Combining Monte Carlo Tree Search and Heuristic Search for Weighted Vertex Coloring

Cyril Grelier[1], Olivier Goudet[1] and Jin-Kao Hao[1*]

[1]LERIA, Université d'Angers, 2 Boulevard Lavoisier, Angers, 49045, France.

*Corresponding author(s). E-mail(s): jin-kao.hao@univ-angers.fr; Contributing authors: cyril.grelier@univ-angers.fr; olivier.goudet@univ-angers.fr;

Abstract

This work investigates the Monte Carlo Tree Search (MCTS) method combined with dedicated heuristics for solving the Weighted Vertex Coloring Problem. In addition to the basic MCTS algorithm, we study several MCTS variants where the conventional random simulation is replaced by other simulation strategies including greedy and local search heuristics. We conduct experiments on well-known benchmark instances to assess these combined MCTS variants. We provide empirical evidence to shed light on the advantages and limits of each simulation strategy. This is an extension of the work [1] presented at EvoCOP2022.

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 [2] and has been studied intensively since the 19th century in mathematics and for over 50 years in computer science.

The Weighted Vertex Coloring Problem (WVCP), a variant of the GCP, has recently attracted much interest in the literature [3–6]. In this problem, each vertex of the graph has a weight and the objective is to find a *legal* coloring 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 ($n = |V|$) 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 $S = \{V_1, \ldots, V_k\}$ ($1 \leq k \leq n$) 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. Note that the value of k is not predetermined for a WVCP instance and may vary during the search as a solution with more colors may have a better score than a solution with less colors. 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. The WVCP can be seen as a more general problem than the GCP and is therefore NP-hard.

The WVCP is a relevant model for several applications such as matrix decomposition [7], buffer size management, and scheduling of jobs into batches in a multiprocessor environment [8]. Let us consider the last application as illustrated in Figure 1. The objective of this scheduling problem is to execute a set of jobs in a minimum total amount of time. There is no constraint on the number of jobs that can be run in parallel in this environment. However, each job requires a specific execution time and exclusive access to certain resources. Therefore, the time required to complete a batch of jobs in parallel is the time required to complete the longest job in that batch, and two jobs requiring the same resource cannot be launched in the same batch. Solving this problem within the WVCP modeling framework can be done in five steps as displayed in Figure 1: (i) a bipartite graph is used to represent the jobs and the resources required for each job; (ii) this bipartite graph is projected onto the resources to obtain a weighted graph where each vertex is a job and two jobs requiring the same resources are linked by an edge; (iii) a weight corresponding to the time needed to complete a job is set on the corresponding vertex of this graph; (iv) after solving the WVCP associated to this graph, a legal solution is found with an optimal score of 25, corresponding to the sum of the weights of the heaviest vertex of each color group; (v) this partition of vertices allows to set up a job schedule in four batches, which respects the resource constraints, and whose minimum total execution time is 25 seconds.

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 [9], two ILP models proposed in [10] and [11] with a transformation of the WVCP into a maximum weight independent set problem, and constraint programming in [12]. These exact methods can prove the optimality on small instances but tends to fail on graphs with more than 250 vertices.

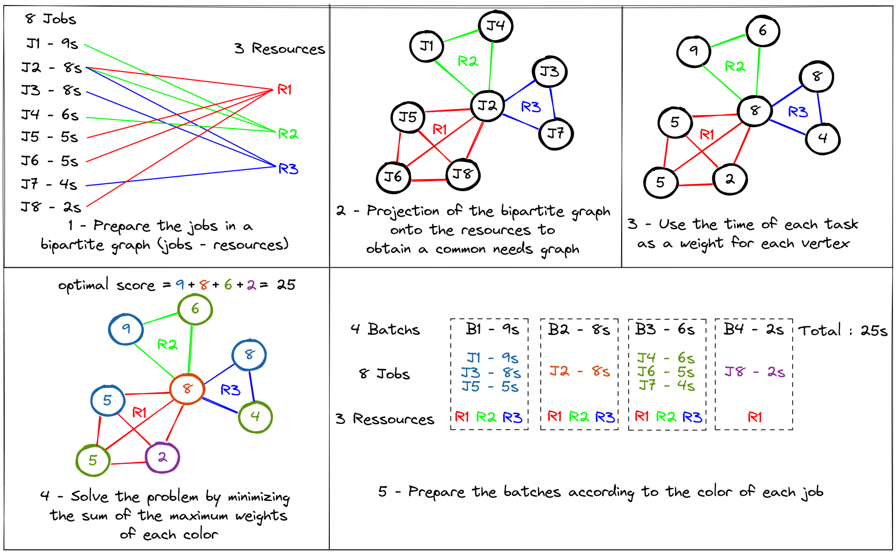

Fig. 1 This figure shows an application of the WVCP for scheduling jobs into batches in a multiprocessor environment with restricted access to certain resources.

To handle large graphs, several heuristics have been introduced to solve the problem approximately [4–7]. The first category of heuristics is based on the local search framework, which iteratively makes 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 [7]. The partial legal strategy allows only legal coloring and keeps a set of uncolored vertices to avoid conflicts [4]. The penalty strategy considers both legal and illegal solutions in the search space [5, 6], 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 [5, 7], tabu list [4, 5] and constraint reweighting schemes [6].

The second category of existing heuristics for the WVCP relies on the population-based memetic framework that combines local search with crossovers. The DLMCOL algorithm [3] 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. The AHEAD algorithm [13] involves automatic selection of crossover and local search operators for the WVCP and GCP.

Research on combining such learning techniques and heuristics has received increasing attention in the past years for graph coloring problems [14, 15]. In these new frameworks, after each search trajectory, a matrix that specifies the probability of a vertex belonging to each color group is updated. This matrix is then used to guide the local search algorithm for subsequent iterations.

This study continues in that vein and investigates the potential benefits of combining Monte Carlo Tree Search (MCTS) and sequential coloring or local search algorithms for solving the WVCP. MCTS is a heuristic search algorithm that generated considerable interest due to its spectacular success for the game of Go [16], and in other domains (see the survey [17] on this topic). It has been recently revisited in combination with modern deep learning techniques for difficult two-player games (cf. AlphaGo [18]). MCTS has also been applied to combinatorial optimization problems seen as a one-player game such as the traveling salesman problem [19] or the knapsack problem [20]. An algorithm based on MCTS has recently been implemented with some success for the GCP in [21]. 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.

Unlike backtracking algorithms, the main aim of MCTS is not to exhaustively test all solutions as fast as possible. Instead, MCTS prioritizes the most promising branches of the search tree using a heuristic that balances exploitation and exploration. This approach can potentially find high-quality solutions faster than backtracking algorithms. However, MCTS can revisit the same solution multiple times during the search process. Nevertheless, as explained later, MCTS, like backtracking algorithms, can provide a proof of optimality for any given instance if it is given sufficient time.

The contributions of this work are summarized as follows.

First, we present a MCTS algorithm dedicated to the WVCP, which considers the problem from the perspective of sequential coloring with a predefined 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 local search procedures within 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 local search. Section 5 reports computational results of 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 n 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 [2, 22, 23]. 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 maximum depth of the tree is n, the number of vertices in the graph.

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, $k + 1 \leq i \leq n$).

Applying a succession of n 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 ($0 \leq t < n$), 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 $|U| = n - t$.

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 $1 \leq i \leq n$. 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 $(n - t) \times n$.

