuitively relevant for the WVCP, because it is more important to place first the vertices with heavy weights which have the most impact on the score as well as the vertices with the highest degree because they are the most constrained decision variables. Such ordering has already been shown to be effective with greedy constructive approaches for the GCP [1] and the WVCP [16].

Moreover, this vertex ordering allows a simple score calculation while building the tree. Indeed, as the vertices are sorted by descending order of their weights, and the score of the WVCP only counts the maximum weight of each color group, with this vertex ordering, the score only increases by the value $w(v)$ when a new color group is created for the vertex v.

3   Monte Carlo Tree Search for weighted vertex coloring

The search tree presented in the last subsection can be huge, in particular for large instances. Therefore, in practice, it is often impossible to perform an exhaustive search of this tree, due to expensive computing time and memory requirements. We turn now to an adaptation of the MCTS algorithm for the WVCP to explore this search tree. MCTS keeps in memory a tree (hereinafter referred to as the MCTS tree) which only corresponds to the already explored nodes of the search tree presented in the last subsection. In the MCTS tree, a leaf is a node whose children have not yet all been explored while a terminal node corresponds to a complete solution. MCTS can guide the search toward the most promising branches of the tree, by balancing exploitation and exploration and continuously learning at each iteration.

3.1   General framework

The MCTS algorithm for the WVCP is shown in Algorithm 1. The algorithm takes a weighted graph as input and tries to find a legal coloring S with the minimum score $f(S)$. The algorithm starts with an initial solution where the first vertex is placed in the first color group. This is the root node of the MCTS tree. Then, the algorithm repeats a number of iterations until a stopping criterion is met. At every iteration, one legal solution is completely built, which corresponds to walking along a path from the root node to a leaf node of the MCTS tree and performing a simulation (or playout/rollout) until a terminal node of the search tree is reached (when all vertices are colored).

Each iteration of the MCTS algorithm involves the execution of 5 steps to explore the search tree with legal moves (cf. Section 2):

1. **Selection** From the root node of the MCTS tree, successive child nodes are selected until a leaf node is reached. The selection process balances the exploration-exploitation trade-off. The exploitation score is linked to the average score obtained after having selected this child node and is used to guide the algorithm to a part of the tree where the scores are the lowest (the WVCP is a minimization problem). The exploration score is linked to the number of visits to the child node and will incite the algorithm to explore new parts of the tree, which have not yet been explored.
2. **Expansion** The MCTS tree grows by adding a new child node to the leaf node reached during the selection phase.
3. **Simulation** From the newly added node, the current partial solution is completed with legal moves, randomly or by using heuristics.

Algorithm 1 MCTS algorithm for the WVCP

1: **Input:** Weighted graph $G = (V, W, E)$
2: **Output:** The best legal coloring S^* found
3: $S^* = \emptyset$ and $f(S^*) = MaxInt$
4: **while** stop condition is not met **do**
5:     $C \leftarrow R$ ▷ Current node corresponding to the root node of the tree
6:     $S \leftarrow \{V_1, U\}$ with $V_1 = \{v_1\}$ and $U = V \backslash V_1$ ▷ Current solution initialized with
   the first vertex in the first color group
7:     /* Selection */   ▷ Section 3.2
8:     **while** C is not a leaf **do**
9:        $C \leftarrow$ select_best_child(C) with legal move $< u, U, V_i >$
10:       $S \leftarrow S \oplus < u, U, V_i >$
11:     **end while**
12:     /* Expansion */   ▷ Section 3.3
13:     **if** C has a potential child, not yet open **then**
14:       $C \leftarrow$ open_first_child_not_open(C) with legal move $< u, U, V_i >$
15:       $S \leftarrow S \oplus < u, U, V_i >$
16:     **end if**
17:     /* Simulation */   ▷ Section 3.4
18:     complete_partial_solution(S)
19:     /* Update */   ▷ Section 3.5
20:     **while** $C \neq R$ **do**
21:       update(C,f(S))
22:       $C \leftarrow$ parent(C)
23:     **end while**
24:     **if** $f(S) < f(S^*)$ **then**
25:       $S^* \leftarrow S$
26:       /* Pruning */   ▷ Section 3.6
27:       apply pruning rules
28:     **end if**
29: **end while**
30: return S^*

---

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 algorithm continues until one of the following conditions is reached:

- there are no more child nodes to expand, meaning the search tree has been
  fully explored. In this case, the best score found is proven to be optimal.
- a cutoff time is attained. The minimum score found so far is returned. It
  corresponds to an upper bound of the score for the given instance.

