A Discrete Firefly Algorithm Based on Similarity for Graph Coloring Problems

Kui Chen
Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba, Japan
Email: chen@kslab.cs.tsukuba.ac.jp

Hitoshi Kanoh
Division of Information Engineering, Faculty of Engineering Information and Systems, University of Tsukuba, Japan
Email: kanoh@cs.tsukuba.ac.jp

Abstract—In this paper, we propose a novel non-hybrid discrete firefly algorithm (DFA) for solving planar graph coloring problems. The original FA handles continuous optimization problems only. To apply it to discrete problems, we should redefine the original FA over discrete space. In this work, we introduce a new algorithm based on Similarity and discrete FA directly without any other hybrid algorithm. The experiments show that the proposed method outperforms the success rate of HDPSO and HDABC when solving planar graph coloring problems.

Index Terms—Swarm Intelligence, Graph Coloring Problem, Firefly Algorithm

I. INTRODUCTION

One of the most famous combinatorial optimization problems (COP) is Graph k-coloring problem (GCP). It states that given an undirected graph with n nodes and m edges, each node is assigned one of k colors so that two nodes connected with an edge have different colors. GCP is a kind of NP-complete problems. Because of simple definition, it can be used to model many scheduling and resource allocation problems, such as examination scheduling.

Swarm intelligence is the collective behavior of decentralized, self-organized systems. It is inspired from biological system such as the collective behavior of birds for foraging and defending. Swarm algorithms often consist of simple agents which not only interact locally with other individuals but also with the environment. This concept is introduced into computer science by Gerardo Beni and Jing Wang [1] and has been successfully applied to many COPs. For example, particle swarm optimization proposed by Kennedy and Eberhart [2], has been extended to solve discrete optimization problems [3] such as graph coloring problem [4], [5], [6], [7] and timetabling problem [8], [9], [10]. In addition, ant colony optimization obtains good results by solving Travelling-Salesman Problem (TSP) [11], [12]. Finally, the Artificial Bee Colony (ABC) Algorithm proposed by Karaboga and Basturk [13] exhibits excellent results when solving flow shop scheduling problem [14], [15] and graph coloring problem [16] and [17].

Firefly algorithm (FA) proposed by Xin-She Yang [18] is also a swarm algorithm. It is inspired by the flashing behavior of fireflies. Each firefly flashes its light and attracts other neighbor fireflies. The less brighter one will be attracted by the brighter one. However, the attractiveness between two fireflies depends on their distance. That is, the brightness decreases as two fireflies’ distance increases.

As PSO and ABC, original firefly algorithm is designed to solve continuous optimization problems and only a little research has been done to apply it to the area of combinatorial optimization problems. In this paper, we propose a discrete FA based on Similarity to solve graph 3-coloring problems. Although hybrid FA for GCP has been proposed [19], few general-used discrete ABC has been found. In the proposed method, the original FA has been divided into two parts and descritized directly without any hybrid algorithms. Our method is simple to implement and can be applied to other COPs readily.

In the following sections, we first introduce the original FA, GCP and related work. Next, we describe the proposed discrete firefly algorithm (DFA) in detail. Finally, we compare the results of experiments solving random generated GCP between DFA, HDPSO and DABC and show that our approach can obtain much higher success rate than the other two.

II. PROBLEM DESCRIPTION

A. Original Firefly Algorithm

In original FA, each firefly represents a solution vector \( x = [x_1, x_2, ..., x_n] \) in a given search space and the brightness of each firefly can be regarded as the firefly’s fitness. The search space is explored by moving firefly \( x_i \) towards brighter firefly \( x_j \) in accordance with below function:

\[
x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2} (x_j - x_i) + \alpha (rand(0,1) - \frac{1}{2})
\]

Note that there are two main parts in the update method, one is the attractiveness between \( x_i \) and \( x_j \) which is determined by \( \beta(r) = \beta_0 e^{-\gamma r_{ij}} \), the other is the random move determined by the random parameter \( \alpha \). Furthermore, the attractiveness depends on \( \beta_0 \), which is the attractiveness at \( r = 0 \), absorption coefficient \( \gamma \) and the Euclidian distance \( r_{ij} \) between two fireflies.