To reduce the set of move possibilities, we consider the vertices of the graph in a predefined order $(u_1, \ldots, u_n)$. When choosing a color for the next vertex, we only consider the colors already used in the partial solution plus one new color, as long as this number is less than the vertex degree plus one. This approach bounds the number of colors needed for a vertex [24]. Thus, for the current partial and legal solution $S = \{V_1, \ldots, V_k, U\}$, at most $d(u) + 1$ moves are considered for the next vertex u to be colored. If k colors have already been used, the set of legal moves to place u is, if $k < d(u) + 1$

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

or, if $k \geq d(u) + 1$

$$\mathcal{L}(S) = \{< u, U, V_i >, 1 \leq i \leq d(u) + 1, \forall v \in V_i, (u, v) \notin E\} \quad (2)$$

This decision cuts symmetries in the tree while reducing the number of branching factors at each level of the tree. The potential number of leaf nodes in the tree is, in the worst case, equal to $\prod_{i=1,\ldots,n} min(i, d_{u_i} + 1)$.

2.3 Predefined vertex order

We propose to consider a predefined ordering of the vertices, sorted by weight and 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 [22] and the WVCP [4].

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 order, 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) that 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 several 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.
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:

Algorithm 1 MCTS algorithm for the WVCP

```
Input: Weighted graph G = (V, W, E)
Output: The best legal coloring S* found
S* = ∅ and f(S*) = MaxInt
while stop condition is not met do
    C ← R    ▷ Current node corresponding to the root node of the tree
    S ← {V_1, U} with V_1 = {v_1} and U = V\V_1    ▷ Current solution initialized with the first vertex in the first color group
    /* Selection */    ▷ Section 3.2
    while C is not a leaf do
        C ← select_best_child(C) with legal move < u, U, V_i >
        S ← S ⊕ < u, U, V_i >
    end while
    /* Expansion */    ▷ Section 3.3
    if C has a potential child, not yet open then
        C ← open_first_child_not_open(C) with legal move < u, U, V_i >
        S ← S ⊕ < u, U, V_i >
    end if
    /* Simulation */    ▷ Section 3.4
    complete_partial_solution(S)
    /* Update */    ▷ Section 3.5
    while C ≠ R do
        update(C,f(S))
        C ← parent(C)
    end while
    if f(S) < f(S*) then
        S* ← S
        /* Pruning */    ▷ Section 3.6
        apply pruning rules
    end if
end while
return S*
```

- 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 optimal 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 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 [25] with l levers. The 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 [20]:

$$normalized\_score(C_{t+1}^i) + c \times \sqrt{\frac{2 * ln(nb\_visits(C_t))}{nb\_visits(C_{t+1}^i)}}, \text{ for } 1 \leq i \leq l. \quad (3)$$

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, which is set by default to one.[1] $normalized\_score(C_{t+1}^i)$ 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 to 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.

Equation 3 can be compared with the probabilities of choice at each level of the tree by an Ant Colony Optimisation (ACO) algorithm, except that in our case, the choice of the child node is deterministic, and explicitly involves a trade-off between an intensification term, the normalized score of each child node, and an exploration term, depending on its number of visits.

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, the node associated with the lowest color number i is selected. 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

---

[1] A sensitivity analysis of this important hyperparameter is shown in Section 5.3.

moves $\mathcal{L}(S)$, defined by Equation 1 and Equation 2, until the solution is completed. We call this first version *MCTS+Random* (MCTS+R). As shown in the experimental section, this version is not very efficient as the number of colors grows rapidly. Therefore, we propose three 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. The move applied for each vertex is randomly selected among the legal moves in $\mathcal{L}(S)$. 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 *MCTS+Greedy-Random* (MCTS+GR).
- a greedy deterministic procedure which always chooses a legal move in $\mathcal{L}(S)$ with the first available color i. We call this version *MCTS+Greedy* (MCTS+G).
- a greedy deterministic procedure based on DSatur [22], which colors in priority the heaviest and most saturated vertices. The saturation of a vertex is the number of colors used by its adjacent vertices. We call this version *MCTS+DSatur* (MCTS+DS).

3.5 Update

Once the simulation is over, a complete solution S 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 $f(S)$:

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

- 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 to or higher than the current best-found score $f(S^*)$, the node is deleted as the score of such 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.

3.7 Toy example

Figure 2 displays one iteration of MCTS for the WVCP on a small graph composed of seven vertices named A–G with different weights between 2 and 9. On each diagram is displayed the current state of the partial coloring solution being constructed (right) and the current state of the search tree (left). In the search tree, each square represents a node and the number on the bottom right of a square is the score of the corresponding partial solution. On top of each square are written the average score and the number of visits of each node. In

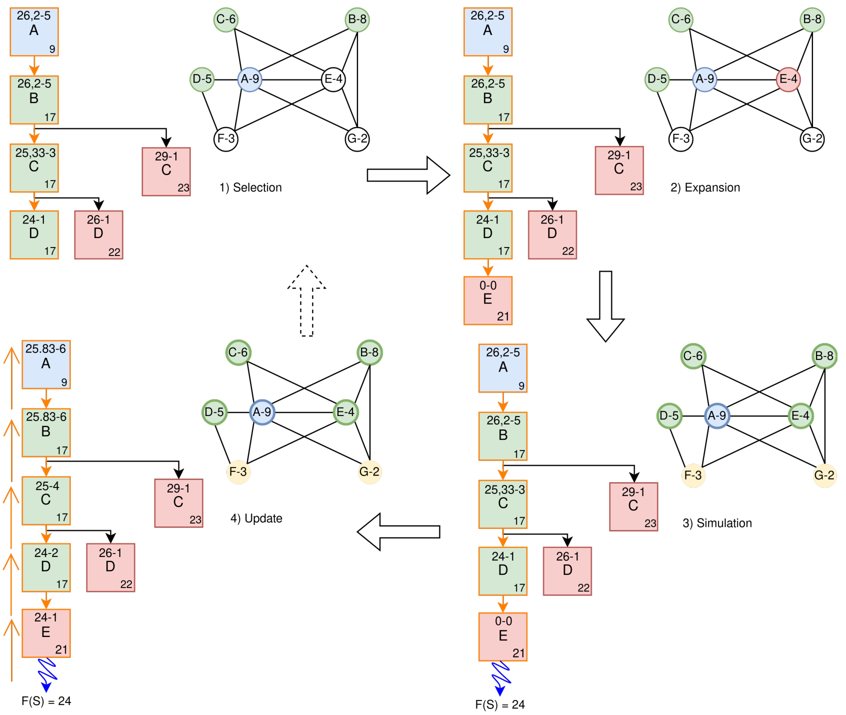

Fig. 2 Toy example of one MCTS iteration

addition to the root node (vertex A colored in blue), five nodes have already been opened in the search tree (five iterations of MCTS). The sixth iteration of MCTS proceeds as follows.

- **Selection**: From the root node, the only possible child corresponding to vertex B in green is selected. From there, there are two options as vertex C can be colored in green or red. The most interesting option is chosen (vertex C in green) regarding the score and the number of visits of each child (cf. equation (3)). Then, the most promising leaf is selected (D in green).
- **Expansion**: From the node D in green, a new node is added to the tree. It corresponds to E in red (as it cannot take the color blue nor green).
- **Simulation**: From there, the solution is completed with a greedy algorithm to obtain a complete legal solution with a score of 24.
- **Update**: This score of 24 is back-propagated on the explored branch (update of the average score and the number of visits of each node in the branch).
- **Pruning**: Figure 3 presents the state of the tree after some iterations. As the best-found score is 24, every branch of the tree with a score greater than or equal to 24 is deleted (indicated with a red cross).

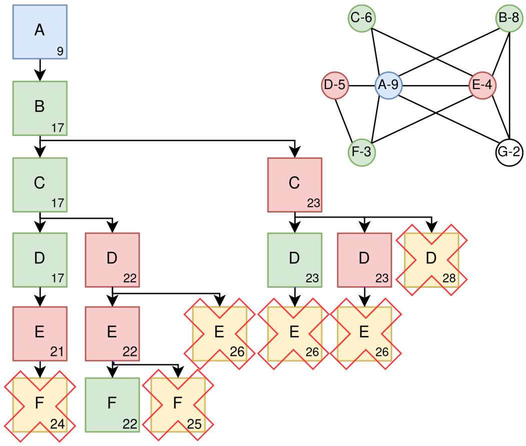

Fig. 3 Toy example of the search tree pruning

4 Combining 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. In this work, we present the coupling of MCTS with a baseline tabu search (TW) created for this work, as well as three *state of the art* local search algorithms, dedicated for the WVCP: AFISA [5], RedLS [6] and ILS-TS [4].

