NERS_HEAD: An New Hybrid Evolutionary Algorithm for Solving Graph Coloring Problem

Ping Guo (guoping@cqu.edu.cn)
Chongqing University https://orcid.org/0000-0002-5239-8896
Bin Guo
Chongqing University

Research Article

Keywords: Graph coloring problem, Hybrid evolutionary algorithm, Tabu search, Combinatorial optimization

Posted Date: March 7th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-740991/v1

Noname manuscript No.
(will be inserted by the editor)

NERS_HEAD:An new hybrid evolutionary algorithm for solving graph coloring problem

Ping Guo · Bin Guo

Received: date / Accepted: date

Abstract The graph coloring problem is an NP-hard problem. Currently, the most effective method to solve the problem is the hybrid algorithm. This paper proposes a hybrid evolutionary algorithm NERS_HEAD with a new elite replacement strategy. In NERS_HEAD, a method to detect the local optimal state is proposed so that the evolutionary process can jump out of the local optimal by introducing diversity on time. A new elite structure and replacement strategy are designed to increase the diversity of the evolutionary population so that the evolution process can not only converge quickly but also jump out of the local optimum in time. Experiments based on 34 instances of the DIMACS benchmark show that compared with the current excellent graph coloring algorithm, NERS_HEAD can reduce the evolutionary generation by an average of 28.3% and the calculation time by 22.11%. Especially in the instance dsjc500.1, NERS_HEAD can reduce 56% of evolution generations and 40% of computing time; On the r1000.1c instance, it can reduce the evolution generations by 82.30% and the calculation time by 73.45%.

Keywords Graph coloring problem · Hybrid evolutionary algorithm · Tabu search · Combinatorial optimization

Ping Guo
College of Computer Science, Chongqing University, Chongqing 400044 ,China
Chongqing Key Laboratory of Software Theory and Technology, Chongqing 400044, China
E-mail: guoping@cqu.edu.cn

Bin Guo
College of Computer Science, Chongqing University, Chongqing 400044, China
E-mail: 15727653642@163.com

1 Introduction

The graph coloring problem (GCP) is a typical combinatorial optimization problem that has been applied to many issues, such as: printed circuit boards testing (Garey et al., 1975), frequency assignment in mobile radio telephone systems(Gamst, 1986), compiler register allocation(Briggs, 1995), noise reduction in Very Large Scale Integration (VLSI) Circuits(Woo et al., 2002) and resource allocation in the bus networks(Maitra et al., 2010), minimize the maximum vertex interference in wifi channel assignment(David et al., 2018), efficient reflection string analysis (Grech, 2018).

The GCP is a well-known NP-hard problem, and there are many approaches to solve it. The greedy heuristic is one of these methods: assign different colors to each vertex in a specific or random order of vertices and ensure that no conflicts occur, forming a K-coloring scheme step by step. Such a method is swift, but it does not give the minimum number of colors in most cases. Some efficient greedy heuristics are DSATUR (Bré and laz, 1979)and RLF (Leighton, 1979).

Local search algorithms are also widely used in GCP. A random scheme with conflicting edges is generated for a fixed number of colors (e.g., the chromatic number). Then the conflicting edges are reduced by iteratively adjusting the vertex color in the neighbourhood. Hertz and Werra proposed the TabuCol (Hertz and Werra, 1987), which introduces the short-term memory mechanism of the tabu list. It records the last vertex color change in the tabu list and does not make the same color change within a certain tabu tenure, effectively preventing loops. In 2006, Ivo Blöchliger and Nicolas Zufferey (BlöChliger and Zufferey, 2008)proposed two improved strategies for partial schemes and a reactive tabu tenure, achieving good results on complex bench-

mark graphs. Other local search algorithms include simulated annealing algorithm(Chams et al., 1987), quantum annealing(Titiloye and Crispin, 2011b), improving probability learning based local search(Zhou et al., 2018), etc.

Population-based evolutionary algorithms are also an effective approach to solve GCP. In 1991, Davis (Mitchell, 1998) introduces the genetic algorithm in detail. In 1996, Fleurent (Fleurent and Ferland, 1996) et al. improved the genetic algorithm to solve GCP. In 2013, Marappan (Marappan and Sethumadhavan, 2013)et al. proposed uniparental conflict gene crossover and conflict gene mutation operators to make the genetic algorithm more effective. In 1999, Galinier and Hao (Galinier and Hao, 1999) introduced the hybrid evolutionary algorithm (HEA) by combining the tabu search algorithm with the evolutionary algorithm framework. In 2008, Galinier and Hertz (A et al., 2008) proposed an adaptive memory algorithm based on recombination operators to achieve the same results as the HEA. However, this algorithm is more flexible than the HEA. In 2010, Lü and Hao (Lü and Hao, 2010)et al. proposed an adaptive multi-parent crossover operator and a pool updating strategies. In 2012, Titiloye (Titiloye and Crispin, 2011a)et al. combined an evolutionary algorithm and an improved simulated annealing algorithm to form a distributed hybrid quantum annealing algorithm, which led to further improvements in the results on the test dataset. For large graphs with high density and a large number of vertices, Wu and Hao (Wu and Hao, 2012) et al. first used a preprocessing method to extract larger independent sets. They then used a memetic algorithm to color the residual graphs, achieving improved results on the test dataset. In 2015, Moalic and Alexandre (Moalic and Gondran, 2015) proposed a hybrid evolutionary algorithm based on two individuals (HEAD), which reduced computation time. In 2017, Moalic and Alexandre(Moalic and Gondran, 2017)introduced random crossover and unbalanced crossover in HEAD to improve the performance of the algorithm. Goudet and Duval (Goudet et al., 2020)et al. proposed a population-based weight learning framework to solve the GCP. Prakash(Prakash et al., 2020) et al. proposed a tree-based maximum independent set extraction method in two steps to solve the GCP, obtaining some reasonable experimental results.

Reducing the computation time and increasing the success rate are the goals of GCP improvement. For example, hybrid evolutionary algorithms use the tabu search to improve the obtained schemes locally and then use crossover operators to provide good diversity globally. Inspired by HEAD, we propose a hybrid evolutionary algorithm NERS_HEAD for solving GCP. The main innovations include:

1) Propose strategy 1: a method for judging local optimal state in the evolutionary process that can introduce diversity at the right time;

2) Propose strategy 2: a new method for managing diversity to make elite individuals more diverse;

3) The strategies of 1 and 2 are combined to form NERS_HEAD to improve the efficiency of solving GCP.