3.2   Selection

The selection starts from the root node of the MCTS tree and selects children nodes until a leaf node is reached. At every level t of the MCTS tree, if the current node C_t is corresponds to a partial solution $S = \{V_1, ..., V_k, U\}$ with t vertices already colored and k colors used, there are l possible legal moves, with $1 \leq l \leq k + 1$. Therefore, from the node C_t, l potential children $C_{t+1}^1, \ldots, C_{t+1}^l$ can be selected.

   If $l > 1$, the selection of the most promising child node can be seen as a multi-armed bandit problem [13] with l levers. This problem of choosing the next node can be solved with the UCT algorithm for Monte Carlo tree search by selecting the child with the maximum value of the following expression [10]:

$$normalized\_score(C_{t+1}^i) + c \times \sqrt{\frac{2 * ln(nb\_visits(C_t))}{nb\_visits(C_{t+1}^i)}}, \text{ for } 1 \leq i \leq l. \quad (2)$$

Here, $nb\_visits(C)$ corresponds to the number of times the node C has been chosen to build a solution. c is a real positive coefficient allowing to balance the compromise between exploitation and exploration. It is set by default to the value of one. $normalized\_score(C_{t+1}^i)$ corresponds to a normalized score of the child node C_{t+1}^i $(1 \leq i \leq l)$ given by:

$$normalized\_score(C_{t+1}^i) = \frac{rank(C_{t+1}^i)}{\sum_{i=1}^l rank(C_{t+1}^i)},$$

where $rank(C_{t+1}^i)$ is defined as the rank between 1 and l of the nodes C_{t+1}^i obtained by sorting from bad to good according their average values $avg\_score(C_{t+1}^i)$ (nodes that seem more promising get a higher score). $avg\_score(C_{t+1}^i)$ is the mean score on the sub-branch with the node C_{t+1}^i selected obtained after all previous simulations.

3.3   Expansion

From the node C of the MCTS tree reached during the selection procedure, one new child of C is open and its corresponding legal move is applied to the current solution. Among the unopened children, we select the node associated with the lowest color number i. Therefore the child node needing the creation of a new color (and increasing the score) will be selected last.

3.4   Simulation

The simulation takes the current partial and legal solution found after the expansion phase and colors the remaining vertices. In the original MCTS algorithm, the simulation consists in choosing random moves in the set of all legal moves $\mathcal{L}(S)$ defined by equation (1) until the solution is completed. We call this first version *random-MCTS*. As shown in the experimental section, this version is not very efficient as the number of colors grows rapidly. Therefore, we propose two other simulation procedures:

- a constrained greedy algorithm that chooses a legal move prioritizing the moves which do not locally increase the score of the current partial solution $S = \{V_1, ..., V_k, U\}$:

$$\mathcal{L}^g(S) = \{< u, U, V_i >, 1 \leq i \leq k, \forall v \in V_i, (u, v) \notin E\} \qquad (3)$$

  It only chooses the move $< u, U, V_{k+1} >$, consisting in opening a new color group and increasing the current score by $w(u)$, only if $\mathcal{L}^g(S) = \emptyset$. We call this version *greedy-random-MCTS*.
- a greedy deterministic procedure which always chooses a legal move in $\mathcal{L}(S)$ with the first available color i. We call this version *greedy-MCTS*.

3.5   Update

Once the simulation is over, a complete solution of the WVCP is obtained. If this solution is better than the best recorded solution found so far S^* (i.e., $f(S) < f(S^*)$), S becomes the new global best solution S^*.

Then, a backpropagation procedure updates each node C of the whole branch of the MCTS tree which has led to this solution:

- the running average score of each node C of the branch is updated with the score $f(S)$:

$$avg\_score(C) \leftarrow \frac{avg\_score(C) \times nb\_visits(C) + f(S)}{nb\_visits(C) + 1} \qquad (4)$$

- the counter of visits $nb\_visits(C)$ of each node of the branch is increased by one.

