A Diversity Algorithm of Nested Rollout Policy Adaptation for Graph Coloring

Wenzhu Yang, Jingwen Li and Li Wang

***Abstract*—The Graph Coloring Problem is a well-known NP-hard problem. Over the years, numerous scholars have been pursuing efficient algorithms to obtain high-quality solutions. Nested Rollout Policy Adaptation (NRPA) is a Monte Carlo Tree Search algorithm for single-player games, and it has been proven effective and good in combinatorial optimization problems. In this paper, we use for the first time the NRPA algorithm combined with the destruction and reconstruction ideas of the Iterative Greedy algorithm to solve the Graph Coloring Problem. First, the basic principle of the NRPA algorithm is introduced. Then, NRPA is extended by using mixed sorting, destruction and reconstruction, and the Diversity-NRPA algorithm is proposed, which improves the diversity of the algorithm. Finally, Diversity-NRPA is applied to solve the Graph Coloring Problem by combining it with the knowledge of the graph theory field. We evaluate the performance of Diversity-NRPA on DIMACS, a well-known graph benchmark instance, and compare it with traditional graph coloring algorithms. The experimental results show that the Diversity-NRPA algorithm can achieve excellent performance in both solution quality and search efficiency in solving the Graph Coloring problem.**

***Index Terms*—Combinatorial optimization, Destruction and reconstruction, Graph coloring, Nested rollout policy adaptation**

I. Introduction

THE Graph Coloring Problem (GCP) can be described as follows: given an undirected graph $G = (V, E)$, which has a set of vertices V and a set of edges E, the GCP is to color the vertices in V, so that two adjacent vertices have different colors while using the minimum number of colors (called the chromatic number of G, denoted by $\chi(G)$). The problem can also be expressed as partitioning the vertex set V into a minimum number of color groups while satisfying that two vertices connected by an edge belong to different color groups, as shown in formula (1):

$$S = \{\{V_1, V_2, ..., V_k\} : \bigcup_{i=1}^{k} V_i = V, V_i \bigcap V_j = \phi\} \quad (1)$$

where k denotes the number of color groups, i, j denote the indexes of the colors and $i \neq j$, $1 \leqslant i, j \leqslant k$, $1 \leqslant k \leqslant |V|$, and S denotes the search space for graph coloring.

Over the past few decades, GCP has been intensively studied, mainly in two directions: exact methods and heuristic methods. Exact methods for solving GCP, such as the implicit enumeration algorithm [1] proposed by Brown in 1972, and the Branch and Price algorithm [2] based on the VCP-SC formulation proposed by Mehrotra and Trick in 1996, ect. However, these exact methods cannot solve instances with hundreds of vertices [3]. Since most GCP is NP-hard, using exact methods requires prohibitively long search times. Thus exact methods fail on the coloring problem of large instances [3].

To handle large instances, scholars have introduced many heuristic algorithms to approximately solve the problem, such as constructive methods [4, 5] in the 1960s and 1970s, local search meta-heuristics [6, 7] in the 1980s and 1990s, and genetic-based local search methods [8, 9] in the 1990s. These algorithms usually use greedy heuristics to construct initial solutions and improve the current solution by considering the best move in a given neighborhood. Among the many algorithms, Satisfiability (SAT) algorithm [10] is often used to solve GCP and performs well. The algorithm is based on Boolean logic, by assigning values to the variables of the Boolean formula one by one to find the solution that satisfies the formula. It is mainly used to solve Boolean satisfiability problems. In 2005, Méndez-Díaz [11] improved the classic Degree of Saturation (DSatur) algorithm based on Brélaz [4] by adding dynamic adjustment of the color order, parallel computing, etc., which effectively improved the performance of the algorithm to solve the GCP. However, for difficult graph coloring instances (NP-?) [12], the heuristic algorithms, due to their characteristic of single-trajectory local search, often lack diversification ability, making it difficult to find high-quality solutions.

Recently, Monte Carlo Tree Search (MCTS) [13] has been used to solve combinatorial optimization problems and has proven effective for many such problems. For example, it has been used to solve problems such as the Traveling Salesman Problem with Time Windows [14, 15], Crosswords Puzzles [16], and Morpion Solitaire. It has set new records in solving the Morpion Solitaire and Crosswords Puzzles problems. Compared to traditional algorithms, there is little literature on using MCTS to solve GCP (except for [17, 18]). At the same time, algorithms based on MCTS naturally combine random search with tree search, a feature that makes it possible for MCTS to solve GCP.

In this paper, we first address the problem of the NRPA algorithm [16] in MCTS that tends to converge to local optimal solutions. We propose two methods to improve the diversity of the NRPA algorithm and name the improved

Manuscript received March 28, 2023; revised September 6, 2023.

This work was supported by the National Natural Science Foundation of China (Grant No. 11961041; Grant No. 6206249), the Gansu Provincial Science and Technology Plan Project (Grant No. 21ZD8RA008) and Excellent Postgraduate Innovation Star Project of Gansu Province (Grant No. 2023CXZX-55).

Wenzhu Yang is a postgraduate student of School of Electronic and Information Engineering in Lanzhou Jiaotong University, Lanzhou, 730070, China. (e-mail: 3224408644@qq.com).

Jingwen Li is a professor of School of Electronics and Information Engineering in Lanzhou Jiaotong University, Lanzhou, 730070, China (corresponding author; lijingwen28@163.com).

Li Wang is a postgraduate student of School of Electronic and Information Engineering in Lanzhou Jiaotong University, Lanzhou, 730070, China. (e-mail: 1175133725@qq.com).

algorithm as Diversity-NRPA. Then, based on the Diversity-NRPA algorithm, the algorithm is adjusted by combining the specialized knowledge in graph theory to solve the problem that it is challenging to color the difficult instances of graph instances in GCP. Finally, Diversity-NRPA is compared with the NRPA algorithm as well as the traditional and popular heuristic algorithms: SAT(Satisfiability) and Méndez-Díaz's improved DSatur on the DIMACS graph benchmark instance [12] to evaluate its performance.