The organization of this paper is as follows: Section 2 presents some necessary knowledge needed for the GCP and the basic framework of the hybrid evolutionary algorithm. Section 3 focuses on the new elite individual replacement strategy proposed in this paper. Experimental results are given in Section 4, and the necessary analysis of the experimental results is presented. Section 5 presents the conclusion and possible future improvements.

2 Related work

This section introduces some concepts and some basic algorithms that need to be used when solving GCP with hybrid evolutionary algorithms and the basic framework of the HEAD algorithm referenced in this paper.

2.1 Schemes and objective function

Given an undirected graph $G = (V, E)$, the graph coloring problem(GCP) can be described as divide the vertex set V into K subsets $V_1, V_2, \cdots, V_K$ and $V_i \cap V_j = \emptyset (i \neq j)$,$V = V_1 \cup V_2 \cup \cdots \cup V_K$, The vertices in each subset are assigned the same color, the vertices in different subsets are assigned different colors. Each such division is called a K-coloring scheme (also called scheme). If the K-coloring scheme $s = \{V_1, V_2, \cdots, V_K\}$ makes $\forall u, v \in V, e(u, v) \in E$, and u and v have different colors, then s is called a K-coloring solution (solution) of the GCP.

For a K-coloring scheme s, the objective function can be defined as for formula (1).

$$f(s) = \sum_{V_i \in S} \sum_{u,v \in V_i} \delta_{uv} \tag{1}$$

where

$$\delta_{uv} = \begin{cases} 1 \ \langle u, v \rangle \in E, \\ 0 \text{ otherwise } . \end{cases}$$

Obviously, $f(s) \geq 0$. In this way, only $f(s) = 0$, the scheme s is the solution of GCP. Therefore, GCP can be expressed as an optimization problem, as shown in formula (2):

$$s = \arg\min_{s \in S} f(s). \quad (2)$$

Where S is all possible schemes.

Solving GCP is finding (searching) K-coloring schemes to reduce the value of the objective function f to 0. The K-coloring scheme s_1 is better than s_2 means $f(s_1) < f(s_2)$.

2.2 The tabu search algorithm Tabucol

Tabu search algorithm has been widely used since it was proposed in 1987 (Hertz and Werra, 1987). Algorithm 1 gives a typical tabu search algorithm for solving the GCP(Galinier and Hao, 1999). Since the Tabu search does not depend on the quality of the initial scheme, an initial scheme is generally initialized at random: each vertex is assigned a color no greater than K to obtain a scheme with conflicting edges as the input to the algorithm. Resets the number of iterations and create the s^* to save the *gbest* scheme(lines 1-2). Line 3 is the termination condition. The size of the MaxIter is determined by the experiment. In line 5, the color of the vertices is changed to minimize the objective function value(using a one-step move strategy, see Section 2.3). In line 6, the tabu list is introduced to avoid making the same vertex color transformation within a certain number of iterations, and it is a two-dimensional list. One dimension is the vertices, and the other dimension is the colors. When a color transformation is performed, it is recorded in the tabu list, and the same operation cannot be performed again within a certain tabu tenure tl. Line 9 updates the *gbest* scheme s^*. In line 12, after reaching the number of iterations, the *gbest* scheme s^* is output.

In tabu search, the parameter tl is called tabu tenure. Its size directly affects the search effect on the neighbourhood schemes, a longer tabu tenure can explore a large search space, but if it is set too long, it will not play the role of tabu, if the tabu tenure is short, the search will be limited to a smaller range. A better method was obtained in (Galinier and Hao, 1999) using formula (3) to generate the tabu tenure in a semi-random manner.

$$tl = random(A) + \partial * f(s) \quad (3)$$

where the range of the parameter A is [0,9], $\partial = 0.6$, and $f(s)$ represents the number of conflicting edges (objective function value) of the current scheme s. In this paper, we also adopt such a configuration. Meanwhile, static, dynamic and reactive tabu tenure change strategies are introduced in (BlöChliger and Zufferey, 2008).

Algorithm 1 :TabuCol; // The tabu search algorithm

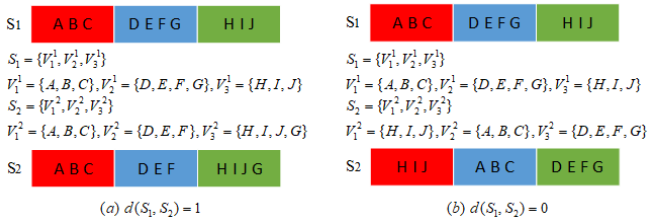
Input: A graph $G = \{V, E\}$, scheme s_0;
Output: the *gbest* scheme s^*;
1: $s \leftarrow s_0$; $Iter \leftarrow 0$;
2: Let s^* be the *gbest* scheme; /* s^* represents the scheme with the smallest objective function in the whole iterative process */
3: **while** $Iter! = MaxIter$ **do**
4: $Iter \leftarrow Iter + 1$;
5: Choose a best authorized move (v, i);
6: Introduce move $(v, s(v))$ in the Tabu list for tl iterations;
7: Perform the move $(v, s(v))$ in s;
8: **if** $f(s) < f(s^*)$ **then**
9: $s^* \leftarrow s$;
10: **end if**
11: **end while**
12: return s^*;

2.3 Distance between schemes

For any two schemes $s_1 = \{V_1^1, \ldots, V_K^1\}$ and $s_2 = \{V_1^2, \ldots, V_K^2\}$. Changing the color of a vertex, e.g. changing the color of vertex u in s_1 to j , is a move of vertex u from V_i^1 to V_j^1, called a one-step move. The distance between schemes s_1 and s_2 is defined as the number of moves in one-step needed to convert schemes s_1 to s_2 , denoted as $d(s_1, s_2)$, obviously $d(s_1, s_2) = d(s_2, s_1)$.

When the distance between two schemes is 1, they are said to be neighbours of each other. s_1 and s_2 in Fig.1(a) are neighbours of each other, and when the vertex $G \in V_3^2$ move to V_2^2, s_2 is the same as s_1, and the transformation between them requires only one-step move, $d(s_1, s_2) = 1$, The distance between schemes is only related to the division of vertices. In Fig.1(b), the colors of the vertices in s_1 and s_2 are different, but the vertex division is exactly the same. So, $d(s_1, s_2) = 0$.