B. Graph Coloring Problems

Graph coloring problem is an NP-hard problem which is used to test the efficiency for many newly developed algorithms frequently. According to Four Color Theorem, the
solution can be found definitely when 4 colors are used. However, it is more difficult to find a solution when only 3 colors are allowed. So, we focus on graph 3-coloring problems in this paper. We generate random graphs with \( n \) nodes and \( m \) edges as follows:

1. Dividing nodes into three groups, each with \( \frac{n}{3} \) nodes;
2. Adding edges randomly between nodes in different groups;
3. Accept the graph until the total number of edges is \( m \).

This method is similar but not exactly identical to the method given in [20]. We also define constraint density \( d \) as

\[
d = \frac{m}{n}
\]

(2)

Constraint density indicates the level of difficulty for a given graph. Hogg pointed out that the most difficult problems arise when \( d \) is between 2.0 and 2.5 because the loose constraints allow for more local optimal solutions [21].

If 0, 1 and 2 represent red(R), green(G) and blue(B) respectively, a candidate solution of GCP can be defined as a \( 1 \times n \) vector \( x = [x_1, x_2, ..., x_i, ..., x_n] \), where \( x_i \in \{0, 1, 2\} \). Based on this scheme, one candidate solution of graph in Fig.1 can be expressed as \([0, 0, 2, 1]\):

\[
\text{Fig. 1. An example of graph 3-coloring problem}
\]

To evaluate a candidate solution \( x \), we define conflict for each solution:

\[
\text{conflict}_{jl} = \begin{cases} 
1 & \text{if } x_j = x_l \land j \in E(G) \\
0 & \text{otherwise}
\end{cases}
\]

(3)

\[
\text{conflict}(x) = \frac{1}{2} \sum_{j=1}^{n} \sum_{l=1}^{n} \text{conflict}_{jl}
\]

(4)

where \( E(G) \) indicates the set of edges. Eq.3 states that if two adjacent nodes share the same color, the conflict between the two nodes is 1, otherwise 0. The total conflict of solution \( x \) is given by Eq.4 obviously. Note that \( \frac{1}{2} \) is indispensable for avoiding calculating the conflict of two nodes twice.

The objective function used in this work is measured by:

\[
\text{fitness}(x) = 1 - \frac{\text{conflict}(x)}{m}
\]

(5)

where \( x \) is a candidate solution and \( m \) is the edge number of a given graph. The target of GCP is to find a solution whose total conflict is 0, that means the optimal solution is reached when \( \text{fitness}(x) = 1 \).

C. Related Work

Swarm algorithms have been used to solve many discrete optimization problems [3]. In this section, we introduce three discrete swarm algorithms which are proposed to solve graph coloring problems.

1) Memetic Firefly Algorithm: Memetic firefly algorithm (MFA) is a hybrid algorithm which combined firefly algorithm with a heuristical swap local search [19]. Using graph 3-coloring problems as test benchmarks, the performance of MFA was compared with hybrid evolutionary algorithm (HEA), Tabucol and evolutionary algorithm with SAW (EA-SAW). Experiments showed that MFA outperformed other three methods when coloring the uniform and equi-partite graphs but behaved worse when solving flat graph coloring problems. On the other hand, because it is a hybrid method and designed for solving graph 3-coloring problems specially, MFA is difficult to be applied to other discrete optimization problems.