3.6   Pruning

During an iteration of MCTS, three pruning rules are applied:

1. during expansion: if the score $f(S)$ of the partial solution associated with a node visited during this iteration of MCTS is equal or higher to the current best-found score $f(S^*)$, then the node is deleted as the score of a partial solution cannot decrease when more vertices are colored.
2. when the best score $f(S^*)$ is found, the tree is *cleaned*. A heuristic goes through the whole tree and deletes children and possible children associated with a partial score $f(S)$ equal or superior to the best score $f(S^*)$.
3. if a node is *completely explored*, it is deleted and will not be explored in the MCTS tree anymore. A node is said *completely explored* if it is a leaf node without children, or if all of its children have already been opened once and have all been deleted. Note that this third pruning step is recursive as a node deletion can result in the deletion of its parent if it has no more children, and so on.

These three pruning rules and the fact that the symmetries are cut in the tree by restricting the set of legal moves considered at each step (see Section 2.2) offer the possibility to explore the whole tree in a reasonable amount of time for small instances. This peculiarity of the algorithm makes it possible to obtain an optimality proof for such instances.

4 Improving MCTS with Local Search

We now explore the possibility of improving the MCTS algorithm with local search. Coupling MCTS with a local search algorithm is motivated by the fact that after the simulation phase, the complete solution obtained can be close to a still better solution in the search space that could be discovered by local search.

For this purpose, we propose a strong integration between the MCTS framework and a local search procedure regarding two aspects:

- first, to stay consistent with the search tree learned by MCTS, we allow the local search procedure to only move the vertices of the complete solution S which are still uncolored after the selection and expansion phases. This set of free vertices is called F_V.
- secondly, the best score obtained by the local search procedure is used to update all the nodes of the branch which has led to the simulation initiation.

When coupling MCTS with a local search algorithm, the resulting heuristic can be seen as an algorithm that attempts to learn a good starting point for the local search procedure, by selecting a different path in every iteration during the selection phase.

In this work, we present the coupling of MCTS with an iterated tabu search (ITS) that searches in the same legal and partial search space presented in Section 2. We call this variant *MCTS+ITS*. The ITS algorithm is an adaptation of the ILS-TS algorithm [16], which is one of the best performing local search algorithms for the WVCP.

From a complete solution, the ITS algorithm iteratively performs 2 steps:

- first it deletes the heaviest vertices in F_V from 1 to 3 color groups V_i and places them in the set of uncolored vertices U.
- then, it improves the solution (i.e., minimizes the score $f(S)$) by applying the *one move* operator (consisting in moving a vertex from one color group to another) and the so-called *grenade* operator until the set of uncolored vertices U becomes empty.

The grenade operator $grenade(u, V_i)$ consists in moving a vertex u to V_i, but first, each adjacent vertice of u in V_i is relocated to other color groups or in U to keep a legal solution. In our algorithm, as we restrict the vertices allowed to move to the set of free vertices F_V, the neighborhood of a solution S associated with this grenade operator is given by: $N(S) = \{S \oplus grenade(u, V_i) : u \in F_V, u \notin V_i, 1 \leq i \leq k, \forall v \in V_i, ((u, v) \notin E \text{ or } v \in F_V)\}$.

5   Experimentation

This section aims to experimentally verify the impacts of the different greedy coloring strategies used during the MCTS simulation phase, and the interest of coupling MCTS with local search.

5.1   Experimental settings and benchmark instances

A total of 161 instances in the literature were used for the experimental studies: 30 rxx graphs and 35 pxx graphs from matrix decomposition [18] and 46 small and 50 large graphs from the DIMACS and COLOR competitions.

Two preprocessing procedures were applied to reduce the graphs of the different instances. The first one comes from [22]: if the weight of a vertex of degree d is lower than the weight of the $d+1$th heaviest vertex from any clique of the graph, then the vertex can be deleted without changing the optimal WVCP score of this instance. The second one comes from [4] and is adapted to our problem: if all neighbors of a vertex $v1$ are all neighbors with a vertex $v2$ and the weight of $v2$ is greater or equal to the weight of $v1$, then $v1$ can be deleted as it can take the color of $v2$ without impacting the score.