The organization of the rest of the paper is as follows. Section II introduces the NRPA algorithm based on MCTS. Section III elaborates on Diversity-NRPA. Section IV adapts Diversity-NRPA to make it applicable to GCP by incorporating graph-theoretic domain knowledge. Section V validates the performance of Diversity-NRPA on the DIMACS graph benchmark instance.

II. The Nested Rollout Policy Adaptation Algorithm

NRPA is a learn a playout policy based on Monte Carlo Tree Search (MCTS) proposed by Rosin and given in Algorithm 1. This algorithm has been used to improve MCTS-based Go programs [19] and other programs for combinatorial optimization problems [17], achieving great success.

The algorithm consists of two parts, an adaptive rollout policy and a nested structure. In the adaptive rollout policy, a set of weights represents the probability of each possible move in the game. Firstly, the weights of the policy are randomly initialized. Then the solution is obtained by using the policy in the playout algorithm, wherein the policy will make the solution move closer to the move with a larger weight. This process is repeated for *N* iterations, with each iteration using the current best sequence of solutions to adjust the policy in the adaptive function.

Algorithm 1 The NRPA algorithm

Input: *level*, *policy*
Output: (*best-score, best-squence*)

```
function NRPA(level, policy)
    if level == 0 then
        (score, sequence) ← playout(initial-state, policy)
        return (score, sequence)
    else
        best-score ← -∞
        best-sequence ← [ ]
        for N iterations do
            (score, sequence) ← NRPA(level - 1, policy)
            if score ≥ best-score then
                best-score ← score
                best-sequence ← sequence
            end if
            policy ← Adapt(policy, best-sequence, 1.0)
        end for
        return (best-score, best-sequence)
    end if
end function
```

The playout algorithm uses Gibbs sampling to select the probability proportional to the exponent of its weight as a legal move, as described in Algorithm 2. Finally, in the adaptive function, α is used to increase the weights of the moves in the best sequence and to decrease the weights of the other legal moves according to the value proportional to the exponent of the weights, as described in Algorithm 3. where α takes the value of 1 is verified to be a good value.

Algorithm 2 The playout algorithm

Input: *state*, *policy*
Output: (*score, squence*)

```
function playout(policy, sequence)
    sequence ← [ ]
    while state ≠ terminal-state do
        z ← 0.0
        for m in legal-moves(state) do
            z ← z + exp(policy[code(m)])
        end for
        choose move with probability exp(policy[code(move)]) / z
        state ← play(state, move)
        sequence ← sequence + move
    end while
    return (score(state), sequence)
end function
```

Algorithm 3 The Adapt algorithm

Input: *policy*, *sequence*, α
Output: *new-policy*

```
function Adapt(policy, sequence, α)
    new-policy ← policy
    state ←initial-state
    for move in sequence do
        new-policy [code(move)] ← new-policy [code(move)] + α
        z ← 0.0
        for m in legal-moves(state) do
            z ← z + exp(policy[code(m)])
        end for
        for m in legal-moves(state) do
            new-policy[code(m)] ← new-policy[code(m)]-
α * exp(policy[code(m)]) / z
        end for
        state ← play(state, move)
    end for
    return new-policy
end function
```

III. Diversity-NRPA

Since NRPA is an algorithm with a self-learning policy, it relies on the best solution obtained from the current iteration to adjust the policy so that the solution is constantly close to the optimal solution. Therefore, if the solution sequence generated by the algorithm in the first iteration is not optimal, the algorithm still adjusts its policy based on the current solution sequence, and rollouts are pushed toward this sequence. For the same reason, this behavior will continue to propagate, causing the solutions obtained by the algorithm to gradually deviate from the optimal solution and ultimately lead the algorithm to fall into a local optimum.

To address this issue, we propose two improvement methods: (1) Sorting nodes using DSatur meta-heuristic [20, 21, 22] combined with random sort, and (2) inspired by the Iterative Greedy algorithm, a destruction and reconstruction operation is added at NRPA algorithm *level*=0. This enables the algorithm to better balance exploration and exploitation, and generate more diverse solutions, improving algorithm's search efficiency and quality.

A. Mixed Sorting

Since, in the NRPA algorithm, a large number of legal moves will make the training of samples more difficult, it is particularly important to reduce the number of legal moves. If every possible move is considered at every step, the number of moves will reach $|V|\times|C|$, making the search inefficient. For this reason, we consider each moving vertex in a specific order, so that the maximum number of moves is reduced from $|V|\times|C|$ to $|C|$, which improves the execution efficiency of the algorithm.

For the sorting of vertices, we combine the heuristic algorithm DSatur and random sorting to sort the vertices, where the random sorting can help prevent and break the loop that may occur when using DSatur. As shown in Algorithm 4, during the iterative process of coloring the graph using the NRPA algorithm, the DSatur algorithm and random sorting are mixed in a fixed proportion to prevent algorithm stagnation. Experimental results show that the convergence speed of NRPA can also be effectively improved by mixing random sorting in the DSatur algorithm.

B. Destruction and Reconstruction in NRPA

To address the problem that the NRPA algorithm is easy to fall into the local optimum, we propose a simple and effective method of destroying and reconstructing the NRPA algorithm at *level*=0 to improve the diversity of the GCP solution process to effectively avoid the algorithm from falling into local optimum. The main feature of this algorithm is that it adopts a method that first destroys the current solution and then reconstructs it using the existing conditions. As shown in Algorithm 4, we add a destruction and reconstruction function after the completion of the playout() function. First, we use the playout algorithm of NRPA to generate an initial solution. Then a certain percentage of elements are randomly selected and removed from the solution to destroy the current solution structure. Then, the solution is reconstructed to obtain a new solution. Finally, the final solution is selected based on the acceptance criterion.