2) HDPSO: PSO with Transition Probability Based on Hamming Distance (HDPSO) is a non-hybrid discrete PSO proposed by Takuya Aoki [7]. In this method, Hamming distance is used to calculate the distance of two particles in the swarm. After that, to represent the internal relationship between colors in one particle, three relevant transition probabilities are calculated: \( P_{\text{rand}}, P_{\text{pbest}} \) and \( P_{\text{gbest}} \). Finally, each color in a particle is updated by these three probabilities. The basic update method of HDPSO is as follows:

\[
\text{procedure update-position}(x_i)
\]

for \( j = 1 \) to \( N \)

\[
\begin{align*}
\text{Generate random number } r \text{ in } [0,1]; & \\
\text{if } r <= P_{\text{rand}} & \\
\quad x_i[j] = \text{random color}; & \\
\text{else if } r <= P_{\text{rand}} + P_{\text{pbest}} & \\
\quad x_i[j] = \text{pbest}[i][j]; & \\
\text{else} & \\
\quad x_i[j] = \text{gbest}[j]; & \\
\end{align*}
\]

end if

end for

end procedure

Experiments show that HDPSO is an efficient method to solve graph 3-coloring problem and obtains better results than GA. However, its performance turns to be worse as the size of graph becomes large.

3) Discrete ABC: Discrete artificial bee colony (DABC) is also a non-hybrid efficient discrete swarm algorithm [17]. It uses Similarity as a criterion of telling distance of two bees. When moving a bee (or a candidate solution) \( x_i \), only such \( x_j \) that is similar to \( x_i \) can be selected as a neighbor and used to generate a new candidate solution \( v_i \). Once a neighbor solution \( x_j \) is determined, we randomly choose \( u \) components from \( x_j \) and replace the corresponding components of \( x_i \) by them. The procedure of updating is as follows:

\[
\text{procedure update-strategy(swarm)}
\]

for each solution \( x_i \) in swarm

\[
\begin{align*}
\text{vi} &= x_i; & \\
\text{randomly select a solution } x_j; & \\
\text{calculate the Similarity of } x_i \text{ and } x_j; & \\
\text{generate a random number } r \text{ in } [0,1]; & \\
\text{if } r < \text{Similarity} & \\
\end{align*}
\]
randomly choose u components from xj and replace the corresponding component of vi by them;  
if fitness(vi) > fitness(xi)  
replace xi by vi;  
else  
keep xi unchanged;  
end if  
end for  
end procedure

Compared with HDPSO, the experiments show that discrete ABC obtains higher success rate and lower average evaluation. Especially when applying them to large random graphs, the discrete ABC keeps relatively high success rate while the performance of HDPSO becomes worse.

In this paper, we compare our proposed discrete firefly algorithm with HDPSO and discrete ABC and test the performance of them.

III. PROPOSED METHOD

A. Basic Strategy

As section 2.1 shows, the update method of original firefly algorithm, which is given by equation (1), has two main parts: (a) $\beta_0 e^{-\gamma \beta_i}(x_j - x_i)$, which moves a firefly $x_i$ to another brighter firefly $x_j$, and (b) $\alpha(rand(0, 1) - \frac{1}{2})$, which moves $x_i$ randomly. We call the first part as $\beta$-step and the second part as $\alpha$-step. The basic strategy of proposed discrete firefly algorithm is simulating the two parts and discretizing them separately.

B. Discretize $\beta$-step with Similarity

In discrete $\beta$-step, we use Similarity defined in [7] to represent the difference between $x_i$ and $x_j$:

$$Similarity_{ij} = 1 - \frac{H(x_i, x_j)}{n}$$ (6)

where the $H(x_i, x_j)$ is the Hamming distance between $x_i$ and $x_j$ and $n$ is the dimension of solution vector, which is exactly identical to the node number of a given random graph.