The MCTS algorithms of Section 3 were coded in C++ and compiled with the g++ 8.3 compiler and the -O3 optimization flag using GNU Parallel [21]. To solve each instance, 20 independent runs were performed on a computer equipped with an Intel Xeon E7-4850 v4 CPU with a time limit of one hour. Running the DIMACS Machine Benchmark procedure dfmax[1] on our computer took 9 seconds to solve the instance r500.5 using gcc 8.3 without optimization flag. The source code of our algorithm is available at `https://github.com/Cyril-Grelier/gc_wvcp_mcts`.

For our experimental studies, we report the following statistics in the tables of results. Column BKS (Best-Known Score) reports the best scores from all methods reported in the literature which have been obtained using different computing tools, CPU or GPU (Graphics Processing Units), sometimes under relaxed conditions (specific fine-tuning of the hyperparameters, large run time up to several days). The scores of the different methods equal to this best-known score are marked in bold in the tables. The optimal results obtained with the exact algorithm MWSS [5] and reported in [16] (launched with a time limit of ten hours for each instance) are indicated with a star. Column "best" shows the best results for a method, "Avg" the average score on the 20 runs, and "t(s)" the average time in seconds to reach the best scores.

Due to space limitations, we only present the results for 25 out of the 161 instances tested. These 25 instances can be considered to be among the most difficult ones.

---

[1] `http://archive.dimacs.rutgers.edu/pub/dsj/clique/`

5.2   Monte Carlo Tree Search with greedy strategies

Table 1 shows the results of MCTS with the greedy heuristics for its simulation (cf. Section 3.4). Column 2 gives the number of vertices of the graph. Column 3 shows the best-known score ever reported in the literature for each instance. Column 4 corresponds to the baseline greedy algorithm: using the predefined vertex ordering presented in Section 2.3, each vertex is placed in the first color group available without creating conflicts (if no color is available, a new color group is created). Columns 5-13 show the results of the three greedy strategies used in the simulation phase of MCTS (cf. Section 3.4): random-MCTS, greedy-random-MCTS, and greedy-MCTS. Column 14 presents the gap between the mean score of the greedy-random-MCTS and the greedy-MCTS variants. Significant gaps (t-test with a p-value below 0.001) are underlined.

Table 1. Comparison between the random-MCTS, greedy-random-MCTS and greedy-MCTS variants (**best known**, proven optimum*, significant gap)