Algorithm 4 The Diversity-NRPA algorithm

Input: *level*, *policy*
Output: (*best-score, best-sqence*)

```
function Diversity-NRPA(level, policy)
    if level == 0 then
        (score, sequence) ← playout(initial-state, policy)
        (sequence-d) ← destruction(sequence)
        (sequence-d, score-d) ← construction(sequence-d, policy)
        if score-d ≥ score then
            score, sequence ← score-d, sequence-d
        end if
        return (score, sequence)
    else
        best-score ← -∞
        best-sequence ← [ ]
        for N iterations do
            (score, sequence) ← Diversity-NRPA(level - 1, policy)
            if score ≥ best-score then
                best-score ← score
                best-sequence ← sequence
            end if
            policy ← AdaptWeight(policy, best-sequence, 1.0)
        end for
        return (best-score, best-sequence)
    end if
end function
```

The above two methods for NRPA improvement can reduce rollouts being pushed into the error sequence, thus making the whole nested level less disturbed and improving the quality of the NRPA algorithm solution.

IV. A Diversity-NRPA Algorithm for GCP

In this section, based on the Diversity-NRPA algorithm in section III, we have adjusted the algorithm according to the professional knowledge in graph theory so that the algorithm can solve GCP. This process includes defining the legal moves for vertices, sorting, coloring, objective function, destruction and reconstruction, and adaptive adjustments.

Before coloring the vertices of the graph, it is necessary to define the possible legal moves of the vertices. A legal move refers to assigning a color from the set of colors to an uncolored vertex in the graph while satisfying the condition that adjacent vertices have different colors. That is, given a graph $G=(V, E)$, and a set of colors C, and a legal move can be represented as (v, c) where $v \in V$, $c \in C \setminus C_{N_v}$, N_v represents the set of neighboring vertices of v.

According to section III, in this paper, we use a method of alternating between DSatur and random sorting to sort the nodes. The working process of this method on the GCP is as follows: during the DSatur sorting phase, we prioritize selecting the vertex with the fewest available colors as the next coloring vertex. When there is more than one vertex with a minimum number of selectable colors, the vertex with the most neighbors is selected. Compared with the traditional pre-defined vertex sorting, this method has the advantage of propagating GCP constraints, so DSatur is a good heuristic method for the vertex sorting of Diversity-NRPA. During the random sorting phase, the coloring order of the vertices in the graph is randomly determined. Compared to using DSatur sorting alone, random sorting helps prevent and break the possible loops that may occur when using DSatur. By mixing these two sorting methods, we not only achieve specific sorting of legal moves to reduce the number of moves but also achieve the goal of preventing the algorithm from stalling.

In the process of selecting colors for the vertices in the graph, we first delete the colors used by its adjacent vertices from the possible color set of the current dyed vertex to prevent the algorithm from selecting colors that have been assigned to adjacent vertices. However, if adjacent vertices of the current vertex have used all available colors, the current vertex will have no available colors. In this case, all colors can be used as possible colorings for the current vertex, even if it does not satisfy the constraints of the GCP.

In this paper, we establish an objective function based on the constraint that adjacent vertices in GCP must have different colors. That is, the initial total score is $2|E|$, if two adjacent vertices in the graph have the same color, the total score is reduced by 2. When the score of the scoring function is equal to two times the number of edges in the currently colored graph, a solution has been found.

As indicated in section III, in this paper, we use the method of destruction and reconstruction to diversify the solutions obtained. We apply the destruction phase to the coloring sequence in the solution obtained by the playout() function to break the existing balance. It should be noted that the solution

contains the coloring sequence and its corresponding score. The coloring sequence contains the coloring of n vertices (let n be the number of vertices in the current graph). It randomly selects a certain proportion of vertices (let the number of selected vertex nodes be d) and deletes the color on the selected vertices. The result of this process is two subsequences. The first is a partial sequence π_D with n-d colored vertices, i.e., the sequence after the removal of d vertex colors; the second is the sequence consisting of the d vertices of the deleted color, which we denote as π_R . π_R contains the vertices in π_D that must be recolored. The reconstruction phase starts with the sub-sequence π_D , and the uncolored vertices in this sequence are recolored using the same coloring scheme until all the vertices in π_R are recolored, thereby obtaining a complete solution. Finally, according to the acceptance criterion, the solutions obtained in the playout and the solutions after the destruction and reconstruction are selected, and the best one is chosen as the final solution. The acceptance criterion we set is based on the solution's score, and higher-scoring solutions are retained.

In the NRPA algorithm, the policy of the algorithm is adjusted using an adaptive function based on the solutions found so far, which makes the algorithm keep moving closer to the optimal solution. As shown in Algorithm 5, the Diversity-NRPA algorithm uses an adaptive function to modify the weight values of each color in the set of colors selected by the vertex. Among them, we modify the weight value of all legally moved colors.

Algorithm 5 The AdaptWeight algorithm

Input: *policy*, *sequence*, α
Output: *new-policy*

function AdaptWeight(*policy*, *sequence*, α)
  new-policy ← *policy*
  state ←*initial-state*
  for *move* in *sequence* **do**
    new-policy [*move*] ← *new-policy* [*move*] + α
    $z \leftarrow 0.0$
    for *m* in legal-moves(*state*) **do**
      $z \leftarrow z + \exp(policy[m])$
    end for
    for *m* in legal-moves(*state*) **do**
      $new\text{-}policy[m] \leftarrow new\text{-}policy[m] - \alpha * \frac{exp(policy[m])}{z}$
    end for
    state ← play(*state*, *move*)
  end for
  return *new-policy*
end function

V. Experimentation

This section aims to experimentally verify the impact of diversity processing on the NRPA algorithm and the performance of our designed Diversity-NRPA algorithm compared to traditional graph coloring algorithms on GCP.

A. Benchmark Instances and Implement

We tested the algorithm on the DIMACS graph benchmark instances, which are widely used to evaluate graph coloring algorithms and are considered a standard set for experimental research in this field. Furthermore, since most of these instances have been extensively researched, the optimal chromatic number χ for most of them is known. These instances in DIMACS are classified by difficulty, and roughly divided into three categories: easy graphs (NP-m), medium graphs (NP-h), and hard graphs (NP-?), where NP-? indicates that the optimal chromatic number χ for this instance is unknown or the time required to obtain it is unknown.