During the simulation phase, the solution is first completed with a greedy algorithm and then improved by the local search procedure. Note that in the first version of this work published in [1], to stay consistent with the search tree learned by MCTS, we allowed the local search procedure to only move the vertices of the complete solution S which are still uncolored after the selection and expansion phases. However, we have realized in the meantime that blocking vertices for the local search can lead to a lot of time spent checking for blocked vertices in the complex neighborhood explored by the various local search procedures. It also leads to missing good opportunities to move in the search space. Therefore, in the new version of the algorithm presented in this paper, a more efficient version of the algorithm is presented where the vertices are not frozen during the local search. In this new version, 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 different best promising backbones of partial solutions in every iteration during the selection phase.

As one iteration does not have the same meaning for each local search, we use a time limit of $t = 0.02 \times n$ seconds to perform the search, depending on the number of vertices n in the given instance. Once the local search procedure has reached the time limit, the score corresponding to the best legal solution obtained by the local search procedure is used to update all the nodes of the branch which has led to the simulation initiation. In the following subsections, the four different local search procedures used in this work are presented.

4.1 Basic tabu search

The first local search algorithm tested is a simple tabu search, named TabuWeight (TW), inspired by the classical TabuCol algorithm for the GCP [26]. Starting from a legal solution, TW improves it iteratively by using the *one move* operator, which consists in moving a vertex from its color group to another color group, without creating conflicts. At each iteration, the best *one move* which is not forbidden by the tabu list is selected. Each time a move is performed, the reverse move is added to the tabu list and forbidden for the next tt iterations where tt is a parameter called tabu tenure. A tabu move can still be applied exceptionally if it leads to a solution, which is better than the best solution found so far (aspiration criterion).

4.2 Adaptive feasible and infeasible tabu search

AFISA [5] is an advanced tabu search algorithm, which explores the candidate solutions by oscillating between illegal and legal search spaces[2]. To prevent the search from going too far from legal boundaries, AFISA uses a controlling

[2]The illegal search space consists of solutions with conflicts (some adjacent vertices in the solution have the same color), while the legal search space consists of solutions without any conflicts.

coefficient to adaptively make the algorithm go back and forth between illegal and legal spaces. The controlling coefficient encourages the algorithm to handle in priority the vertices in conflicts before trying to reduce the WVCP score. AFISA uses the popular *one move* operator to explore the search space.

4.3 Local search with multiple operators

Like AFISA, RedLS [6] explores the illegal and legal search spaces. This algorithm uses the configuration checking strategy [27] that applies multiple improvements and perturbation strategies to explore the search space. At each iteration, RedLS perturbs the solution by moving all the heaviest vertices from one color group to another group, before minimizing the number of conflicts to recover a new legal solution. It uses different variants of the *one move* operator to reduce the number of conflicts while keeping the WVCP score as low as possible. Each conflicting edge has a weight that is increased each time it is not resolved, to give priority to its resolution for the next iterations.

4.4 Iterated local search with tabu search

ILS-TS [4] explores the legal and partial search spaces. From a complete solution, the ILS-TS algorithm iteratively performs 2 steps: (i) it deletes the heaviest vertices from 1 to 3 color groups V_i and places them in the set of uncolored vertices U; (ii) it improves the solution (i.e., minimizes the score $f(S)$) by applying different variants of the *one move* operator and the so-called *grenade* operator until the set of uncolored vertices U becomes empty. The *grenade* operator $grenade(u, V_i)$ moves a vertex u to V_i and relocates each adjacent vertex of u in V_i to another color group or in U to keep a legal solution.

5 Experimentation

This section first describes the experimental settings used in this work. Secondly, we experimentally verify the impacts of the different greedy coloring strategies used during the MCTS simulation phase. Thirdly, an analysis of exploration versus exploration is performed. Lastly, the relevance of coupling MCTS with a local search procedure is studied.

5.1 Experimental settings and benchmark instances

A total of 188 instances are used for the experimental studies: 30 rxx graphs and 35 pxx graphs from matrix decomposition [7] and 123 from the DIMACS and COLOR competitions. The instances are used in a reduced version presented in [12]. The original and reduced instances are available online.[3]

All presented algorithms are coded in C++, compiled, and optimized with the g++ 12.1 compiler. Differences in the results may occur compared to [1] as some optimisations have been done and more reduced instances have been

[3] https://github.com/Cyril-Grelier/gc_instances

used. The source code of our algorithm (and reproduced local searches) is available online[4] with complete spreadsheets of the results. To solve each instance, 20 independent runs were performed on a computer equipped with an Intel Xeon ES 2630, 2,66 GHz CPU with a time limit of one hour, except for the exploration vs. exploitation coefficient tests where 5 to 15 hours were used. Running the DIMACS Machine Benchmark procedure dfmax[5] on our computer took 8.94 seconds to solve the instance r500.5 using gcc 12.1 without optimization flag.

In the following subsections, summary tables allowing general comparisons between the different versions of the algorithms are presented. Detailed results on each specific instance are reported with the source code.[4]

The 188 instances have been separated into four sets: (i) **pxx**, with the 35 pxx instances from [7], (ii) **rxx**, with the 30 rxx instances from [7], (iii) **DIMACS_easy**, corresponding to the *easy* 75 DIMACS and COLOL instances with 72 among them which were solved optimally by exact algorithms [11, 12], and (iv) **DIMACS_hard**, the 48 *hard* DIMACS instances which have never been solved optimally in the literature, except for 5 which are really difficult to solve.

For all the different versions of the MCTS algorithm, the coefficient c, allowing to balance the compromise between exploitation and exploration is set to the value of one (cf. equation (3)). A sensitivity analysis of this important hyperparameter is conducted in Section 5.3.

5.2 Monte Carlo Tree Search with greedy strategies

Table 1 summaries the results of MCTS with greedy heuristics for its simulation (cf. Section 3.4).

Columns 2-5 present results for four instance sets: pxx, rxx, DIMACS_easy, and DIMACS_hard. Column 6 shows global results for all 188 benchmark instances. The table header lists the set name, number of instances, and number of instances with proven optimality (marked with a star). For each method in each column, two numbers are given: the number of instances where the method matches the Best Known Scores (BKS) from the literature[6], and the number of instances solved to proven optimality (marked with a star).

First, we observe that all the MCTS variants dominate the baseline MCTS with greedy simulation in terms of the number of BKS obtained, highlighting the relevance of combining the MCTS framework and search heuristics.

With the MCTS variants, almost all the pxx instances are optimally solved. The rxx instances are more difficult to solve except for the version MCTS+Greedy-Random. The instances from DIMACS_easy are partially solved by each MCTS variant. The instances from DIMACS_hard show a real challenge for all the MCTS variants.

---

[4] https://github.com/Cyril-Grelier/gc_wvcp_adaptive_mcts

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

[6] Note that some of these BKS have been found in the literature with extended search time from several hours to several days, while our methods are only run during one hour.

Table 1 Summary of the number of times the Best Known Score (BKS) is reached by each algorithm. The values with a star indicate the number of times a score has been proved optimal. Values in bold highlight the best results for each line.

	pxx 35*	rxx 30*	DIMACS easy 75 - 72*	DIMACS hard 48 - 5*	Total 188 - 142*
R	2	0	3	0	5
MCTS+R	34 – **26***	0	41 – 22*	0	75 – 48*
GR	14	0	15	0	29
MCTS+GR	**35 – 26***	**22**	**57 – 23***	0	**114 - 49***
G	14	3	10	0	27
MCTS+G	**35-26***	11	46 – 22*	0	92 – 48*
DS	14	2	11	0	27
MCTS+DS	**35**-25*	11	47 – 22*	0	93 – 47*

To better compare these different algorithms, and not only relying on the number of best-known scores achieved (that can sometimes be found by "chance"), we performed pairwise comparisons between the algorithms based on the average scores obtained on each instance as displayed in Table 2.