The distance between the schemes can be solved using the bipartite graph maximum weight matching. The schemes s_1 and s_2 as two disjoint sets, color subsets in each scheme are considered as the vertices of the bipartite graph, and the number of matches between the color subsets of the s_1 and s_2 is the weight. The distance between s_1 and s_2 is equal to the number of vertices reduced by the maximum matching number. Fig.2 gives an example of converting two schemes to a bipartite graph and calculating the distance between schemes. Fig.2(b) is the bipartite graph abstracted from Fig.2(a). The number of vertices in Fig.2(a) is 10, and the bigraph shows that the maximum matching number is 6(3+2+1), then the distance = 4 between s_1 and s_2 in Fig.2.



(a) $d(S_1, S_2) = 1$ (b) $d(S_1, S_2) = 0$

Fig. 1: Example of the distance between schemes

(a) Two K-coloring (b) Bigraph

Fig. 2: Example of calculating for distance

2.4 Greedy Partition Crossover GPX

The Greedy Partition Crossover is a crossover operator. The two parents (schemes) $s_1 = \{V_1^1, \ldots, V_K^1\}$ and $s_2 = \{V_1^2, \ldots, V_K^2\}$, partition the vertices into K subsets according to color, such as V_i^1 in s_1, where 1 represents parent 1, i represents color i. All vertices in this subset are assigned color i. Algorithm 2 will get a new offspring combining the two parents. Alternately selecting the large color subset of the two parents in turn (lines 2-7). Then assign this subset to the offspring, and this subset in both parents is removed (lines 8-9), until all subsets in the parents have been selected. In line 11, the vertices that have not yet been assigned are randomly assigned to the subset of the offspring.

Algorithm 2 :GPX; // The GPX algorithm

Input: scheme $s_1 = \{V_1^1, \ldots, V_K^1\}$ and $s_2 = \{V_1^2, \ldots, V_K^2\}$;
Output: scheme $s = \{V_1, \ldots, V_K\}$;
1: **for** $l (l \leq l \leq K)$ **do**
2: **if** l is odd **then**
3: $A = 1$;
4: **else**
5: $A = 2$;
6: **end if**
7: Choose i such that V_i^A has a large color subset;
8: $V_l = V_l^A$;
9: remove the vertices of V_l^A from s_1 and s_2;
10: **end for**
11: Assign colors to the vertices $V - (V_1 \cup \cdots \cup V_K)$ randomly;
12: return s;

2.5 hybrid evolutionary algorithm in Duet HEAD

Hybrid evolutionary algorithms are a class of algorithms that combine local search algorithms with evolutionary algorithms and are often used to solve optimization problems. In 2017, Moalic et al. gave a hybrid evolutionary algorithm HEAD (Moalic and Gondran, 2017), as in Algorithm 3. HEAD removes the complex selection and update operators from the evolutionary algorithm and uses an elite strategy to manage population diversity.

Algorithm 3 randomly initializes two parents(P_1,P_2),

Algorithm 3 :HEAD; // HEA in Duet

Input: K is the number of colors, $Iter_{TC}$ is the number of Tabucol iterations, $Iter_{cycle} = 10$ is the number of generations into one cycle, Maxgeneration is the maximum number of generation;
Output: the *gbest* K-coloring scheme s; /* *gbest* represents the scheme with smallest objective function during the whole evolutionary process*/
1: $P = \{P_1, P_2, elite_1, elite_2, gbest\} \leftarrow init()$; /* Each vertex is randomly assigned a color no greater than K to obtain a scheme */
2: generation $\leftarrow 0$;
3: **while** $f(gbest) > 0$ and $P_1 \neq P_2$ and generation $<$ Maxgeneration **do**
4: $offspring_1 \leftarrow GPX(P_1, P_2)$;
5: $offspring_2 \leftarrow GPX(P_2, P_1)$;
6: $P_1 \leftarrow Tabucol(offspring_1, Iter_{TC})$;
7: $P_2 \leftarrow Tabucol(offspring_2, Iter_{TC})$;
8: $elite_1 \leftarrow saveBest(P_1, P_2, elite_1)$ /* Compare the objective function value in the three schemes and return the scheme with the smallest objective function value*/
9: $gbest \leftarrow saveBest(P_1, P_2, gbest)$;
10: **if** generation$\%Iter_{cycle} = 0$ **then**
11: $P_1 \leftarrow elite_2$; /* $elite_2$ is the individual with the smallest objective function value in previous cycle*/
12: $elite_2 \leftarrow elite_1$; /* $elite_1$ is the individual with the smallest objective function value in current cycle*/
13: $elite_1 \leftarrow init()$;
14: **end if**
15: generation++;
16: **end while**
17: return the *gbest* scheme s

elite individuals($elite_1$,$elite_2$) and the *gbest*(line 1). The GPX was used to generate two different offspring, and then the Tabucol was used to improve the two offspring to replace the two parents (lines 4-7). The *gbest* and the $elite_1$ are updated in lines 8 and 9. After each generation cycle ($Iter_{cycle}$=10), the $elite_2$ from the previous generation cycle replaces the P_1 (line 11). The final output is the *gbest* scheme.

Because the population uses only two individuals, there is no redundant selection, and each individual is involved in the evolutionary process. Experiments show that it can quickly find the solution of small and

medium-sized graphs. However, the introduction of the elite individual with a fixed generation cycle is not flexible. Moreover, there are fewer options for elite individuals, which sometimes do not effectively provide diversity. So this paper proposes a new elite individual replacement strategy to improve the HEAD algorithm.

3 hybrid evolutionary algorithm (NERS_HEAD)

This section discusses the hybrid evolutionary algorithm NERS_HEAD proposed in this paper to solve GCP and the replacement strategy of its elite individuals. Including:(1) The framework of NERS_HEAD (section 3.1); (2)The local optimal state detection method based on the change of objective function value(section 3.2); (3) the elite construction method and fitness function for selecting elite individual replacements (section 3.3).

3.1 NERS_HEAD framework

Inspired by HEAD, we give a hybrid evolutionary algorithm NERS_HEAD with a new elite replacement strategy, as Algorithm 4.