Since we are concerned with decision problems, i.e., determining whether a graph can be colored with a given number of colors. Therefore, it is necessary to set a termination condition for the algorithm. At the same time, to test the algorithm's stability, it is necessary to execute the algorithm several times under the condition of a given number of colors k. We set different time limits and the number of executions of the algorithm based on the literature data and the difficulty of solving the graph. Specifically, for simple NP-m instances, the algorithm is executed twice, each time limited to 20 minutes; for medium NP-h instances, the algorithm is executed five times, each time limited to 30 minutes; for the relatively easy graph instance in the difficult NP-? instance, the algorithm is executed six times in total, and the time limit for each time is 1 hour; for the difficult graph instance in the difficult NP-? instance, the algorithm is executed four times, and the time limit for each time is 2 hours. Among them, if the algorithm finds a valid K-coloring for the current graph instance within the corresponding time limit, the value of k is subtracted by one. The process is repeated until the termination condition is reached. The final value of k obtained is the smallest number of colorings found by the current algorithm on the current instance, and the current coloring scheme is the best solution. Where the simple greedy algorithm determines the initial value of k, and the vertices are sorted according to the method described in Section III. In mixed sorting, DSatur sorting and random sorting are set at a ratio of 5:1.

The experiments reported in this paper were conducted on a computer equipped with an Intel i7 12700k CPU. The Diversity-NRPA algorithm in Section III, the graph coloring algorithm in Section IV, and the SAT, DSatur, and NRPA algorithms used in the experiments were all implemented in Python and initialized with a simple greedy algorithm. We set the nested level to *level*=7 and the iteration number to N=100 for the NRPA and Diversity-NRPA algorithms.

We report the following statistics in the experimental results table. The BKS (Best-Known Score) column reports the best chromatic number χ that is widely recognized for the current instance. Note that these results were obtained using different algorithms, computational tools, CPUs, or GPUs and also under loose conditions (e.g., sufficiently long runtimes). When a χ is obtained for an experimental method, it is marked in bold in the table. The column "Best" represents the best result of the current method, where "unk" indicates that the current algorithm has not obtained a result under the current experimental conditions, and "Reached" indicates the proportion of the current method that reaches the best result of the algorithm in multiple solutions, which is used to test the stability of the algorithm.

Due to space limitations, we have provided partial experimental results for different types of instances in the test cases (26 in total). Among these, the very difficult instance (NP-?) in the experimental results table (Table IV, Table VIII)

comes from the recognized most difficult instance.

B. Compare NRPA with Diversity-NRPA in Graph Coloring

Tables I, II, III, and IV show the experimental results of the Diversity-NRPA and the NRPA algorithm on DIMACS graph instances with different solving difficulties. In the table, column 1 is the name of the graph instance, column 2 is the number of vertices corresponding to the graph instance, column 3 is the number of edges corresponding to the graph instance, and column 5 corresponds to the initial solution obtained by using a simple greedy algorithm. Columns 6-7 are the results obtained by the NRPA algorithm in the corresponding graph instance, and columns 8-17 correspond to the results obtained by the Diversity-NRPA algorithm at different destruction ratios. The last row shows the percentage of the current algorithm that achieves χ on all instances in the current table.

Combining the results in Tables I, II, III, and IV, it can be observed that the Diversity-NRPA algorithm has better performance than the NRPA algorithm in the coloring results of the graph instances and the percentage of the coloring results that reach the χ value. At the same time, it can also be noticed that under different destruction ratios, the performance of the Diversity-NRPA algorithm for graph instances of different difficulties is also different. For easy NP-m instances (Table I), the Diversity-NRPA algorithm performs the worst when the destruction ratio is 50%, and the results of other destruction ratios are the same and consistent

TABLE I
THE RESULTS OF NRPA AND DIVERSITY NRPA WITH DIFFERENT DESTRUCTION RATIOS ON EASY INSTANCES

Instance	|V|	|E|	BKS χ	Greedy Initial K	NRPA		Diversity-NRPA (30%)		Diversity-NRPA (40%)		Diversity-NRPA (50%)		Diversity-NRPA (60%)		Diversity-NRPA (70%)	
					Best	Reached	Best	Reached	Best	Reached	Best	Reached	Best	Reached	Best	Reached
le450_15a	450	8168	15	17	**15**	100%	**15**	100%	**15**	100%	**15**	100%	**15**	100%	**15**	100%
myciel6	95	755	7	7	**7**	100%	**7**	100%	**7**	100%	8	100%	**7**	100%	**7**	100%
wap05a	905	43081	50	50	**50**	100%	**50**	100%	**50**	100%	**50**	100%	**50**	100%	**50**	100%
mug100_25	100	166	4	4	**4**	100%	**4**	100%	**4**	100%	**4**	100%	**4**	100%	**4**	100%
Reach a ratio of χ					4/4		4/4		4/4		3/4		4/4		4/4	

TABLE II
THE RESULTS OF NRPA AND DIVERSITY NRPA WITH DIFFERENT DESTRUCTION RATIOS ON MEDIUM INSTANCES

Instance	|V|	|E|	BKS χ	Greedy Initial K	NRPA		Diversity-NRPA (30%)		Diversity-NRPA (40%)		Diversity-NRPA (50%)		Diversity-NRPA (60%)		Diversity-NRPA (70%)	
					Best	Reached	Best	Reached	Best	Reached	Best	Reached	Best	Reached	Best	Reached
DSJC125.5	125	3891	17	23	18	100%	**17**	100%	**17**	40%	**17**	20%	**17**	20%	18	100%
DSJC125.9	125	6961	44	50	**44**	100%	**44**	100%	**44**	100%	**44**	100%	**44**	40%	45	100%
DSJC250.9	250	27897	72	90	76	20%	75	20%	76	100%	76	80%	76	40%	77	100%
DSJR500.1c	500	121275	85	88	87	60%	**85**	80%	86	20%	86	20%	86	20%	87	60%
DSJR500.5	500	58862	122	231	**122**	40%	**122**	100%	**122**	100%	**122**	40%	**122**	100%	123	100%
flat300_28_0	300	21695	28	41	35	20%	33	100%	32	20%	33	100%	33	80%	35	100%
Reach a ratio of χ					2/6		4/6		3/6		3/6		3/6		0/6	