instance	$|V|$	BKS best	greedy best	random-MCTS (1) best	avg	t(s)	greedy-random-MCTS (2) best	avg	t(s)	greedy-MCTS (3) best	avg	t(s)	gap (3) - (2)
C2000.5	2000	2151	2470	3262	3281.5	3150	2584	2603.6	2524	2386	2387.7	3462	-215.9
C2000.9	2000	5486	6449	7300	7334.4	1128	6420	6452.3	3403	6249	6249	382.6	-203.3
DSJC500.1	500	184	224	240	247.6	1243	205	208.15	358	209	209	480.6	0.85
DSJC500.5	500	686	840	815	827.7	1336	760	766.5	606	757	757	1037.1	-9.5
DSJC500.9	500	1662	1916	1830	1857.5	680	1766	1788.3	496	1788	1788	1354.65	-0.3
DSJR500.1	500	169	187	179	184.9	656	**169**	169.05	496.05	177	177	4.6	7.95
flat1000_50_0	1000	924	1350	1572	1639.7	3211	1284	1304.9	3566	1273	1273.6	3516	-31.3
flat1000_60_0	1000	1162	1388	1666	1700.9	3571	1330	1346.75	3416	1303	1303.15	3265.26	-43.6
GEOM120b	120	35*	40	38	38.55	42.69	36	36.95	4	38	38	0	1.05
inithx.i.1	864	569	**569**	571	574.35	3584	**569**	569	0	**569**	569	0	0
latin_square_10	900	1483	1876	2178	2288.15	3575	1729	1766.25	3444	1766	1766	2746.85	-0.25
le450_25a	450	306	321	313	318.35	2311	308	313.45	3380	312	312	18.7	-1.45
le450_25b	450	307	312	310	312.7	2217.5	309	309.4	1797.85	309	309	0	-0.4
miles1500	128	797*	799	**797**	797	1.3	**797**	797	0	**797**	797	0	0
p41	116	2688*	2724	**2688**	2709.55	446.17	**2688**	2688	0.15	**2688**	2688	0	0
p42	138	2466*	2517	2489	2514.05	35	**2466**	2466.45	10	**2466**	2466	0.95	-0.45
queen10_10	100	162	190	173	178.2	691.5	164	168.55	1	169	169	7.35	0.45
queen16_16	256	234	264	269	274.6	854	248	252.65	116	244	244	70.25	-8.65
r28	288	9407*	9409	9439	9560.8	543	**9407**	9411.9	2413.8	**9407**	9407	0	-4.9
r29	281	8693*	8973	8875	9054	1519	**8693**	8698.05	1723.5	8694	8694	0.2	-4.05
r30	301	9816*	9831	9826	9881.4	770	**9816**	9819.6	27.75	9818	9818	0	-1.6
R50_5gb	50	135*	158	**135***	135	727	**135***	135	652.45	**135***	135	384.4	0
R75_1gb	70	70*	91	74	78.4	170	**70***	74.05	1150	72	72	1291.1	-2.05
wap01a	2368	545	628	1050	1071.95	3530	659	663.05	2278	603	603	1385.9	-60.05
wap02a	2464	538	619	1034	1051.2	3505	651	653.6	1268.67	591	591	1586.05	-62.6
			1/25	3/25			10/25			6/25			

First, we observe that the version random-MCTS (Columns 5-7) performs badly even in comparison with the baseline greedy algorithm (Column 4). This is particularly true for the largest instances such as C2000.5 and C2000.9. Indeed, for large instances, choosing random moves in the set of all legal moves is not very efficient as the number of color groups grows rapidly.

The greedy-random-MCTS variant (Columns 8-10) is equipped with a simulation procedure that chooses a legal move in priority in the set of all possible

legal moves which does not locally increase the WVCP score. We observe that this simulation strategy really improves the scores of MCTS in comparison with the random-MCTS version (Columns 5-7) and can reach 10 out of the 25 best-known results. This better heuristic used during the simulation phase helped also to prove the optimality for the instance R75_1gb (indicated with a star) because the best score of 70 was found during the search which led to a pruning of the tree that allowed the tree to be fully explored earlier.

The greedy-MCTS variant (Columns 11-13) always dominates the baseline greedy algorithm (Column 3) for all the instances as it always starts with a simulation using the greedy algorithm at the root node of the tree (first simulation). It only finds 6 over 25 best-known results but is better for the largest instance when we compare the average results with those obtained by the greedy-random-MCTS variant ((Column 14). This is probably due to the fact that forcing the heaviest vertices to be grouped together in the first colors enables a better organization of the color groups. This greedy variant used during the simulation leads to a better intensification which is beneficial for large instances, for which the search tree cannot be sufficiently explored due to the time limit of one hour.

However, with the deterministic simulation of this greedy-MCTS variant, we lose the sampling part with the random legal moves used in the greedy-random-MCTS allowing greater exploration of the search space and a better estimation of the most promising branches of the search tree. This particularity of the greedy-random-MCTS allows finding the best-known score for DSJR500.1, r29, or R75_1gb for example.

To confirm this intuition, we varied the coefficient c balancing the compromise between exploration and exploitation in equation 2 from 0 (no exploration) to 2 (encourage exploration) for different instances. For each value of the coefficient, we performed 20 runs of greedy-random-MCTS per instance during 1h per run. Figure 1 displays 4 box plots showing the distribution of the best scores for the instances queen10_10, le450_25c, latin_square_10 and C2000.5 for the different values of c. We observe 3 typical patterns also seen for the other instances.

The pattern observed for the instance queen10_10 is typical for easy instances or medium instances. For these instances, having no exploration performs badly, then the results get better when the coefficient rises to approximately 1 where it starts to stagnate. For these instances, having no exploration rapidly leads to a local minimum trap and it seems better to secure a minimum of diversity to reach a better score.