In Table 2, the numbers in each row correspond to the number of instances for which the method is significantly better than another (with a maximum of 188 instances). A method is said significantly better than another on a given instance if its average score measured over 20 runs is significantly better (t-test with a p-value below 0.001). The column *Total* corresponds to the number of times a method is better than another.

Table 2 shows the ranking of the different methods. Unsurprisingly, the pure random heuristic is completely dominated by all methods. The variant MCTS+DS is significantly better compared to the others. In particular, it stays significantly better 48 times out of 188, versus 32 times in favor of the MCTS+Greedy-Random. Indeed, it seems that for the WVCP, the DSatur procedure, which colors the heaviest vertices that have the most colored neighbors, leads to better organization of color groups.

Table 2 Comparison between all greedy and MCTS variants. As an example, the row for MCTS+Random means that the method is better for 186 instances compared to the random procedure (R), is better for 169 instances compared to the Greedy-Random procedure (GR), and is never better compared to MCTS+GR. Values in bold highlight the highest value between two methods. As an example, the variant MCTS+G is more often significantly better (44 times) than MCTS+GR (which is better than MCTS+G 23 times).

/188 instances	R	MCTS+R	GR	MCTS+GR	G	MCTS+G	DS	MCTS+DS
R	-	0	0	0	0	0	0	0
MCTS+R	**186**	-	**169**	0	**92**	1	**90**	1
GR	**186**	10	-	0	4	0	7	0
MCTS+GR	**186**	**122**	**178**	-	**151**	23	**152**	32
G	**186**	36	**133**	10	-	0	37	1
MCTS+G	**186**	**121**	**178**	**44**	**158**	-	**160**	30
DS	**186**	40	**139**	9	**47**	0	-	0
MCTS+DS	**186**	**121**	**178**	**48**	**157**	**35**	**158**	-

Table 3 Results of the MCTS with greedy simulation on a part of the 188 instances of the literature. At the foot of the table, for each method, we report the number of BKS achieved, the number of best scores, the number of average best scores compared to other methods, and the number of instances solved to optimality. The mean is not shown if it is equal to the best score. A star (*) indicates that the score has been proven to be optimal.

instance	BKS	MCTS+R			MCTS+GR			MCTS+G			MCTS+DSatur		
		best	mean	time	best	mean	time	best	mean	time	best	mean	time
C2000.5	2144	3178	3198.3	2912	2505	2537.7	3422	**2385**		1685	2397	2398.8	3390
C2000.9	5477	7022	7166.3	3554	6233	6272.9	3575	**6125**	**6147.8**	3238	6275		163
DSJC125.1g	23*	27	29.4	2332	**25**	25.4	22	**25**		0	**25**		0
DSJC125.5g	71	78	80.8	755	**74**	75.3	52	77		0	**74**		67
DSJC125.9g	169*	173	176.8	17	**170**	172.7	25	171		1630	171		21
DSJC250.1	127	159	165.2	68	**134**	141.4	13	141		4	139		1151
DSJC250.5	392	444	457.8	2941	422	429.4	14	427		92	**421**		705
DSJC250.9	934*	1002	1026.2	20	**973**	988.7	2793	986		16	984		559
DSJC500.1	184	237	245.9	498	**203**	208.3	148	**203**		638	**203**		2943
DSJC500.5	685	809	824.2	899	**754**	765.6	136	755		635	775	780.9	3542
DSJC500.9	1662	1831	1858.5	2645	**1771**	**1787.4**	113	1794		171	1795	1797.1	3453
DSJC1000.1	300	392	440.8	3480	334	337.4	1618	**333**		2823	340		2474
DSJC1000.5	1185	1385	1399.2	2699	**1271**	**1293.6**	1668	1318		1437	1338		203
DSJC1000.9	2836	3164	3200.8	1750	**3040**	**3070.4**	978	3078		2863	3172		3338
DSJR500.1	169*	178	184.4	154	**169**		46	177		0	176		21
flat1000_50_0	924	1337	1358.5	3022	**1236**	1255.8	1452	1251	**1251.1**	3037	1303		1843
flat1000_60_0	1162	1375	1400	3400	1275	1295.8	1478	**1260**	**1260.7**	3424	1343		311
flat1000_76_0	1165	1349	1370.3	2910	1252	1269.7	1477	**1244**	**1248.2**	3557	1313		1798
GEOM120a	105*	107	111.5	14	106	106.6	330	**105**		1129	109		5
GEOM120b	35*	**37**	38	10	**37**		0	**37**		498	**37**		110
GEOM120	72*	74	76.8	5	**72**	73	0	**72**		0	**72**		36
inithx.i.1	569*	**569**	569.5	63	**569**		0	**569**		0	**569**		0
inithx.i.2	329*	330	334.6	69	**329**	**329.9**	1720	330		0	331		0
inithx.i.3	337*	338	340.9	116	**337**	**337.7**	1449	339		0	339		0
latin_square_10	1480	1861	1888.6	2092	**1721**	1757.1	966	1726	**1726.8**	3418	1805		2360
le450_25a	306	315	320.2	879	**307**	**310**	3364	312		5	310		1919
le450_25b	307*	**309**	313.4	948	**309**	309.1	993	**309**		0	**309**		224
le450_25c	342	390	397.6	1501	**365**	372.7	2844	369		950	376		1572
le450_25d	330	384	391.8	1830	**354**	**358.8**	2802	364		1657	369		1411
myciel7gb	109*	120	133.8	18	117	118.3	476	**109**		42	**109**		38
myciel7g	29*	32	36.1	670	**29**	29.8	578	**29**		0	**29**		0
queen10_10	162	174	178.2	692	**165**	169.1	986	171		0	169		8
queen11_11	172	185	189.8	384	**178**	180.4	941	180		80	179		50
queen12_12	185	198	208.2	547	**189**	196.8	2	193		1102	197		1823
queen13_13	194	209	219.8	43	204	207.6	1125	205		5	**199**		71
queen14_14	215	237	241.4	247	**224**	226.1	38	**224**		13	225		2351
queen15_15	223	252	261.6	415	**237**	241.3	216	239		49	238		315
queen16_16	234	268	274	1799	250	252.7	127	**247**		86	248		420
R75_1gb	70*	73	77.5	1171	**70***	**73.6**	2679	76		184	75		9
R75_1g	18*	**18***	19.8	2579	**18***	18.8	1164	**18***		956	**18***		1348
R75_5gb	186*	195	200.7	91	**186**	**191.7**	237	192		0	192		2234
R75_5g	51*	54	55.2	608	**51**	52.7	725	52		203	**51**		1956
R75_9gb	396*	398	400.9	1281	**396**	397.9	1902	**396**		208	**396**		191
R75_9g	110*	**110**	111.5	850	**110**		39	**110**		7	**110**		202
R100_1gb	81*	89	97.1	1326	**84**	87	0	**84**		0	**84**		1
R100_1g	21*	24	26.2	1	**22**	**22.6**	728	23		0	23		0
R100_5gb	220	232	238.9	28	**224**	230.7	4	230		0	225		20
R100_5g	59	63	64.7	1389	62	62.4	604	63		0	**61**		7
R100_9gb	518*	530	538.5	43	**518**	**521**	273	526		2696	528		46
R100_9g	141*	144	146.7	0	**143**	143.9	29	**143**		5	**143**		301
wap01a	545	1039	1053.3	3450	657	659.8	2965	599		3272	**595**		2904
wap02a	538	1021	1038	3533	645	648.7	1301	**588**		3265	590		262
wap03a	562	1416	1445.4	2767	746	749	803	653		1096	**647**		1805
wap04a	563	1480	1493.9	1750	755	757.8	3085	649		603	**643**		828
wap05a	541	713	724.9	3536	583	591.1	1867	566		172	**565**		2882
wap06a	516	747	753.8	3359	591	595.4	2280	566		9	**561**		664
wap07a	555	1018	1030.2	2630	693	697.2	3186	**635**		165	639		986
wap08a	529	984	998.2	1245	664	672	3539	612		2401	**604**		2583
p40	4984*	**4984**	4992.8	45	**4984**		0	**4984**		0	**4984**		0
p41	2688*	**2688**	2700.7	14	**2688**		0	**2688**		0	**2688**		0
p42	2466*	2480	2515.1	16	**2466**	2466.8	33	**2466**		0	**2466**		4
r28	9407*	9435	9532	3599	**9407**	9409.9	1530	**9407**		0	**9407**		0
r29	8693*	8750	8999.5	155	**8693**	8695.5	1316	8694		0	8694		3
r30	9816*	9825	9876.5	384	**9816**	9819.6	11	9818		0	9818		0
#BKS			75			114			92			93	
#Best			79			166			111			117	
#Best Avg			56			104			131			138	
#Optimal			48			49			48			47	