In Algorithm 4, six schemes are randomly generated, which are two parents (P_1,P_2), two elite individuals ($elite_1$,$elite_2$), *temp* is the individual with the smallest objective function in each generation period and the *gbest* is the individual with the smallest objective function during the whole Evolutionary process(line 1). The GPX was used to generate two different offspring, and then the Tabucol was used to improve the two offspring to replace the two parents (lines 4-7). Update *temp* and *gbest* (lines 8-9). During evolution, the change of the objective function value is used to determine whether it falls into a local optimal state (lines 10-13) (Section 3.2). If trapped in the local optimal state, a new scheme is obtained by MPX of elite individuals from the previous two periods(line 15). Then the scheme is added to the elite pool after the Tabucol (line 16) (Section 3.3). The fitness between elite and parents is calculated, and the smallest fitness is selected for elite replacement (lines 17-21) (Section 3.3). Finally, update the elite individual and the generations (lines 22-27). Output the *gbest*(line29).

Algorithm 4 :NERS_HEAD; // HEAD with new elite replacement strategy

Input: K is the number of colors, $Iter_{TC}$ is the number of Tabucol iterations, generation is the number of evolutive generations, Maxgeneration is the maximum number of evolutive generations, $\varphi = 5$, $\varepsilon = 0.1$;
Output: the *gbest* scheme s;

1: $P = \{P_1, P_2, elite_1, elite_2, temp, gbest\} \leftarrow init()$;
2: generation $\leftarrow 0$;$f_{pre} \leftarrow 999$;$f_{curr} \leftarrow 0$;
3: **while** $f(gbest) > 0$ and $P_1 \neq P_2$ and generation $<$ Maxgeneration **do**
4: $offspring_1 \leftarrow GPX(P_1, P_2)$;
5: $offspring_2 \leftarrow GPX(P_2, P_1)$;
6: $P_1 \leftarrow Tabucol(offspring_1, Iter_{TC})$;
7: $P_2 \leftarrow Tabucol(offspring_2, Iter_{TC})$;
8: $temp \leftarrow saveBest(P_1, P_2, temp)$;
9: $gbest \leftarrow saveBest(P_1, P_2, gbest)$;
10: **if** generation$\%\varphi = 0$ **then**
11: $f_{curr} = f(gbest)$;
12: $\Delta = f_{pre} - f_{curr}$; /*(section 3.2)*/
13: $f_{pre} = f_{curr}$;
14: **if** $\Delta = 0$ or $d(P_1, P_2) < \varepsilon * n$ **then** /*(section 3.3)*/
15: $elite_3 \leftarrow MPX(elite_1, elite_2)$;
16: $elite_3 \leftarrow Tabucol(elite_3, 0.1 * Iter_{TC})$;
17: for i=1,2,3, j=1,2 do
18: P(i,j) $\leftarrow$ fit(i,j) /*according to Eq.(4)*/
19: end for
20: (i,j)=agrmin{P(i,j)| i=1,2,3,j=1,2};
21: $P_j \leftarrow elite_i$;
22: $elite_2 \leftarrow elite_1$;
23: $elite_1 \leftarrow temp$;
24: $temp \leftarrow init()$;
25: **end if**
26: **end if**
27: generation ++;
28: **end while**
29: return the *gbest* scheme s;

3.2 Method of detecting local optimal state

Moalic and Alexandre proposed two versions of HEAD (Moalic and Gondran, 2017), the first version without the strategy of adding elite individuals, which allows the population to converge quickly but with a low success rate. The second version introduces elite individuals according to a fixed number of generations, which improves the success rate but reduces the efficiency. Inspired by these two versions, in NERS_HEAD, only P_1 and P_2 are used to participate in evolution so that the evolution process can converge faster. At the same time, after every φ generation of evolution, it is judged whether the evolution process falls into a local optimal state. If it is, it needs to jump out in time. This section discusses the judgment method of whether the evolution process is in a local optimal state, and the strategy of jumping out of the local optimal state is discussed in section 3.3.

In NERS_HEAD, the change of the objective function value and the distance between P_1 and P_2 are used to detect whether the evolutionary process falls into a local optimal state.

First, we believe that the objective function values after φ generations are equal means that the evolution-

ary process has fallen into a local optimal state. For scheme s, searching for a feasible scheme according to Eq.(1) is the process of searching for the objective function value $f(s)$ that decreases to 0. Every φ generations, the objective function value of the *gbest* is recorded in f_{curr}(Algorithm 4, line 11). The f_{pre} records the objective function value of the *gbest* in the last φ generations(Algorithm 4, line 13). Δ is equal to the difference between f_{pre} and f_{curr} (Algorithm 4, lines 12). If $\Delta = 0$ means the population fall into a local optimal state. The number of the generation period φ is determined experimentally (section 4.2).

Second, when the $d(P_1, P_2) < \varepsilon * n$ means that the evolutionary process is trapped in a local optimal state. n is the number of the vertices, ε is 0.1. This condition is to prevent the algorithm from stopping if the first condition is not satisfied since the algorithm ends when the distance between the parents is 0.

3.3 Strategy for jumping out of local optimal state

In the process of finding the *gbest* scheme, HEAD can achieve fast convergence in the early stage, and after falling into a local optimal state, it is challenging to jump out. If the structure of the scheme is completely broken to increase diversity will lead to worse results, and too little diversity cannot be jumped out in time. Hence, it is crucial to introduce proper diversity.

Since there are only two individuals, as much diversity as possible is obtained using the elite individuals in the pre-generation period. This paper proposes saving the elite individuals in the first two generation periods and combining two elites using multi-parental crossover (MPX) to be added to the elite pool. It can add more diversity while preserving the structure of most schemes.

This paper uses MPX for the case of 2 parents (m=2), and the pseudo-code is as Algorithm 5. The input of the algorithm is two schemes (parents), and the output is one scheme (offspring), which first selects the large color subset among the parents, assigns it to the offspring, and then removes all the vertices containing this subset from the parents (lines 2-4). After all the K color subsets are selected, the vertices in the offspring that are not assigned a color are randomly assigned a color (line 6). MPX(m=2) differs from GPX. Instead of alternating the selection of both parents, the large color subsets is selected in both parents each time. The color subsets of the same parent can be chosen consecutively and simultaneously. Since crossover does not improve the objective function value, but on the contrary, random coloring in the last step will also increase the objective function value, so this offspring has to be improved by Tabucol and then added to the elite pool (Algorithm 4, line 16). The Tabucol is more time-consuming, so we set the number of the iteration to be 0.1* $Iter_{TC}$. Because the purpose of this step is not to find the solution but to reduce the number of conflicts generated in the last step. In this way, the elite pool contains the elite individuals ($elite_1$ and $elite_2$) from the first two generation periods and the newly generated individuals($elite_3$) from MPX.