Conversely, for very large instances such as C2000.5, we observe a pattern completely different with a best solution when $c = 0$. Indeed, for very large instances, as the search tree is huge and cannot be sufficiently explored due to the time limit, it seems more beneficial for the algorithm to favor more intensification to better search for a good solution in a small part of the tree.

An intermediate pattern appears for medium/hard instances such as le450_25c or latin_square_10 with a U-shaped curve. For these instances, an optimal value of the coefficient c is around the value of 1, with a good balance between exploitation and exploration.
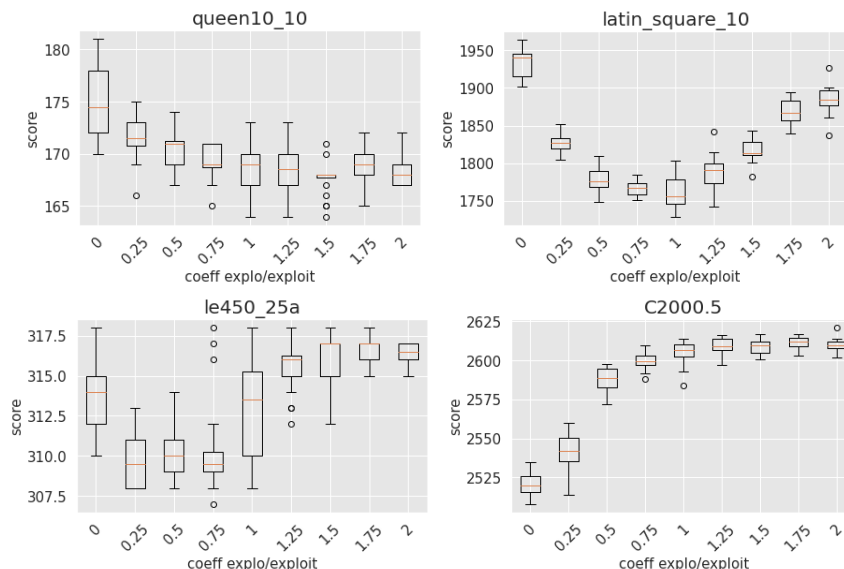
Fig. 1. Boxplots of the distribution of the scores for different values of the coefficient c between 0 and 2, for the instances queen10_10, le450_25c, latin_square_10 and C2000.5. For each configuration, 20 runs are launched with the greedy-random-MCTS variant.

5.3   Monte Carlo Tree Search with local search

Table 2 reports the results of the greedy-MCTS variant (Columns 4-6), the state-of-the art local search ILS-TS algorithm [16] (Columns 7-9) and the MCTS algorithm combined with ITS (MCTS+ITS) with 500 iterations per simulation (Columns 10-12). Column 13 shows the gap between the mean scores of greedy-MCTS and MCTS+ITS. Column 14 shows the gap between the mean scores of ILS-TS and MCTS+ITS. Statistically significant gaps (t-test with a p-value below 0.001) are underlined.

As shown in Table 2, the combination MCTS+ITS significantly improves the results of the greedy-MCTS version (without local search). However, the advantage of this coupling is less convincing compared with the results of the ILS-TS algorithm alone. In fact it degrades the results significantly for 2 instances DSJC500.9 and C2000.9 on average. Still, it greatly improves the average results for the flat1000_60_0 instance.

This may be due to the size of the tree that grows too fast for large instances with a high edge density such as DSJC500.9 and C2000.9. The algorithm does not allow to intensify the search enough with a timeout of 1h. This method would probably require a much longer time to learn more information on the higher part of the tree to reach better results, in particular for large instances.

We also studied combinations of MCTS with other local searches recently proposed in the literature for this problem such as AFISA [20] or RedLS [22] (these results are available at `https://github.com/Cyril-Grelier/gc_wvcp_`

`mcts`). The combination of MCTS with these methods significantly improves the results compared to the local search alone. In general, we observe that the weaker the local search, the greater the interest of combining it with MCTS. ILS-TS [16] being a powerful local search algorithm, the utility of combining it with the MCTS is actually less obvious.