This is particularly true for the largest instances, where choosing random moves in the set of all legal moves is not very efficient as the number of color groups grows rapidly. It explains also why the variant MCTS+Random performs badly on larger or denser instances such as the rxx instances or some difficult DIMACS instances.

However, with the deterministic simulation of the MCTS+Greedy or MCTS+DSatur variants, there is no sampling of the legal moves like in the MCTS+Greedy-Random variant allowing greater exploration of the search space and a better estimation of the most promising branches of the search tree. This particularity of the MCTS+Greedy-Random variant allows us to find the BKS for more instances. Moreover, when exploring Table 3, which presents the results for a part of the 188 instances of the literature, one can see that the R75_1gb instance from the DIMACS_easy set is proven optimal by MCTS+Greedy-Random but not by MCTS+Greedy or MCTS+DSatur. With stochastic help, the MCTS+Greedy-Random version can reach the best known score of 70, which leads to an early pruning of the tree and allows to prove the optimality earlier.

5.3 Exploitation vs exploration coefficient analysis

One key element of the MCTS algorithm is the coefficient c balancing the compromise between exploration and exploitation in Equation (3). In this subsection, we investigate the importance of this coefficient by varying it and presenting the evolution of the score over time. For this experimentation, the coefficient c varied from 0 (no exploration) to 5 (encourage exploration). For each coefficient value, 20 runs of the MCTS+GR variant per instance are performed during 5h per run (15h for the very large C2000.x instances)[7].

Figure 4 displays 6 plots showing the evolution of the mean of the best scores over time for the instances DSJC500.5, latin_square_10, le450_25a, wap01a, C2000.5 and C2000.9 for the different values of the coefficient c. These 6 instances come from the set of DIMACS_hard instances and can be considered as very difficult. Four typical patterns also seen for other instances are observed:

- P1: instances requiring a lot of exploration,
- P2: instances requiring more exploration than exploitation,
- P3: instances requiring more exploitation than exploration,
- P4: instances requiring a lot of exploitation.

The first pattern P1 is observed for the instance DSJC500.5 and also queen instances. For these instances, the lack of exploration leads to poor results, and better results are reached as the coefficient c increases. The pattern P2 is observed for the instance latin_square_10, but also for other instances such as flat1000 where the best score obtained in function of the coefficient c has a U-shape, with an optimal value of c between 1 and 2. This phenomenon can

[7]This longer execution time explains some differences with the sensitivity analysis of this parameter made in [1] with only 1h of computation time.

also be observed on instances such as C2000.5 and C2000.9 where it becomes quickly more interesting to explore up to a certain point.

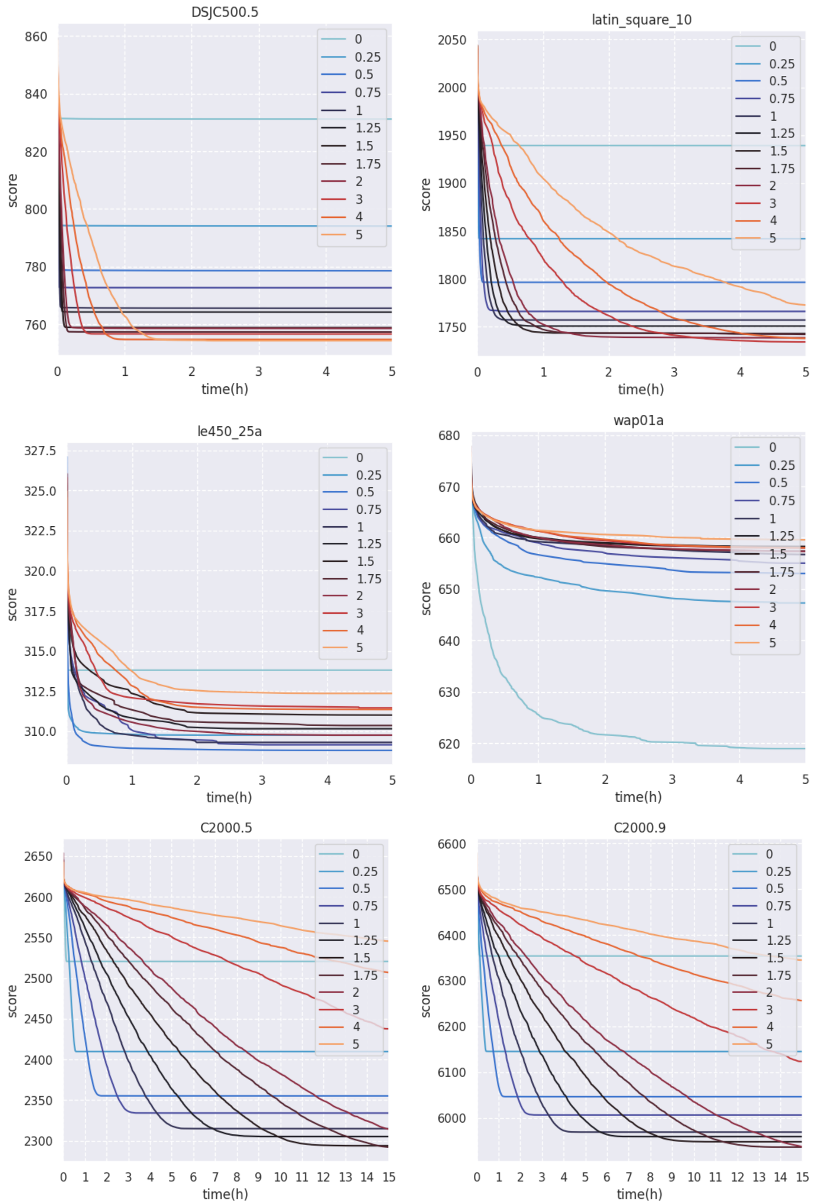

Fig. 4 Plots of the evolution of the means of the scores over time for different values of the coefficient c between 0 and 5, for the instances DSJC500.5, latin_square_10, le450_25a, wap01a, C2000.5, and C2000.9. For each configuration, 20 runs are launched with the MCTS+Greedy-Random variant for 5h and 15h for the C2000.x instances.

The pattern P3 found for the le450 instance shows the best results when there is only weak exploration, but the results are worse when c is set to zero.

In general, for the patterns P1, P2 and P3, having no exploration at all rapidly leads to a local minimum trap and it seems better to secure a minimum of diversity to reach a better score, while, for the pattern P4, found for the wap instances (very large instances), giving a chance to the exploration leads the algorithm to be lost in the search space. 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. To sum, the most suitable exploration vs exploitation coefficient thus depends on the instance considered. Finding the right general coefficient is a challenging task. In this work, We adopted the coefficient $c = 1$ for all other experiments.