Meanwhile, this paper proposes a new approach to

Algorithm 5 :MPX(m=2) // The MPX (m=2) algorithm

Input: solution $s_1 = \{V_1^1, \ldots, V_K^1\}$ and $s_2 = \{V_1^2, \ldots, V_K^2\}$;
Output: solution $s = \{V_1, \ldots, V_K\}$;
1: **for** $l(l \le l \le K)$ **do**
2: Choose i such that $V_i^A (A = 1\, or\, 2)$ has a large color subset;
3: $V_l = V_l^A$;
4: remove the vertices of V_l^A from s_1 and s_2;
5: **end for**
6: Assign colors to the vertices $V - (V_1 \cup \cdots \cup V_K)$ randomly;
7: return S;

choose the elite for replacement, which calculates the fitness between the elite individuals($elite_1$,$elite_2$,$elite_3$) and the parents (P_1,P_2) based on the objective function and distance, as in formula (4).

$$fit(i,j) = f_{ij}^{\frac{1}{2}} + exp^{\frac{d_{ij}}{\beta * n}} \tag{4}$$

where $fit(i,j)$ represents the fitness value of i and j, i represents elite individuals, j represents parents. f_{ij} represents the difference between the objective functions vaule of i and j, d_{ij} represents the distance between i and j, β is 0.15, and n is the number of vertices. The proposed fitness calculation method combines distance and objective function.

According to the experiment (Section 4.5), the difference of the objective function can reflect the degree of similarity between two individuals to some extent. If the difference is larger, the less similar the two individuals are. But d_{ij} is more responsive to the similarity of two individuals than f_{ij} because when the distance of two individuals is 0, these two individuals must be the same. Still, if f_{ij} is 0, it cannot be judged that two individuals must be the same. So the value of f_{ij} is weakened by opening the root sign on the fitness value. Adjusting the ratio of β can influence the degree of distance on the fitness value. The smaller the value of β, the greater the influence of distance on the fitness value, and the larger the value of β, the smaller the influence on the fitness value. The idea of the formula is to find out the smallest fitness between elite individuals and parents, that is, the more similar individuals. Two

variables are used so that similar individuals can still be selected when one variable is the same.

The purpose of increasing diversity is to prevent gradual homogenization between individuals. Replacing j with i corresponding to the minimum fitness value $fit(i, j)$ prevents the two individuals after replacement from being too similar.

4 Experimental and analysis

This section will introduce the experiment datasets and the parameter settings of the experiments, and the detailed results and analysis. Experiments on the effectiveness of using strategy 2 are given in Section 4.3. A comparison with the result of the excellent current algorithms will also be made to verify the effectiveness of the algorithm improvements(section 4.5).

4.1 Dataset and experimental environment

The datasets used in our experiments comes from the DIMACS challenge benchmark dataset, most of which are random or quasi-random. There are a total of 34 datasets, which can be divided roughly into four categories by name. These datasets are widely used in the research of graph coloring algorithms.

The dataset details are shown in Table 1, where the chromatic number is marked as $\chi(G)$, '?' means that the chromatic number has not been found yet. The Edge destiny is 2m/n(n-1), where m and n are the number of edges and vertices of the graph, respectively.

NERS_HEAD algorithm is coded in c++. The experimental environment is windows 10, and the processor is Intel Xeon Platinum 8369HC 3.3GHz processer-4 cores and 8GB of RAM. Since HEAD is open source, we run the HEAD algorithm and NERS_HEAD under the same experimental conditions to compare the experimental results.

Table 1: The details of datasets

Instances	Vertex	Edge	Edge destiny	$\chi(G)$
dsjc125.1	125	736	0.1	5
dsjc125.5	125	3891	0.5	17
dsjc125.9	125	6961	0.9	44
dsjc250.1	250	3218	0.1	?
dsjc250.5	250	15668	0.5	?
dsjc250.9	250	27897	0.9	?
dsjc500.1	500	12458	0.1	?
dsjc500.5	500	62624	0.5	?
dsjc500.9	500	112437	0.9	?
dsjc1000.1	1000	49629	0.1	?
dsjc1000.5	1000	249826	0.5	?
dsjc1000.9	1000	449449	0.9	?
le450_5a	450	5714	0.06	5
le450_5b	450	5734	0.06	5
le450_5c	450	9803	0.1	5
le450_5d	450	9757	0.1	5
le450_15a	450	8168	0.08	15
le450_15b	450	8169	0.08	15
le450_15c	450	16680	0.17	15
le450_15d	450	16750	0.17	15
le450_25a	450	8260	0.08	25
le450_25b	450	8263	0.08	25
le450_25c	450	17343	0.17	25
le450_25d	450	17425	0.17	25
flat300_28_0	300	21695	0.48	28
flat1000_50_0	1000	245000	0.49	50
flat1000_60_0	1000	245830	0.49	60
flat1000_76_0	1000	246708	0.49	76
dsjr500.1c	500	121275	0.97	?
r250.5	250	14849	0.48	65
r1000.1c	1000	485090	0.97	?
r1000.5	1000	238267	0.48	?
c2000.5	2000	999836	0.5	?
c4000.5	4000	4000268	0.5	?

4.2 Local optimal state detection period φ

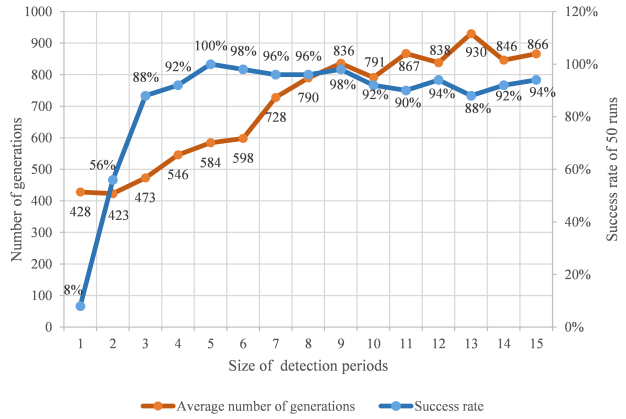
During the evolution of NERS_HEAD, every φ generation will detect whether the algorithm falls into a local optimal state. Two examples, dsjc1000.1 and dsjc500.5, are used to determine a more suitable value for the local optimal state detection period φ . The experiment results are shown in Fig.3, where the horizontal coordinates are the size of the period φ, ranging from 1 to 15. Each value runs the example 50 times, the left vertical coordinate indicates the number of generations, and the right vertical coordinate indicates the success rate.