Table 2. Comparison between the greedy-MCTS variant, ILS-TS [16], and MCTS + ITS. (**best known**, proven optimal*, significant gap)

instance	|V|	BKS best	greedy-MCTS (1) best	avg	t(s)	ILS-TS (2) best	avg	t(s)	MCTS+ITS (3) best	avg	t(s)	gap (3) - (1)	gap (3) − (2)
C2000.5	2000	2151	2386	2387.7	3462	2256	2278.52	3438.27	2251	2273.45	3658	-114.25	-5.07
C2000.9	2000	5486	6249	6249	382.6	5944	5996.76	3579.44	6056	6105.4	3611	-143.6	108.64
DSJC500.1	500	184	209	209	480.6	185	188.52	3226.7	189	189.35	1485.08	-19.65	0.83
DSJC500.5	500	686	757	757	1037.1	719	735.76	320.75	721	737.9	2124	-19.1	2.14
DSJC500.9	500	1662	1788	1788	1354.65	1711	1726.1	2178.08	1728	1755.45	2900	-32.55	29.35
DSJR500.1	500	169	177	177	4.6	**169**	169	0.34	**169**	169	31.75	-8	0
flat1000_50_0	1000	924	1273	1273.6	3516	1220	1235	192.4	1211	1228.7	2555	-44.9	-6.3
flat1000_60_0	1000	1162	1303	1303.15	3265.26	1252	1271.86	283.38	1248	1261.05	2787	-42.1	-10.81
GEOM120b	120	35*	38	38	0	**35**	35	1.06	**35**	35	7.05	-3	0
inithx.i.1	864	569	**569**	569	0	**569**	569	0	**569**	569	59.45	0	0
latin_square_10	900	1483	1766	1766	2746.85	1547	1577.14	2508.73	1566	1591.1	3613	-174.9	13.96
le450_25a	450	306	312	312	18.7	**306**	306.29	679.64	**306**	306.4	1382.58	-5.6	0.11
le450_25b	450	307	309	309	0	**307**	307	51.43	**307**	307	81.25	-2	0
miles1500	128	797*	**797**	797	0	**797**	797	2.04	**797**	797	5.1	0	0
p41	116	2688*	**2688**	2688	0	**2688**	2688	0.42	**2688**	2688	4.2	0	0
p42	138	2466*	**2466**	2466	0.95	**2466**	2466	6.73	**2466**	2466	26.3	0	0
queen10_10	100	162	169	169	7.35	**162**	162	20.91	**162**	162	29.35	-7	0
queen16_16	256	234	244	244	70.25	240	241.86	873.87	241	241.85	1411	-2.15	-0.01
r28	288	9407*	**9407**	9407	0	**9407**	9407	0.31	**9407**	9407	23.2	0	0
r29	281	8693*	8694	8694	0.2	**8693**	8693	5.51	**8693**	8693	20.85	-1	0
r30	301	9816*	9818	9818	0	**9816**	9816	1.31	**9816**	9816	31	-2	0
R50_5gb	50	135*	**135***	135	384.4	**135**	135	0.21	**135**	135	1.35	0	0
R75_1gb	70	70*	72	72	1291.1	**70**	70	0.31	**70**	70	1.6	-2	0
wap01a	2368	545	603	603	1385.9	552	555.14	2499.01	550	554.7	1512	-48.3	-0.44
wap02a	2464	538	591	591	1586.05	544	546.14	1739.2	545	547.25	2867.33	-43.75	1.11
				6/25			14/25			14/25			

6   Conclusions and discussion

In this work, we investigated Monte Carlo Tree Search applied to the weighted vertex coloring problem. We studied different greedy strategies and local searches used for the simulation phase. Our experimental results lead to three conclusions.

When the instance is large, and when a time limit is imposed, MCTS does not have the time to learn promising areas in the search space and it seems more beneficial to favor more intensification, which can be done in three different ways: (i) by lowering the coefficient which balances the compromise between exploitation and exploration during the selection phase, (ii) by using a dedicated heuristic exploiting the specificity of the problem (grouping in priority the heaviest vertices in the first groups of colors), (iii) and by using a local search procedure to improve the complete solution. For very large instances, we also observed that

it may be better to use the local search alone, to avoid too early restarts with the MCTS algorithm, which prevents the local search from improving its score.