5.4 Monte Carlo Tree Search with local search

This section studies the effects of the combination of MCTS with a local search heuristic. Table 4 summarizes the results of the local search procedures presented in Section 4 followed by the combination of MCTS with each of these local search procedures during the simulation phase. Each line presents the number of times a BKS is reached with the method. Note that the objective of these MCTS variants coupled with a local search procedure is not to prove optimality. For instances where the optimal score is known, MCTS and local search stop when this score is reached.

We observe from this table that the local search procedure allowing to reach the highest number of BKS is ILS-TS, followed by RedLS. When coupled with the MCTS framework, TW, AFISA, and RedLS improve their results. In particular, the variant MCTS+RedLS finds the BKS for 17 difficult instances of the DIMACS_hard set and 152 instances over 188 in total. It highlights that the MCTS framework proposed in this paper can help a local search algorithm such as RedLS to continuously find new promising starting points in the search space.

Table 4 Summary of the number of times the BKS is reached for the local search procedures and MCTS combined with the different local search procedures. Values in bold highlight the best results for each column.

	pxx 35*	rxx 30*	DIMACS easy 75(72*)	DIMACS hard 48 (5*)	Total 188 (142*)
AFISA	**35**	5	71	3	114
MCTS+AFISA	34	3	73	5	115
TW	29	2	61	7	99
MCTS+TW	**35**	16	72	9	132
RedLS	31	3	70	8	112
MCTS+RedLS	**35**	25	**75**	17	152
ILS-TS	**35**	**30**	**75**	**19**	**159**
MCTS+ILS-TS	**35**	**30**	**75**	**15**	155

However, when coupled with the ILS-TS algorithm (variant MCTS+ILS-TS), it does not seem to improve the results. It may be explained by the fact that ILS-TS is an iterated local search algorithm already integrating various perturbation mechanisms allowing to escape local optima. Therefore, finding new good starting points in the search space guided by MCTS seems less interesting for ILS-TS than for other local search procedures such as TW, AFISA, and RedLS.

Table 5 shows pairwise comparisons between all the MCTS variants and all the local search procedures.

When comparing the performances of the local search algorithms between each other by looking at the number of times they are significantly better than the others RedLS is at the end of the ranking after TabuWeight and AFISA while ILS-TS is far more often better.

The MCTS+local_search variants always improve the number of BKS found and are more often significantly better than the corresponding local search only, except for ILS-TS, which is better on its own.

The variant MCTS+RedLS is more often significantly better than the other methods, even compared to ILS-TS. These results indicate that combining MCTS with a local search is of interest to improve the underlying local search procedures such as AFISA, TW, and RedLS.

Table 6 shows the results for each local search and MCTS+LS on a portion of the 188 instances in the literature, with the most difficult instances more represented than the easiest. The values in bold show the best results among the different methods for the best score or the mean score.

Table 5 Comparison between all the MCTS variants and all the local search procedures. Each row corresponds to the number of times the method of the row is significantly better than the method on the column on the 188 instances. For example, out of 188 instances, TW is better than MCTS+TW on 25 instances, 48 times against RedLS and 3 times against MCTS+RedLS. Values in bold means that method a is more often better than method b when we look at how many times b is significantly better than a. The Total column corresponds to the number of times a method is more often significantly better than the others.

/188 instances	MCTS+GR	MCTS+DSatur	AFISA	MCTS+AFISA	TabuWeight	MCTS+TW	RedLS	MCTS+RedLS	ILSTS	MCTS+ILSTS	Total
MCTS+GR	-	32	**77**	**63**	**77**	33	**85**	9	0	1	4/9
MCTS+DSatur	**48**	-	**93**	**72**	**89**	42	**107**	22	0	2	5/9
AFISA	33	50	-	35	**49**	17	**53**	2	0	0	2/9
MCTS+AFISA	40	39	**40**	-	**73**	10	**86**	2	1	0	3/9
TabuWeight	56	54	40	40	-	25	**48**	3	0	3	1/9
MCTS+TW	**52**	**64**	**74**	**73**	**78**	-	**101**	8	1	0	6/9
RedLS	51	57	41	35	47	29	-	5	14	15	0/9
MCTS+RedLS	**88**	**87**	**99**	**75**	**105**	**65**	**103**	-	**23**	**27**	9/9
ILSTS	**101**	**95**	**104**	**82**	**106**	**75**	**96**	22	-	**18**	8/9
MCTS+ILSTS	**101**	**93**	**102**	**82**	**103**	**73**	**92**	18	2	-	7/9

Table 6 Results of the LS alone and MCTS + LS variants on a portion of the 188 instances of the literature. At the foot of the table, for each method, we report the number of BKS achieved, the number of best scores, the number of average best scores compared to other methods, and the number of instances solved to optimality. The mean is not shown if it is equal to the best score.