TABLE III
THE RESULTS OF NRPA AND DIVERSITY NRPA WITH DIFFERENT DESTRUCTION RATIOS ON DIFFICULT INSTANCES

Instance	|V|	|E|	BKS χ	Greedy Initial K	NRPA		Diversity-NRPA (30%)		Diversity-NRPA (40%)		Diversity-NRPA (50%)		Diversity-NRPA (60%)		Diversity-NRPA (70%)	
					Best	Reached	Best	Reached	Best	Reached	Best	Reached	Best	Reached	Best	Reached
le450_5a	450	5714	5	10	**5**	100%	**5**	100%	**5**	100%	**5**	100%	**5**	50%	6	100%
le450_5b	450	5734	5	7	**5**	50%	**5**	50%	**5**	100%	**5**	50%	**5**	50%	**5**	33%
le450_15c	450	16680	15	17	**15**	100%	**15**	100%	**15**	100%	**15**	100%	**15**	100%	16	100%
le450_25d	450	17425	25	29	26	100%	**25**	33%	**25**	50%	**25**	50%	**25**	33%	26	50%
le450_25c	450	17343	25	29	26	100%	26	33%	**25**	33%	26	100%	26	100%	26	33%
qg.order60	3600	212400	60	63	62	100%	62	100%	62	100%	62	100%	62	50%	63	100%
Reach a ratio of χ					3/6		4/6		5/6		4/6		4/6		1/6	

TABLE IV
THE RESULTS OF NRPA AND DIVERSITY NRPA WITH DIFFERENT DESTRUCTION RATIOS ON VERY DIFFICULT INSTANCES

Instance	|V|	|E|	BKS χ	Greedy Initial K	NRPA		Diversity-NRPA (30%)		Diversity-NRPA (40%)		Diversity-NRPA (50%)		Diversity-NRPA (60%)		Diversity-NRPA (70%)	
					Best	Reached	Best	Reached	Best	Reached	Best	Reached	Best	Reached	Best	Reached
DSJC250.1	250	3218	4	10	8	100%	7	25%	7	100%	7	25%	8	50%	9	100%
DSJC250.5	250	15668	26	37	32	100%	32	100%	32	100%	32	100%	32	100%	32	100%
flat1000_50_0	1000	245000	15	113	111	50%	90	50%	90	75%	90	50%	91	75%	91	75%
flat1000_60_0	1000	245830	14	112	112	100%	112	100%	112	100%	112	100%	112	100%	112	50%
flat1000_76_0	1000	246708	14	115	110	50%	106	50%	105	25%	106	25%	107	50%	110	25%
wap01a	2368	110871	41	47	45	50%	45	50%	45	100%	45	50%	45	25%	46	100%
wap02a	2464	111742	40	46	45	100%	43	100%	42	25%	43	100%	42	25%	45	25%
DSJC500.5	500	62624	43	65	59	75%	51	100%	49	75%	51	100%	51	50%	53	100%
DSJC1000.5	1000	249826	73	114	112	50%	89	25%	89	25%	89	25%	91	100%	98	100%
qg.order100	10000	990000	100	106	102	50%	102	100%	102	100%	102	100%	102	100%	102	100%
Reach a ratio of χ					0/10		0/10		0/10		0/10		0/10		0/10	

with those of the NRPA algorithm; for medium NP-h instances (Table II), the Diversity-NRPA algorithm performs the best when the destruction ratio is 30%, and it performs the worst and yields experimental results inferior to the NRPA algorithm when the destruction ratio is 70%. Under other destruction ratios, the experimental results of the Diversity-NRPA algorithm are similar and all superior to the NRPA algorithm. For the difficult NP-? instances (Tables III, VI), the Diversity-NRPA algorithm performs best when the destruction ratio is 40%, and the experimental results are significantly better than the NRPA algorithm. From the above experimental results, it can be concluded that the Diversity-NRPA algorithm outperforms the NRPA algorithm in all other cases, except for the case where the destruction ratio is 70%.

In order to further explore the reason why the experimental results of the Diversity-NRPA algorithm are inferior to the NRPA algorithm when the destruction ratio is 70%, we canceled the restrictions on the running time and execution times for all instances. And only the graph instances in which the Diversity-NRPA algorithm results are inferior to the NRPA algorithm when the destruction ratio is 70% in the above experimental results table are used for testing. The experimental results are shown in Figure 1. It can be seen that under the same number of iterations, the experimental results of the Diversity-NRPA algorithm are always equal to or better than those of the NRPA algorithm. As the number of iterations increases, the performance of the Diversity-NRPA algorithm has a stronger advantage. This is because as the algorithm's running time increases, the NRPA algorithm falls into a local optimum, and the Diversity-NRPA algorithm can effectively avoid falling into a local optimum due to its diversity.

Based on the above experiments, it can be seen that after adding destruction and reconstruction to the NRPA algorithm, the quality of the algorithm's solution is significantly improved. So, it is essential to introduce diversification into the NRPA algorithm. This method makes the algorithm better balance between exploration and exploitation in the search process, which improves the diversity of the NRPA algorithm and thus improves the quality of the algorithm solution. It can also be found that the Diversity-NRPA algorithm performs differently on different types of instances with different destruction ratios. As a whole, in our experiments, the algorithm performs best when the destruction ratio is 40%. For the situation in Tables II, III, and IV, where the Diversity-NRPA algorithm is inferior to the NRPA algorithm when the destruction ratio is 70%, this is because when the destruction ratio is too large, the execution time of the algorithm will be greatly increased. The execution efficiency of the algorithm will be reduced, which will lead to a reduction in the number of iterations performed by the algorithm within the same period of time, which will then affect the quality of the solution.