Then, we calculate the attractiveness between two fireflies as:

$$\beta(x_i, x_j) = \beta_0 e^{-\gamma Similarity_{ij}^2}$$ (7)

After that, we generate a uniformly distributed random number $r$ from interval $[0, 1]$ for each component $k \in \{1, 2, ..., n\}$ and replace $x_{ik}$ with $x_{jk}$ when $r < \beta(x_i, x_j)$. The discrete $\beta$-step can be shown as below:

procedure discrete beta step (xi, xj)
  calculate the Similarity;
  calculate attractiveness beta;
  for k in [1, 2, ..., n]
    generate a random number r;
    if r <= beta
      replace xi[k] with xj[k];
  end for
end procedure

C. Discretize $\alpha$-step

Original $\alpha$-step adds a small random turbulence to some components of $x_i$. To simulate this procedure in discrete space, $\alpha$ different components of $x_i$ are randomly selected and replaced by other colors. The discrete $\alpha$-step can be shown as below:

procedure discrete alpha step (xi)
  select alpha different components of xi;
  replace them by different colors;
end procedure

D. Main Procedure of Discrete Firefly Algorithm

Now, we can build the main procedure of discrete firefly algorithm (DFA), which is quite similar with the original one. Let $I_i$ be the brightness of firefly $x_i$ and $I_i$ can be calculated by equation (5). In each generation, a firefly is moved towards another brighter one based on discrete $\beta$-step and $\alpha$-step. After all fireflies’ positions have been updated, we sort fireflies according to their brightness and compare the current best firefly’s brightness with its parent best firefly’s brightness. The parent best firefly will take the place of the current best firefly when the former is brighter than the latter. The procedure is as follows:

procedure of discrete firefly algorithm
  Initialization;
  while (not find optimal solution || max iteration is not reached)
    old_fireflies = new_fireflies;
    for (i from 1 to N)
      for (j from 1 to N)
        if (Ii <= Ij)
          beta-step;
          alpha-step;
        end if
      end for
    end for
    sort new_fireflies;
    if (current best < parent best)
      replace current best with parent best;
    end if
  end while
end procedure

where $N$ is the number of fireflies in swarm.

Note that $\beta$-step is essentially a monotonically decreasing function and $\alpha$-step seems like a random function. If firefly $x_i$ is similar to $x_j$, which means they are in the immediate vicinity, $x_i$ is hard to be affected by $\beta$-step but only affected by $\alpha$-step. On the other hand, if $x_i$ is far away from $x_j$, it has a good chance to be moved towards $x_j$ according to $\beta$-step.
In short, originally near fireflies are avoided becoming closer and closer, and a distant firefly gets a high probability to move to a better solution. By this strategy, discrete firefly algorithm is getting rid of being trapped in a sub-optimal solution.

IV. EXPERIMENTS

A. Parameter Dependence

Before comparing with HDPSO and DABC, we must examine the dependence of success rate on DFA’s three parameters: \(\alpha\), \(\beta_0\) and \(\gamma\). Success rate represents the ratio of the times of successfully finding the optimal solution from all tries. Using the method introduced in section 2.2, 30 random graphs with 90 nodes (that is \(n = 90\)) are generated. The constraint density \(d\) is 2.5, which is the most difficult problem. The swarm size \((N)\) is 200 and the max iteration is 10000.

Firstly, fixing \(\alpha\) on 3 and \(\beta_0\) on 1.0, we test DFA with different \(\gamma\) on the 30 random graphs. The result is shown by Fig. 2.

Fig. 2 shows that the best \(\gamma\) is 0.2 with which DFA successfully finding 18 times optimal solution from 30 tries. Accepting 0.2 as the optimal value of \(\gamma\) and fixing \(\alpha\) on 3, we test DFA with different \(\beta_0\) to find the optimal value of \(\beta_0\). The result is illustrated in Fig. 3.

We find that the best success rate arises when \(\beta_0 = 0.9\). Finally, fixing \(\gamma\) and \(\beta_0\) on their optimal values, we test DFA with different \(\alpha\) on the 30 random graphs. The result is shown in Fig. 4.

The optimal \(\alpha\) is 2 obviously. Using the optimal values of parameters of DFA, \(\alpha = 2\), \(\beta_0 = 0.9\), and \(\gamma = 0.2\), we compare the proposed method with HDPSO and DABC in next section.