As φ increases, the success rate tends to a stable value, but the average number of generations increases gradually and takes more time. Therefore, $\varphi = 5$ is chosen as the local optimal state detection period for the final experiment because the success rate is high and the number of generations is low at this point.

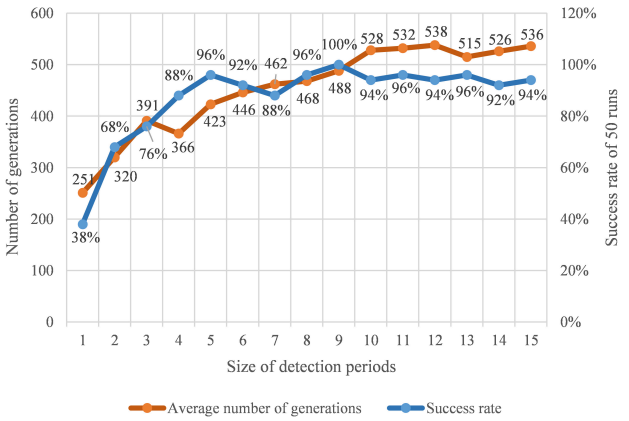
4.3 Elite individual replacement effectiveness experiment

Fig.4 gives a comparison of the average evolutionary generations solved using the elite replacement of strategy 2 and the elite replacement strategy of HEAD. NERS_HEAD can reduce the number of generations in most instances. Fig.4(a) gives instances for the generations less than 2000 and Fig.4(b) gives instances for the generations greater than 2000.

If better quality elites can be selected to provide diversity during the evolutionary process, it can reduce



(a) Success rate and average number of generations of dsjc1000.1



(b) Success rate and average number of generations of dsjc500.5

Fig. 3: The experiment of the local optimal state detection period φ

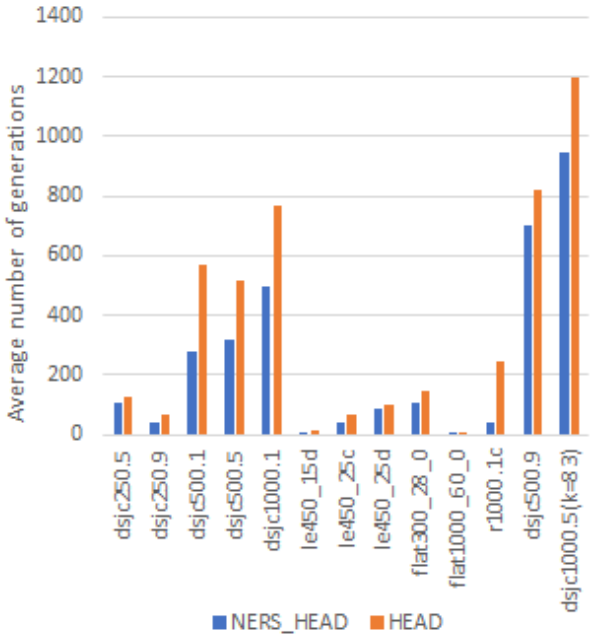
the number of generations. So, strategy 2 is effective.

Meanwhile, the dsjc1000.1 is used to explore the changes in the objective function value during the evolutionary process. HEAD and NERS_HEAD output the objective function value of the *gbest* scheme at every 50 generations. After running 20 times, the objective function values of the same evolutionary generation were averaged and plotted as Fig.5.

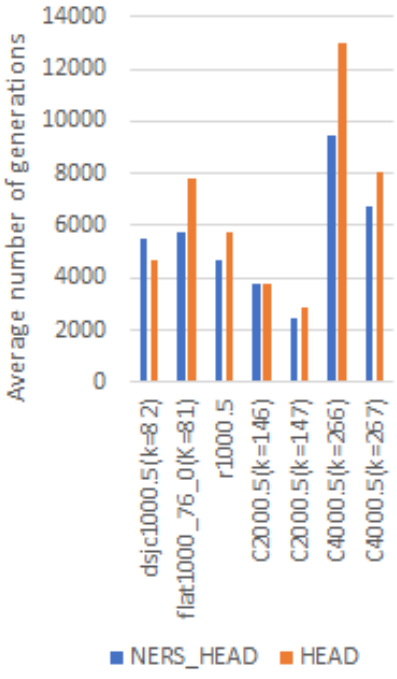
Overall, the value of the objective function decreases with the number of generations during the evolution. NERS_HEAD can find the solution faster.

4.4 Relationship between the difference of objective function and distance

In designing the fitness function, the objective function difference is needed to determine the degree of similarity between two individuals. There is no theoretical basis for whether individuals with larger or smaller objec-



(a) generation less than 2000



(b) generation more than 2000

Fig. 4: Comparison of the number of generations of the same instance

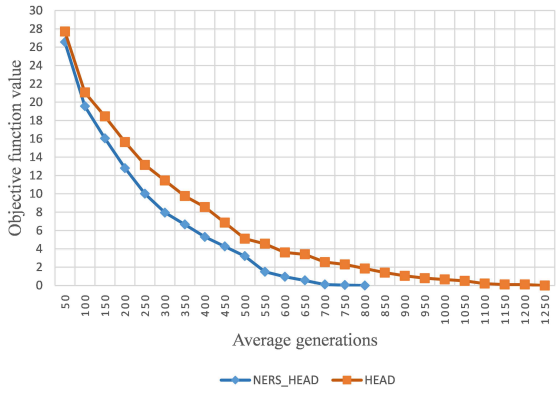


Fig. 5: The change of the objective function value on dsjc1000.1(k=20)

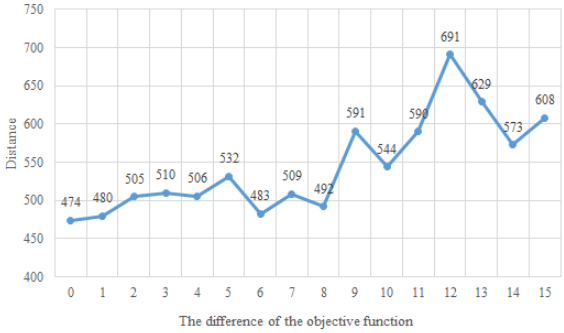


Fig. 6: Relationship between the difference of objective function and distance on dsjc1000.1(k=20)

tive function differences are more similar, so we determine this roughly by experiment. The dsjc1000.1 example, during each generation, outputs the distance and the difference of the objective function between two parents. The difference of the objective function for each scheme after the Tabucol is in a small range, and two schemes of the same objective function difference may correspond to multiple distances. Therefore, the horizontal coordinate is the objective function difference, and the vertical coordinate is the average distance corresponding to each objective function difference.