C. Compare Traditional Approaches with Diversity-NRPA in Graph Coloring

Tables V, VI, VII, and VIII report the experimental results

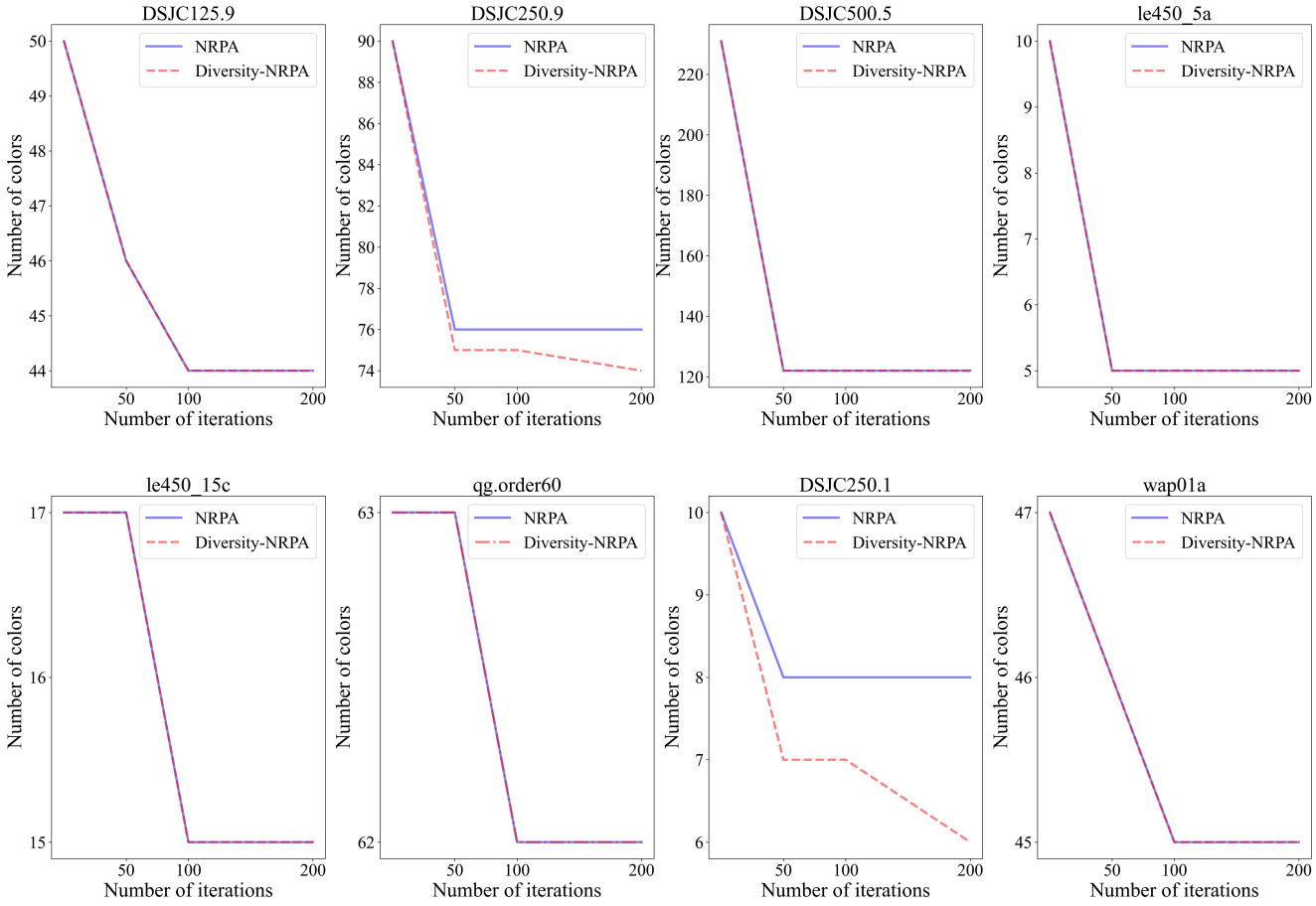

Fig. 1. Results of Diversity-NRPA (70%) and NRPA at different numbers of iterations. The experiments monitored the values of k for both algorithms only at iteration counts of 50, 100, and 200 without real-time data monitoring.

TABLE V
THE RESULTS OF DIVERSITY-NRPA, SAT, AND DSATUR ON EASY INSTANCES

Instance	|V|	|E|	BKS χ	Greedy Initial K	SAT(1)		Diversity-DSatur(2)		Diversity-NRPA (40%)(3)		Gap1	Gap2
					Best	Reached	Best	Reached	Best	Reached	(3)-(1)	(3)-(2)
le450_15a	450	8168	15	17	**15**	100%	**15**	100%	**15**	100%	0	0
myciel6	95	755	7	7	**7**	100%	**7**	100%	**7**	100%	0	0
wap05a	905	43081	50	50	**50**	100%	**50**	100%	**50**	100%	0	0
mug100_25	100	166	4	4	**4**	100%	**4**	100%	**4**	100%	0	0
Reach a ratio of χ					4/4		4/4		4/4			

TABLE VI
THE RESULTS OF DIVERSITY-NRPA, SAT, AND DSATUR ON MEDIUM INSTANCES

Instance	|V|	|E|	BKS χ	Greedy Initial K	SAT(1)		Diversity-DSatur(2)		Diversity-NRPA (40%)(3)		Gap1	Gap2
					Best	Reached	Best	Reached	Best	Reached	(3)-(1)	(3)-(2)
DSJC125.5	125	3891	17	23	19	100%	19	100%	**17**	20%	-2	-2
DSJC125.9	125	6961	44	50	45	100%	45	100%	**44**	100%	-1	-1
DSJC250.9	250	27897	72	90	86	100%	88	100%	76	100%	-10	-12
DSJR500.1c	500	121275	85	88	86	100%	87	100%	86	20%	-1	-2
DSJR500.5	500	58862	122	231	**122**	100%	123	100%	**122**	100%	0	-1
flat300_28_0	300	21695	28	41	33	100%	34	100%	32	20%	-1	-2
Reach a ratio of χ					1/6		0/6		3/6			