B. Comparative Study

The proposed method is compared with HDPSO and DABC on different constraint density \(d\) to evaluate its performance. They are evaluated by two measures: success rate and average evaluation times of objective function. The latter reflects the convergence speed of an algorithm. The parameters used in HDPSO and DABC are taken from [7] and [17] respectively. For the sake of fairness, 100 independent tries are observed for each \(d\) because of the stochastic nature of swarm algorithms. The experiment conditions are given in Table I and the results are illustrated from Fig. 5 to Fig. 10.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>EXPERIMENT CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DFA</td>
</tr>
<tr>
<td>Swarm size</td>
<td>200</td>
</tr>
<tr>
<td>Max iteration</td>
<td>10000</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>2</td>
</tr>
<tr>
<td>(\beta_0)</td>
<td>0.9</td>
</tr>
<tr>
<td>(\gamma)</td>
<td>0.2</td>
</tr>
<tr>
<td>(w)</td>
<td>–</td>
</tr>
<tr>
<td>(c_1)</td>
<td>–</td>
</tr>
<tr>
<td>(c_2)</td>
<td>–</td>
</tr>
<tr>
<td>limit</td>
<td>–</td>
</tr>
<tr>
<td>(u)</td>
<td>–</td>
</tr>
</tbody>
</table>
As Figs. 5, 6, and 7 show, the proposed DFA obtains an excellent success rate. When applied to relatively small graphs, say $n = 90$, the success rate of DFA is higher than HDPSO and nearly equal well with DABC at $d = 2.5$, which is the most difficult problem. However, as the size of graph increases, DFA brings obvious advantages not only at $d = 2.5$, but also at $d = 2$ and $d = 3$. For example, the success rate of DFA is 78% when the graph size is 150 and $d = 2.5$. This is much higher than DABC’s 20% and HDPSO’s 1%.

The reason is that given a maximum iteration, DFA evaluates objective function more times than HDPSO and DABC do because of its double-loop design. In the comparative experiments, we set maximum iteration to 10000 because all three algorithms can evaluate objective function sufficiently when the graph size is 90. If a larger maximum iteration is used, the performance of HDPSO and DABC may improve when the graph size is 120 or 150.

On the other hand, as you can see from Figs. 8, 9, and 10, though our method performs much better than the other two algorithms on the aspect of success rate, its evaluation times of solving the most difficult problems are much larger.
than HDPSEO and DABC. That means DFA spends more time finding the optimal solution than the other two algorithms. This is because there are double loops when updating the position of fireflies which denotes the objective function will be calculated $N^2$ times at the worst case ($N$ is the swarm size). This is a weak point existing in both original FA and proposed DFA.

In summary, the proposed DFA outperforms the success rate of HDPSEO and DABC. Especially when the graph size is large, DFA obtains better success rate compared with other two methods. However, DFA’s convergency speed is much slower than HDPSEO and DABC when it is used to solve the most difficult problems at $d = 2, 2.5, \text{ and } 3$.

V. Conclusions

In this paper, we proposed a discrete firefly algorithm based on Similarity and used it to solve graph 3-coloring problems. Experiments on 100 randomly generated graphs show that the proposed method obtains good success rate even if the size of graph is very large. Furthermore, because our method discretizes original firefly algorithm directly without any other hybrid techniques, it can be applied to various large-scale real-life COPs. However, the weak point of the proposed method is also obvious. It spends more time than its rivals before finding the optimal solution. So, we do not claim that the proposed method is the best method to solve graph coloring problems.

In our future work, three things will be done. Firstly, the problem of slow convergency speed will be studied in depth. Second, Hamming distance and Similarity are used to measure the difference between fireflies in this work. We will try other methods of measuring distance and compare the performance between them. Finally, we will apply this method to other COPs and test its efficiency.

Acknowledgment

The authors wish to thank Dr. Claus Aranha of University of Tsukuba for his helpful comments and suggestions. This work is supported by JSPS KAKENHI Grant Number 15K00296.

References