In Fig.6, the curve shows that as the difference in the objective function becomes larger, the distance between the two individuals increases. It shows that the difference of the objective function of the two schemes is inversely correlated with the degree of similarity.

4.5 Algorithm comparison

The comparison result of HEAD and NERS_HEAD is shown in Table 2. The form of each item is x(y), x and y are the experimental results of NERS_HEAD and HEAD(Moalic and Gondran, 2017), respectively. When x=y, only x is listed. The first column is the name of the instance. The second column is chromatic number. The third column K represents the number of colors. The fourth column is the number of Tabucol iterations. The fifth column Success: success_runs/total_runs. The right side is the number of total runs, usually set to 20 (10 for C2000.5 and C4000.5 for larger graphs), and the left side is the number of times the solution was found successfully. The Generation in the sixth column represents the number of crossovers or generations. Bold font indicates better values.

For the success rate, NERS_HEAD has two data that are worse than HEAD, three data is better than HEAD, and the other success rates are the same. Only on the basis that the success rate is guaranteed, it makes sense to compare the generations and computational time reductions. For 12 instances of the first category(dsjc), it can reduce the number of generations for seven instances and the computation time for six instances. The reduction in computation time is not particularly significant because of the additional time cost required to construct the elite pool. In the second category(le450), most of the examples are relatively simple and can be solved quickly. NERS_HEAD reduces the number of generations and computation time of 3 instances. For le450_15c and le450_15d, an alternative approach is used: adding a randomly generated scheme inside the elite pool and directly replacing one of the two parents with that approach gives a more significant success rate improvement (le450_15c improve from 3/20 to 20/20, and le450_15d improve from 1/20 to 20/20). In the remaining instances, the number of generations and the computation time can be reduced for most of the instances.

From Table 2, among 34 instances and 41 data items, NERS_HEAD can reduce the evolutionary generation of 21 data items and the computation time of 18 data items than HEAD. The average reduction in the number of evolutionary generations was calculated to be 28.3%. The average reduction in computation time is 22.11%. In particular, dsjc500.1 reduces the number of evolutionary generations by 51.40% and the computation time by 40%. r1000.1c reduces the number of evolutionary generations by 82.30% and the computation time by 73.45%. Although the number of generations is somewhat random in each calculation, the effectiveness of the strategies can be demonstrated if it is reduced in most instances. Therefore, NERS_HEAD is effective in reducing the number of evolutionary generations and computation time.

Table 2: Comparison of experimental results between NER_HEAD and HEAD

Instance	$\chi(G)$	k	$Iter_{TC}$	Success	Generation	Time(min)
dsjc125.1	5	5	400	20/20	7	< 0.01
dsjc125.5	17	17	5000	20/20	16	< 0.01
dsjc125.9	44	44	1000	20/20	4	< 0.01
dsjc250.1	?	8	1000	20/20	26	< 0.01
dsjc250.5	?	28	6000	20/20	**104**(124)	0.04
dsjc250.9	?	72	5000	20/20	**43**(68)	0.03
dsjc500.1	?	12	4000	20/20	**278**(572)	**0.06**(0.1)
dsjc500.5	?	48	8000	20/20	**318**(516)	**0.29**(0.38)
dsjc500.9	?	126	15000	10/20	**701**(818)	**1.46**(1.62)
dsjc1000.1	?	20	3000	20/20	**493**(768)	**0.16**(0.21)
dsjc1000.5	?	82	60000	3/20	5522**(4700)**	44.91**(41.5)**
		83	40000	20/20	**947**(1197)	**6.65**(7.26)
dsjc1000.9	?	222	50000	1/20**(2/20)**	**8942**(13262)	**67.39**(89.25)
		223	30000	13/20**(17/20)**	**2615**(3211)	**18.15**(20.7)
le450_5a	5	5	30000	20/20	1	< 0.01
le450_5b	5	5	60000	20/20	1	< 0.01
le450_5c	5	5	8000	20/20	1	< 0.01
le450_5d	5	5	8000	20/20	1	< 0.01
le450_15a	15	15	30000	20/20	3	< 0.01
le450_15b	15	15	30000	20/20	1	< 0.01
le450_15c	15	15	110000	3/20	6	0.02
		15	4000	**20/20**	**119**	**0.03**
le450_15d	15	15	100000	1/20	**2**(12)	**0.02**(0.05)
		15	5000	**20/20**	**756**	**0.33**
le450_25a	25	25	1000	20/20	1	< 0.01
le450_25b	25	25	1000	20/20	1	< 0.01
le450_25c	25	25	22000000	20/20	**40**(65)	**39.51**(51.76)
le450_25d	25	25	21000000	17/20	**86**(101)	**92**(96.31)
flat300_28_0	28	31	4000	20/20	**107**(148)	**0.03**(0.04)
flat1000_50_0	50	50	130000	20/20	5	0.33
flat1000_60_0	60	60	130000	20/20	**7**(9)	**0.43**(0.52)
flat1000_76_0	76	81	60000	**5/20**(3/20)	**5779**(7695)	**66.61**(85.76)
		82	60000	20/20	**1002**(1314)	**7.24**(8.63)
dsjr500.1c	?	85	42000000	1/20	1	0.2
r250.5	65	65	10000	1/20	**30565**(34898)	**12.4**(13)
r1000.1c	?	98	45000	2/20	**43**(243)	**0.47**(1.77)
r1000.5	?	245	360000	20/20	**4665**(5757)	**343**(377)
C2000.5	?	146	140000	5/10	**3795**(3799)	289
		147	140000	10/10	**2464**(2870)	**206**(220)
C4000.5	?	266	140000	4/10	**9432**(12949)	**1948**(2113)
		267	140000	8/10	**6723**(8034)	**1786**(1832)