TABLE VII
THE RESULTS OF DIVERSITY-NRPA, SAT, AND DSATUR ON DIFFICULT INSTANCES

Instance	|V|	|E|	BKS χ	Greedy Initial K	SAT(1)		Diversity-DSatur(2)		Diversity-NRPA (40%)(3)		Gap1	Gap2
					Best	Reached	Best	Reached	Best	Reached	(3)-(1)	(3)-(2)
le450_5a	450	5714	5	10	**5**	100%	9	100%	**5**	100%	0	-4
le450_5b	450	5734	5	7	**5**	100%	9	100%	**5**	100%	0	-4
le450_15c	450	16680	15	17	**15**	100%	16	100%	**15**	100%	0	-1
le450_25d	450	17425	25	29	27	100%	28	100%	**25**	50%	-2	-3
le450_25c	450	17343	25	29	27	100%	28	100%	**25**	33%	-2	-3
qg.order60	3600	212400	60	63	61	100%	63	100%	62	100%	1	-1
Reach a ratio of χ					3/6		0/6		5/6			

TABLE VIII
THE RESULTS OF DIVERSITY-NRPA, SAT, AND DSATUR ON VERY DIFFICULT INSTANCES

Instance	|V|	|E|	BKS χ	Greedy Initial K	SAT(1)		Diversity-DSatur(2)		Diversity-NRPA (40%)(3)		Gap1	Gap2
					Best	Reached	Best	Reached	Best	Reached	(3)-(1)	(3)-(2)
DSJC250.1	250	3218	4	10	9	100%	9	100%	7	100%	-2	-2
DSJC250.5	250	15668	26	37	35	100%	36	100%	32	100%	-3	-4
flat1000_50_0	1000	245000	15	113	113	100%	113	100%	90	75%	-23	-23
flat1000_60_0	1000	245830	14	112	112	100%	112	100%	112	100%	0	0
flat1000_76_0	1000	246708	14	115	113	100%	114	100%	105	25%	-8	-9
wap01a	2368	110871	41	47	43	100%	46	100%	45	100%	2	0
wap02a	2464	111742	40	46	42	100%	43	100%	42	25%	0	-1
DSJC500.5	500	62624	43	65	63	100%	62	100%	49	75%	-14	-15
DSJC1000.5	1000	249826	73	114	114	100%	114	100%	89	25%	-25	-25
qg.order100	10000	990000	100	106	unk	unk	unk	unk	102	100%	unk	unk
Reach a ratio of χ					0/10		0/10		0/10			

of graph coloring using traditional algorithms SAT (columns 6-7), the saturation algorithm DSatur (columns 8-9), and the Diversity-NRPA algorithm (columns 10-11) on graphs of different difficulties from the DIMACS benchmark. In the table, Gap1 (column 12) shows the gap between the best experimental results of the SAT and Diversity-NRPA algorithms on the same instances, and Gap2 (column 13) shows the gap between the best experimental results of the DSatur and Diversity-NRPA algorithms on the same instances. Among them, according to the conclusion in Section B, the destruction ratio of the Diversity-NRPA algorithm is set to 40%.

As shown in Table V, all methods we discuss in this paper can handle these easy NP-m instances and find χ values. As shown in Table VI, the performance gap between the three algorithms on medium NP-h instances is small (Columns 12-13) (except on instance DSJC250.9). However, for the relatively easy graph instance in the difficult NP-? instance, the number of χ values found by the Diversity-NRPA algorithm is significantly more than that of the algorithm SAT and DSatur algorithms, as shown in Table VII. For the difficult graph instance in the difficult NP-? Instance, i.e., very difficult instances (NP-?), the Diversity-NRPA algorithm, under the current experimental conditions, not find the χ value. However, when the other two algorithms failed to find any results, the Diversity-NRPA algorithm found relatively good coloring results, especially on the instance qg.order100, as shown in Table VIII. This is due to

the adaptive function of the Diversity-NRPA algorithm that continuously improves the policy while exploring the search space. It also shows that the advantages of the Diversity-NRPA algorithm learning policy will be more obvious in the long-term operation of the algorithm, so it has outstanding experimental results for difficult NP-? instances (Table VII, Table VIII). Generally speaking, the Diversity-NRPA algorithm is significantly better than these two algorithms.

From the results of the experimental table in this section, we can conclude that the Diversity-NRPA algorithm has shown superior ability to the traditional graph coloring algorithm on the GCP problem, so it is worthy of further research and optimization. This further proves that using MCTS to solve combinatorial optimization problems has a good research prospect.

VI. Conclusion

In this work, we propose the Diversity-NRPA algorithm to address the problem of the NRPA algorithm getting trapped in the local optimum and successfully apply it to solve the GCP. The algorithm improves the diversity of the solution by introducing mixed sorting, destruction and reconstruction operations. Based on the Diversity-NRPA algorithm, we combine the domain knowledge of graph theory and apply it to solve the GCP problem. Compared with traditional algorithms, this algorithm shows superior performance. The following conclusions can be drawn from the experimental results: (1) Diversification of NRPA can effectively improve its performance, and when using destruction and reconstruction operations on the algorithm, the difference in the proportion of destruction will have different impacts on the algorithm. (2) Compared to other traditional graph coloring algorithms, using the Diversity-NRPA algorithm effectively improves the quality of solutions. Particularly for difficult instances (NP-?), the learning strategy of the Diversity-NRPA algorithm exhibits significant advantages.

In future work, other methods can be used to improve the problem that the NRPA algorithm is prone to fall into local optimum. Furthermore, the Diversity-NRPA algorithm can be applied to other variants of the Graph Coloring Problem, such as Weighted Vertex Coloring.

References

[1] Brown J. Randall, "Chromatic Scheduling and the Chromatic Number Problem," Management Science, vol. 19, no. 4, pp456-463, 1972.