instance	BKS	AFISA			MCTS+AFISA			TabuWeight			MCTS+TW			RedLS			MCTS+RedLS			ILSTS			MCTS+ILSTS		
		best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time
C2000.5	2144	2384	2403.4	3601	2635	2648.9	2665	2318	2332.2	3477	2410	2423.6	1578	**2167**	**2193.8**	2403	2199	2215.2	2807	2237	2266.4	3498	2318	2340.2	697
C2000.9	5477	6582	6650.1	0	6547	6570.2	1107	6049	6073.4	2819	6242	6293.6	1845	**5502**	**5528.1**	3303	5724	5742.2	1604	5910	5969.9	3587	6103	6119.9	430
DSJC125.1g	23*	**23**	24.4	13	**23**	23.5	1203	24	24.6	68	**23**	23.1	1684	**23**	23.4	0	**23**		37	**23**		2	**23**		5
DSJC125.5g	71	**71**	72.2	1360	72		529	**71**	71.5	1661	**71**	71.8	1686	72		257	**71**	71.2	1462	**71**		126	**71**		40
DSJC125.9g	169*	170	174.1	2	**169**	169.1	1047	**169**	170.7	747	**169**	169.8	959	**169**		0	**169**		0	**169**		182	**169**		171
DSJC250.1	127	129	133.6	3294	133	133.8	1764	134	136.8	25	132	133.5	780	130	131.6	1	**127**	128.4	867	**127**	**127.8**	1608	**127**	128	642
DSJC250.5	392	411	424.2	541	421	428.8	2586	397	406	2717	406	408.6	2349	398	401.2	103	394	398.4	1989	**393**	**397.6**	2567	397	399.8	1695
DSJC250.9	934*	949	976.1	232	962	978.8	1380	959	963.2	1766	950	956	2292	**934**	935.6	718	935	**935.5**	1279	936	942.1	3053	948	953.4	2604
DSJC500.1	184	198	201.5	1817	220	228.6	165	194	197	1526	201	202.1	1312	**187**	201.9	537	**187**	**188.1**	2087	188	188.8	1744	**187**	188.6	3168
DSJC500.5	685	762	778.1	1307	848	863.5	1474	733	741.3	3571	751	756	2233	**706**	716.1	2840	710	**713.6**	2626	724	735.5	1744	729	734.7	3410
DSJC500.9	1662	1744	1775.5	652	1909	1919.3	2002	1733	1755.1	228	1764	1775	1705	**1670**	**1675.1**	945	1674	1680.1	404	1720	1742	3039	1762	1775.7	77
DSJC1000.1	300	319	325	2182	396	401.9	357	313	316.2	2943	365	368.8	2604	305	307.2	1235	304	306.5	2814	305	307.4	1574	**303**	**304.9**	1176
DSJC1000.5	1185	1308	1330	3531	1463	1475	2226	1271	1282.3	3460	1304	1311.1	2562	**1198**	**1213.7**	2381	1214	1218.7	2278	1245	1269.2	231	1257	1274.2	1890
DSJC1000.9	2836	3066	3107	3601	3303	3325.1	2793	3034	3061.1	1381	3135	3152.9	546	**2840**	**2858.5**	2953	2884	2892.8	2211	3026	3066.8	3580	3098	3123.7	3003
DSJR500.1	169*	**169**		54	**169**		157	171	178.2	21	**169**	169.2	1273	171	184.5	0	**169**	169.2	1270	**169**		0	**169**		5
flat1000_50_0	924	1267	1293.3	2736	1421	1431.3	1512	1238	1247.5	3230	1271	1274.9	2194	**1155**	**1173.8**	3238	1173	1179.1	1005	1222	1235	1712	1221	1231.3	42
flat1000_60_0	1162	1309	1323.2	3584	1465	1476.8	1575	1275	1288	2275	1305	1313.5	21	**1191**	**1205.7**	1080	1208	1216	201	1250	1270	125	1259	1266.2	3360
flat1000_76_0	1165	1288	1304.4	3278	1442	1449.8	2341	1237	1263	2975	1285	1292.8	693	**1176**	**1194**	1107	1189	1196.6	201	1232	1247.5	1198	1237	1246.5	1008
GEOM120a	105*	**105**	106.6	136	**105**	105.1	515	**105**	106.1	453	**105**		351	**105**	109.2	9	**105**		67	**105**		0	**105**		0
GEOM120b	35*	**35**	36.2	422	**35**	36	2557	**35**	35.3	1156	**35**	35.3	2051	**35**	35.5	0	**35**		5	**35**		0	**35**		0
GEOM120	72*	**72**	73	0	**72**		8	74	76.2	0	**72**	73	2097	**72**	75.2	0	**72**		77	**72**		0	**72**		0
inithx.i.1	569*	**569**	573.1	0	**569**		19	**569**	572.4	0	**569**		25	**569**		11	**569**		12	**569**		16	**569**		8
inithx.i.2	329*	334	337.6	448	331	333.9	150	338	342.5	0	332	334.1	2510	**329**	329.1	311	**329**		127	**329**		8	**329**		7
inithx.i.3	337*	340	346.4	1184	339	341	985	342	347.1	3	338	340.4	3380	**337**	341.6	75	**337**		643	**337**		29	**337**		30
latin_square_10	1480	1607	1652.4	2257	2037	2055.9	2394	1816	1855.2	45	1774	1789.8	247	**1505**	**1523**	2369	1518	1527.1	543	1559	1581.2	1368	1572	1585.8	3458
le450_25a	306	311	316.3	2699	316	318.9	283	321	325.4	83	313	315.9	1456	**306**	307.4	503	**306**		441	**306**		174	**306**		736
le450_25b	307*	308	312.6	1891	308	310.1	2114	308	312.9	219	**307**	308.5	1491	**307**	313.4	56	**307**		203	**307**		10	**307**		18
le450_25c	342	355	364.4	3436	386	398.2	3375	361	366	677	371	373.6	1908	351	354.8	43	**348**	**349.6**	2165	**348**	351.8	3207	351	353.1	1689
le450_25d	330	351	357.8	1630	383	391.8	1494	351	356.4	1955	363	366.3	3415	**332**	338.9	154	334	**335.2**	2477	339	342.5	1999	342	343.8	2283
myciel7gb	109*	**109**	111.8	1129	**109**	109.1	1343	114	115.8	738	112	113.5	2646	**109**	116.4	0	**109**	109.1	955	**109**		4	**109**		5
myciel7g	29*	**29**	30.7	2542	**29**		497	**29**	29.2	324	**29**		115	**29**	29.8	244	**29**		66	**29**		0	**29**		4
queen10_10	162	165	166.9	524	**162**	164.1	2994	163	163.9	1629	**162**	162.3	1249	**162**	166.2	504	**162**		122	**162**		13	**162**		10
queen11_11	172	179	181.1	722	176	178.8	2859	**172**	172.9	1731	**172**	173.2	1071	174	177.1	761	**172**	173.8	2575	**172**	**172.6**	1820	**172**	172.8	1703
queen12_12	185	193	196.7	2062	191	193.9	3216	187	188.1	1881	187	188.4	2603	188	190.2	7	186	186.9	2494	**185**	**186.1**	1261	186	186.4	1572
queen13_13	194	203	206.7	288	202	205.2	1684	198	201.2	2205	199	200.5	2765	195	198.7	1528	**194**	**194.7**	2243	**194**	195.7	3475	195	196.6	2480
queen14_14	215	224	230.5	3113	226	229.2	184	217	218.4	1678	219	220.1	2014	217	222.4	18	**216**	**217.3**	3040	**216**	**217.3**	2018	217	218.4	1514
queen15_15	223	236	240.2	1572	241	242.9	1745	233	236.8	2417	233	235.5	1835	227	229.7	1109	**225**	**226.6**	1794	227	228.4	2167	227	229.2	1925
queen16_16	234	248	255.2	1055	255	258.2	3012	239	242.3	2745	243	244.8	1002	237	239.9	91	**235**	**237.4**	1300	238	240.2	2938	240	241.2	1204
R75_1gb	70*	**70**	73.7	0	**70**		17	76	80.3	0	**70**	70.5	1432	**70**	78.3	0	**70**		60	**70**		0	**70**		0
R75_1g	18*	**18**		94	**18**		15	19	19.6	0	**18**		164	**18**	19.1	0	**18**		24	**18**		0	**18**		0
R75_5gb	186*	**186**	188.6	28	**186**		45	**186**	186.3	135	**186**		9	**186**	192.1	0	**186**		72	**186**		1	**186**		1
R75_5g	51*	**51**	51.9	17	**51**		13	**51**		0	**51**		0	**51**	51.3	108	**51**		31	**51**		0	**51**		0
R75_9gb	396*	**396**	396.4	8	**396**		12	**396**	396.6	1911	**396**	396.1	1380	**396**		0	**396**		0	**396**		10	**396**		16
R75_9g	110*	**110**	110.2	3	**110**		0	**110**		0	**110**		0	**110**		0	**110**		0	**110**		0	**110**		0
R100_1gb	81*	**81**	82.5	1627	**81**		246	86	90	0	**81**	82.8	2664	82	84.5	0	**81**		594	**81**		1	**81**		1
R100_1g	21*	**21**	22.1	812	**21**	21.2	1591	23	23.1	8	**21**	21.1	899	**21**	22.1	0	**21**		480	**21**		8	**21**		4
R100_5gb	220	**220**	223	1970	**220**	220.7	1924	**220**		72	**220**		92	**220**	220.7	87	**220**		189	**220**		5	**220**		7
R100_5g	59	**59**		358	**59**		42	**59**		0	**59**		3	**59**		17	**59**		26	**59**		0	**59**		3
R100_9gb	518*	519	524	10	**518**	518.2	1288	**518**	518.4	1275	**518**		312	**518**		0	**518**		0	**518**	518.8	1853	**518**		879
R100_9g	141*	**141**	144.6	0	**141**		102	**141**	141.1	1138	**141**		289	**141**		0	**141**		0	**141**		608	**141**		183
wap01a	545	653	664.5	3413	676	680.9	792	636	645.9	3392	671	678.6	2068	563	688.4	252	561	628.6	1728	**548**	**552**	3263	584	590.9	484
wap02a	538	658	666.2	3416	665	671.6	1440	633	642.9	3450	666	671.9	2925	552	586.5	589	548	559.8	444	**540**	**542.7**	2658	579	584	630
wap03a	562	767	788	0	759	770.8	1104	767	787.8	0	765	770.1	3496	**569**	**573.1**	2893	572	573.8	2738	576	578.6	3102	699	711.2	2714
wap04a	563	773	792.9	0	773	779.6	1782	773	792.6	0	772	778.2	396	**564**	572.5	3280	567	**569.9**	2949	569	574.6	3529	710	727.1	297
wap05a	541	563	572.5	3378	614	626	2822	565	573.2	2213	595	599.9	1275	543	544.5	970	543	543.8	1870	**541**	**543.4**	3306	548	549.2	1499
wap06a	516	559	567.2	2859	612	621.3	774	542	547.5	2550	590	595	1458	**518**	590.4	1005	520	**522**	1662	519	522.4	1123	529	531.1	2358
wap07a	555	670	683.2	3584	719	725.2	2555	641	650.1	3513	711	720	3500	729	745.7	0	**560**	725.8	1384	564	**569.5**	2680	607	610.1	2381
wap08a	529	654	665.3	3402	692	697.6	1620	627	634	3456	680	692.2	1584	**536**	613.9	3035	538	**541**	716	543	549.5	3585	572	581.1	2308
p40	4984*	**4984**	4986.9	64	**4984**	4984.4	907	4985	5029.2	0	**4984**		73	4987	5055.7	0	**4984**	4985	1590	**4984**		0	**4984**		0
p41	2688*	**2688**	2707.1	1225	**2688**	2694.2	2634	2713	2764.6	0	**2688**	2689.8	2040	2718	2787.3	0	**2688**	2689.2	1824	**2688**		0	**2688**		0
p42	2466*	**2466**	2474.8	1180	2471	2484.4	2391	2475	2532.6	0	**2466**	2475.9	786	**2466**	2522.5	0	**2466**		969	**2466**		4	**2466**		4
r28	9407*	9415	9511.4	1819	9462	9508.6	3120	9460	9583.2	0	9411	9420.4	78	9410	9563	45	**9407**	9409.1	1363	**9407**		8	**9407**		5
r29	8693*	8715	8799	379	8816	8957.9	1554	8768	8894.9	0	8707	8743.4	2454	8696	8817.6	0	**8693**	8694.4	1693	**8693**		1	**8693**		1
r30	9816*	9826	9960	2378	9820	9840	504	9819	9871.4	0	**9816**	9816.8	1517	9836	9988.2	1	9819	9832.8	2867	**9816**		5	**9816**		7
#BKS		114/188			115/188			99/188			132/188			112/188			150/188			159/188			155/188		
#Best		114/188			115/188			99/188			132/188			128/188			158/188			164/188			157/188		
#Best Avg		45/188			97/188			53/188			98/188			41/188			129/188			161/188			152/188		