5 Conclusion and future work

This paper proposes a hybrid evolutionary algorithm NERS_HEAD based on an elite replacement to solve GCP. NERS_HEAD guides the evolution direction of the population by setting the criteria of whether the evolutionary process is trapped in the Local optimal state and improves the global search ability by increasing the diversity of elites. The comparison experiment with the current excellent GCP solving algorithm on 34 DIMACS instances shows that NERS_HEAD can reduce the number of evolutionary generations and the computing time of most instances while ensuring the success rate. The average reduction in evolution generations and calculation time reached 28.3% and 22.11%, respectively. Therefore, NERS_HEAD is a more effective GCP solving algorithm.

Constructing elite individuals suitable for general instances has always been one of the main problems in solving GCP. MPX is used in this paper, but it has many crossover operators and mutation operators (Marappan and Sethumadhavan, 2020) when solving the GCP. For different types of graphs, changing the operator may get better results. In future work, we can study the characteristics of graphs and choose different operators to construct elite individuals.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Human and animal rights This article does not contain any studies with human participants or animals performed by any of the authors.

References

A PG, B AH, C NZ (2008) An adaptive memory algorithm for the k -coloring problem. Discrete Applied Mathematics 156(2):267–279, https://doi.org/10.1016/j.dam.2006.07.017

BlöChliger I, Zufferey N (2008) A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers & Operations Research 35(3):960–975, https://doi.org/10.1016/j.cor.2006.05.014

Briggs P (1995) Register allocation via graph coloring. Phd Thesis Rice University https://doi.org/10.1002/asna.19302381605

Bré D, laz (1979) New methods to color the vertices of a graph. Communications of the ACM https://doi.org/10.1145/359094.359101

Chams M, Hertz A, Werra DD (1987) Some experiments with simulated annealing for coloring graphs. European Journal of Operational Research 32(2):260–266, https://doi.org/10.1016/S0377-2217(87)80148-0

David O, Jose GG, Ivan MM, Enrique D (2018) Spectrum graph coloring and applications to wifi channel assignment. Symmetry 10(3):65–, https://doi.org/10.3390/sym10030065

Fleurent C, Ferland JA (1996) Genetic and hybrid algorithms for graph coloring. Annals of Operations Research 63(3):437–461, https://doi.org/10.1007/BF02125407

Galinier P, Hao JK (1999) Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4):379–397, https://doi.org/10.1023/A:1009823419804

Gamst A (1986) Some lower bounds for a class of frequency assignment problems. IEEE Transactions on Vehicular Technology 35(1):8–14, https://doi.org/10.1109/T-VT.1986.24063

Garey M, Johnson D, So H (1975) An application of graph coloring to printed circuit testing. Circuits & Systems IEEE Transactions on 23(10):591–599, https://doi.org/10.1109/TCS.1976.1084138

Goudet O, Duval B, Hao JK (2020) Population-based gradient descent weight learning for graph coloring problems. Knowledge-Based Systems 212:106581, https://doi.org/10.1016/j.knosys.2020.106581

Grech N (2018) Efficient reflection string analysis via graph coloring https://doi.org/10.4230/LIPIcs.ECOOP.2018.26

Hertz A, Werra D (1987) Using tabu search techniques for graph coloring. Computing https://doi.org/10.1007/BF02239976

Leighton FT (1979) A graph coloring algorithm for large scheduling problems. Journal of Research of the National Bureau of Standards (United States) 84(6), https://doi.org/10.1016/0305-0548(86)90061-4

Lü Z, Hao JK (2010) A memetic algorithm for graph coloring. European Journal of Operational Research 203(1):241–250, https://doi.org/10.1016/j.ejor.2009.07.016

Maitra T, Pal AJ, Bhattacharyya SDE, Kim TH (2010) Noise reduction in vlsi circuits using modified ga based graph coloring. International Journal of Control & Automation 3(2):37–44

Marappan R, Sethumadhavan G (2013) A new genetic algorithm for graph coloring. In: International Conference on Computational Intelligence, pp 49–54, https://doi.org/10.1109/CIMSim.2013.17

Marappan R, Sethumadhavan G (2020) Complexity analysis and stochastic convergence of some well-known evolutionary operators for solving graph coloring problem https://doi.org/10.3390/math8030303

Mitchell M (1998) L.d. davis, handbook of genetic algorithms. Artificial Intelligence 100(1):325–330, https://doi.org/10.1016/S0004-3702(98)00016-2

Moalic L, Gondran A (2015) The new memetic algorithm head for graph coloring: an easy way for managing diversity. Springer International Publishing https://doi.org/10.1007/978-3-319-16468-7_15

Moalic L, Gondran A (2017) Variations on memetic algorithms for graph coloring problems. Journal of Heuristics https://doi.org/10.1007/s10732-017-9354-9

Prakash, C, Sharma, Narendra, S, Chaudhari (2020) A tree based novel approach for graph coloring problem using maximal independent set. Wireless Personal Communications 110(3):1143–1155, https://doi.org/10.1007/s11277-019-06778-0

Titiloye O, Crispin A (2011a) Graph coloring with a distributed hybrid quantum annealing algorithm. In: Kes International Conference on Agent & Multi-agent Systems: Technologies & Applications, https://doi.org/10.1007/978-3-642-22000-5_57

Titiloye O, Crispin A (2011b) Quantum annealing of the graph coloring problem. Discrete Optimization 8(2):376–384, https://doi.org/10.1016/j.disopt.2010.12.001

Woo TK, Su S, Newman-Wolfe R (2002) Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm. IEEE Transactions on Communications 39(12):1794–1801, https://doi.org/10.1109/26.120165

Wu Q, Hao JK (2012) Coloring large graphs based on independent set extraction. Computers & Operations Research 39(2):283–290, https://doi.org/10.1016/j.cor.2011.04.002

Zhou Y, Duval B, Hao JK (2018) Improving probability learning based local search for graph coloring. Applied Soft Computing 65, https://doi.org/10.

1016/j.asoc.2018.01.027