[2] Mehrotra Anuj and Michael A. Trick, "A Column Generation Approach for Graph Coloring," Informs Journal on Computing, vol. 8, no. 4, pp344-354, 1996.

[3] Malaguti Enrico and Paolo Toth, "A Survey on Vertex Coloring Problems," International Transactions in Operational Research, vol. 17, no.1, pp1-34, 2010.

[4] Brélaz Daniel, "New Methods to Color the Vertices of a Graph," Communications of the ACM, vol. 22, no. 4, pp251-256, 1979.

[5] Leighton Frank Thomson, "A Graph Coloring Algorithm for Large Scheduling Problems," Journal of Research of the National Bureau of Standards, vol. 84, no. 6, pp489–506, 1979.

[6] Hertz Alain and D. De Werra, "Using Tabu Search Techniques for Graph Coloring," Computing, vol. 39, no. 4, pp345-351, 1987.

[7] Ferland J. and C. Fleurent, "Object-oriented Implementation of Heuristic Search Methods for Graph Coloring," Maximum Clique and Satisfiability, No. CONF-9408161-. Univ. of Michigan, Ann Arbor, MI (United States), 1994.

[8] Fleurent Charles and Jacques A. Ferland, "Genetic and Hybrid Algorithms for Graph Coloring," Annals of Operations Research, vol. 63, no. 1, pp437-461, 1996.

[9] Morgenstern Craig, "Distributed Coloration Neighborhood Search," No. CONF-9408161-. Univ. of Michigan, Ann Arbor, MI (United States), 1994.

[10] Alexey Ignatiev, Antonio Morgado and Joao Marques-Silva, "Cardinality Encodings for Graph Optimization Problems," International Joint Conference on Artificial Intelligence, Aug. 19-25, Melbourne, Australia, 2017.

[11] Isabel Méndez-Díaz and Paula Zabala, "A Branch-and-cut Algorithm for Graph Coloring," Discrete Applied Mathematics, vol. 154, no. 5, pp826-847, 2006.

[12] S. Gualandi and M. Chiarandini. (2019, Nov 17). Graph Coloring Benchmarks. Available: https://sites.google.com/site/graphcoloring/vertex-coloring.

[13] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas and Peter I. Cowling, etal, "A Survey of Monte Carlo Tree Search Methods," IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 1, pp1–43, 2012.

[14] Tristan Cazenave and Fabien Teytaud, "Application of the Nested Rollout Policy Adaptation Algorithm to the Traveling Salesman Problem with Time Windows," Learning and Intelligent Optimization: 6th International Conference, 16-20 January, 2015, Paris, France, pp42–54.

[15] Stefan Edelkamp, Max Gath, Tristan Cazenave and Fabien Teytaud, "Algorithm and Knowledge Engineering for the TSPTW Problem," 2013 IEEE Symposium on Computational Intelligence in Scheduling, 16-19 April, 2013, Singapore, pp44-51.

[16] Christopher D Rosin, "Nested rollout policy adaptation for Monte Carlo tree search," International Joint Conference on Artificial Intelligence, 16–22 July, 2011, Barcelona, Catalonia, Spain, pp 649–654.

[17] Stefan Edelkamp, Eike Externest, Sebastian Kühl, and Sabine Kuske, "Solving Graph Optimization Problems in A Framework for Monte-Carlo Search," International Symposium on Combinatorial Search, vol. 8, no. 1, pp163-164, 2021.

[18] Cazenave T, Negrevergne B, and Sikora F, "Monte Carlo graph coloring," Monte Carlo Search. MCS 2020. Communications in Computer and Information Science, vol 1379. Springer, Cham. https://doi.org/10.1007/978-3-030-89453-5_8.

[19] Graf, Tobias and Marco Platzner, "Adaptive Playouts in Monte-carlo Tree Search with Policy-gradient Reinforcement Learning," Advances in Computer Games. ACG 2015. Lecture Notes in Computer Science, vol 9525. Springer, Cham. https://doi.org/10.1007/978-3-319-27992-3_1.

[20] Ruiz, Rubén and Thomas Stützle, "A Simple and Effective Iterated Greedy Algorithm for the Permutation Flowshop Scheduling Problem," European Journal of Operational Research, vol. 177, no. 3, pp2033-2049, 2007.

[21] Culberson Joseph, "Iterated Greedy Graph Coloring and the Difficulty Landscape," University of Alberta, Department of Computing Science, 1992, https://books.google.com/books?id=hPs5ugAACAAJ.

[22] Stützle, Thomas and Rubén Ruiz, "Iterated Greedy," In: Martí, R., Pardalos, P., Resende, M. (eds) Handbook of Heuristics. Springer, Cham. https://doi.org/10.1007/978-3-319-07124-4_10.

Wenzhu Yang was born in Weinan, Shanxi Province, China in 1995. She received a bachelor's degree in engineering at Northwest Normal University in 2019 and now studying for a master's degree at Lanzhou Jiaotong University. Her research direction is graph theory algorithm and applications.

Jingwen Li was born in Shenyang, Liaoning Province, China in 1965. He is a Professor, master tutor. The research direction is graph theory algorithms and its applications. Since 2009, he has overseen or participated in the completion of three projects funded by the National Natural Science Foundation of China (two general projects); he has won two first prizes, one second prize, and three third prizes at the Provincial Science and Technology Progress Award; Over 100 academic papers have been published by him in journals such as "Discrete Mathematics", "Acta Mathematicae Applicatae Sinica", "Science in China Ser.A", "Ars Combinatoria", and are indexed by SCI, EI or ISTP. A total of more than 70 papers have been published in authoritative core journals,

including the "International Journal of Pure and Applied Mathematics.". He has served as a commentator for the American Mathematical Review since 2007.

Li Wang was born in Zigong, Sichuan Province, China in 1999. She received the bachelor's degree in engineering at Qiannan Normal University in 2021. Now she is studying for master's degree at Lanzhou Jiaotong University. Her research direction is graph theory algorithm and applications.