According to Table 6, compared to the other local search procedures, RedLS achieves more heterogeneous results, with the average score often far from the best score. This is due to the fact that RedLS is a local search algorithm that strongly favors intensification at the expense of exploration, and can therefore get stuck in a local minimal trap without being able to get out of it. With the help of MCTS, the combination of fast RedLS local optima searches and frequent restarts at each iteration from different good starting points suggested by MCTS, results in a very robust and efficient algorithm. This MCTS+RedLS variant is able to find the best score more often than RedLS alone, and is more stable in terms of average results.

6 Conclusions

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.

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 example, an interesting study would be to automatically choose the balance coefficient between exploitation and exploration on the fly when solving each specific instance. It could also 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

We would like to thank Dr. Wen Sun [5], Dr. Yiyuan Wang, [6] and Pr. Bruno Nogueira [4] for sharing their codes. We also thank the organizers and editors of Evostar 2022 to give us the opportunity to present this extended version of our conference paper [1] for this Topical Issue of the SN Computer Science journal. We thank the reviewer for the useful comments, which helped us to improve the work. This work was granted access to the HPC resources of IDRIS (Grant No. 2020-A0090611887) from GENCI and the Centre Régional de Calcul Intensif des Pays de la Loire (CCIPL).

Authors' contributions

C. Grelier developed the code, prepared the reduced instances, and performed the tests. O. Goudet and J.K. Hao planned and supervised the work. All the authors contributed to the analysis and the writing of the manuscript.

Conflict of Interest

The authors declare that they have no conflicts of interest.

Availability of data set, code, and results

The instances used in this work, in reduced version, are available at https://github.com/Cyril-Grelier/gc_instances. The source code of the tested methods and their detailed results are available at https://github.com/Cyril-Grelier/gc_wvcp_adaptive_mcts.

References

[1] Grelier, C., Goudet, O., Hao, J.-K.: On monte carlo tree search for weighted vertex coloring. In: Pérez Cáceres, L., Verel, S. (eds.) Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science, vol. 13222, pp. 1–16 (2022)

[2] Lewis, R.: Guide to Graph Colouring: Algorithms and Applications. Springer, (2021)

[3] Goudet, O., Grelier, C., Hao, J.-K.: A deep learning guided memetic framework for graph coloring problems. arXiv:2109.05948 [cs] (2021)

[4] Nogueira, B., Tavares, E., Maciel, P.: Iterated local search with tabu search for the weighted vertex coloring problem. Computers & Operations Research **125**, 105087 (2021)

[5] Sun, W., Hao, J.-K., Lai, X., Wu, Q.: Adaptive feasible and infeasible tabu search for weighted vertex coloring. Information Sciences **466**, 203–219 (2018)

[6] Wang, S. Yiyuanand Cai, Pan, S., Li, X., Yin, M.: Reduction and local search for weighted graph coloring problem. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020, pp. 2433–2441 (2020)

[7] 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 (2000)

[8] 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. Lecture Notes in Computer Science, pp. 1064–1075 (2005)

[9] Furini, F., Malaguti, E.: Exact weighted vertex coloring via branch-and-price. Discrete Optimization **9**(2), 130–136 (2012)

[10] Malaguti, E., Monaci, M., Toth, P.: Models and heuristic algorithms for a weighted vertex coloring problem. Journal of Heuristics **15**(5), 503–526

(2009)

[11] Cornaz, D., Furini, F., Malaguti, E.: Solving vertex coloring problems as maximum weight stable set problems. Discrete Applied Mathematics **217**, 151–162 (2017)

[12] Goudet, O., Grelier, C., Lesaint, D.: New bounds and constraint programming models for the weighted vertex coloring problem. Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China, 1927–1934 (2023)

[13] Grelier, C., Goudet, O., Hao, J.-K.: A memetic algorithm with adaptive operator selection for graph coloring. In: Stützle, T., Wagner, M. (eds.) Evolutionary Computation in Combinatorial Optimization, pp. 65–80. Springer, Cham (2024)

[14] Zhou, Y., Hao, J.-K., Duval, B.: Reinforcement learning based local search for grouping problems: A case study on graph coloring. Expert Systems with Applications **64**, 412–422 (2016)

[15] Zhou, Y., Duval, B., Hao, J.-K.: Improving probability learning based local search for graph coloring. Applied Soft Computing **65**, 542–553 (2018)

[16] Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of uct with patterns in monte-carlo go. Technical report, RR-6062, INRIA (2006)

[17] 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)

[18] 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 (2016)

[19] Edelkamp, S., Greulich, C.: Solving physical traveling salesman problems with policy adaptation. In: 2014 IEEE Conference on Computational Intelligence and Games, pp. 1–8 (2014). IEEE

[20] Jooken, J., Leyman, P., Wauters, T., De Causmaecker, P.: Exploring search space trees using an adapted version of Monte Carlo tree search for combinatorial optimization problems. Computers & Operations Research **150**, 106070 (2023)

[21] Cazenave, T., Negrevergne, B., Sikora, F.: Monte carlo graph coloring. In: Monte Carlo Search 2020, IJCAI Workshop (2020)

[22] Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM **22**(4), 251–256 (1979)

[23] Kubale, M., Jackowski, B.: A generalized implicit enumeration algorithm for graph coloring. Communications of the ACM **28**(4), 412–418 (1985)

[24] Demange, M., Werra, D.d., Monnot, J., Paschos, V.T.: Time slot scheduling of compatible jobs. Journal of Scheduling **10**(2), 111–127 (2007)

[25] Lai, T.L., Robbins, H.: Asymptotically efficient adaptive allocation rules. Advances in Applied Mathematics **6**(1), 4–22 (1985)

[26] Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Computing **39**, 345–351 (1987)

[27] 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)