Conversely, for small instances, it seems more beneficial to encourage more exploration, to avoid getting stuck in local optima. It can be done, by increasing the coefficient, which balances the compromise between exploitation and exploration, and by using a simulation strategy with more randomness, which favors more exploration of the search tree and also allows a better evaluation of the most promising branches of the MCTS tree. For these small instances, the MCTS algorithm can provide some optimality proofs.

For medium instances, it seems important to find a good compromise between exploration and exploitation. For such instances, coupling the MCTS algorithm with a local search procedure allows finding better solutions, which cannot be reached by the MCTS algorithm or the local search alone.

Other future works could be envisaged. For instance, it could be interesting to use a more adaptive approach to trigger the local search, or to use a machine-learning algorithm to guide the search toward more promising branches of the search tree.

Acknowledgements

References

1. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM **22**(4), 251–256 (1979)
2. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (2012)
3. Cazenave, T., Negrevergne, B., Sikora, F.: Monte carlo graph coloring. In: Monte Carlo Search 2020, IJCAI Workshop (2020)
4. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: Proceedings of the 12th International Joint Conference on Artificial intelligence - Volume 1. p. 331–337. IJCAI'91, Morgan Kaufmann Publishers Inc. (1991)
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). https://doi.org/10.1016/j.dam.2016.09.018
6. Edelkamp, S., Greulich, C.: Solving physical traveling salesman problems with policy adaptation. In: 2014 IEEE Conference on Computational Intelligence and Games. pp. 1–8. IEEE (2014)
7. Furini, F., Malaguti, E.: Exact weighted vertex coloring via branch-and-price. Discrete Optimization **9**(2), 130–136 (2012)

8. Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with patterns in Monte-Carlo Go. Ph.D. thesis, INRIA (2006)
9. Goudet, O., Grelier, C., Hao, J.K.: A deep learning guided memetic framework for graph coloring problems. arXiv:2109.05948 [cs] (Sep 2021), `http://arxiv.org/abs/2109.05948`
10. Jooken, J., Leyman, P., De Causmaecker, P., Wauters, T.: Exploring search space trees using an adapted version of monte carlo tree search for combinatorial optimization problems. arXiv:2010.11523 [cs, math] (Nov 2020), `http://arxiv.org/abs/2010.11523`
11. Kavitha, T., Mestre, J.: Max-coloring paths: tight bounds and extensions. Journal of Combinatorial Optimization **24**(1), 1–14 (Jul 2012)
12. Kubale, M., Jackowski, B.: A generalized implicit enumeration algorithm for graph coloring. Commun. ACM **28**(4), 412–418 (1985)
13. Lai, T.L., Robbins, H.: Asymptotically efficient adaptive allocation rules. Advances in Applied Mathematics **6**(1), 4–22 (1985)
14. Lewis, R.: A Guide to Graph Colouring - Algorithms and Applications. Springer (2016), `https://doi.org/10.1007/978-3-319-25730-3`
15. Malaguti, E., Monaci, M., Toth, P.: Models and heuristic algorithms for a weighted vertex coloring problem. Journal of Heuristics **15**(5), 503–526 (2009)
16. 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). https://doi.org/10.1016/j.cor.2020.105087
17. 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, Springer (2005). https://doi.org/10.1007/11523468_86
18. 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)
19. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (Jan 2016). https://doi.org/10.1038/nature16961
20. 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)
21. Tange, O.: Gnu parallel - the command-line power tool. ;login: The USENIX Magazine **36**(1), 42–47 (Feb 2011)
22. 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 (Apr 2020). https://doi.org/10.1609/aaai.v34i03.5624
23. Zhou, Y., Duval, B., Hao, J.K.: Improving probability learning based local search for graph coloring. Applied Soft Computing **65**, 542–553 (2018)
24. Zhou, Y., Hao, J.K., Duval, B.: Frequent pattern-based search: A case study on the quadratic assignment problem. IEEE Transactions on Systems, Man, and Cybernetics: Systems (2020)