Computer Modeling in Engineering & Sciences
DOI:10.32604/cmes.2020.09502
Article

Tech Science Press

A Novel Binary Firefly Algorithm for the Minimum Labeling Spanning Tree Problem

Mugang Lin[1,2,*], Fangju Liu[3], Huihuang Zhao[1,2] and Jianzhen Chen[1,2]

[1]College of Computer Science and Technology, Hengyang Normal University, Hengyang, 421002, China
[2]Hunan Provincial Key Laboratory of Intelligent Information Processing and Application, Hengyang, 421002, China
[3]School of Computer Science, University of South China, Hengyang, 421001, China
*Corresponding Author: Mugang Lin. Email: mglin@hynu.edu.cn
Received: 23 December 2019; Accepted: 17 July 2020

Abstract: Given a connected undirected graph G whose edges are labeled, the minimum labeling spanning tree (MLST) problem is to find a spanning tree of G with the smallest number of different labels. The MLST is an NP-hard combinatorial optimization problem, which is widely applied in communication networks, multimodal transportation networks, and data compression. Some approximation algorithms and heuristics algorithms have been proposed for the problem. Firefly algorithm is a new meta-heuristic algorithm. Because of its simplicity and easy implementation, it has been successfully applied in various fields. However, the basic firefly algorithm is not suitable for discrete problems. To this end, a novel discrete firefly algorithm for the MLST problem is proposed in this paper. A binary operation method to update firefly positions and a local feasible handling method are introduced, which correct unfeasible solutions, eliminate redundant labels, and make the algorithm more suitable for discrete problems. Computational results show that the algorithm has good performance. The algorithm can be extended to solve other discrete optimization problems.

Keywords: Minimum labeling spanning tree problem; binary firefly algorithm; meta-heuristics; discrete optimization

1 Introduction

Given a connected undirected labeled graph $G = (V, E, L)$, where V, E and L represent the set of vertices, the set of edges, and the set of labels, respectively. Each edge of E is assigned a label from set L and each label in L can be assigned to one or more edges. The minimum labeling spanning tree (MLST) problem is to find a spanning tree of G with the least number of different labels. Fig. 1 illustrates the MLST. In Fig. 1a, a label graph $G = (V, E, L)$ is given, where $V = \{v_1, v_2, \ldots, v_6\}$, $L = \{R, B, Gr, Y, P\}$, and the label with each edge is indicated by the letter close to it. Fig. 1b shows an optimal solution $L_{opt} = \{R, B\}$ with $|L_{opt}| = 2$, and its minimum labeling spanning tree. The MLST problem is a combinatorial optimization problem, which has a wide range of applications in communication networks [1], multimodal transportation networks [2], and data compression [3]. For example, in communication networks, there are many different types of communication media, such as fiber optics, cable, microwave, telephone line, and so on. A node can communicate with other nodes by selecting different types of media. When a communication network is constructed, it requires that the network must be connected. To reduce the cost and complexity of the network, it is often desirable to find a spanning tree that uses as few types of media as possible.

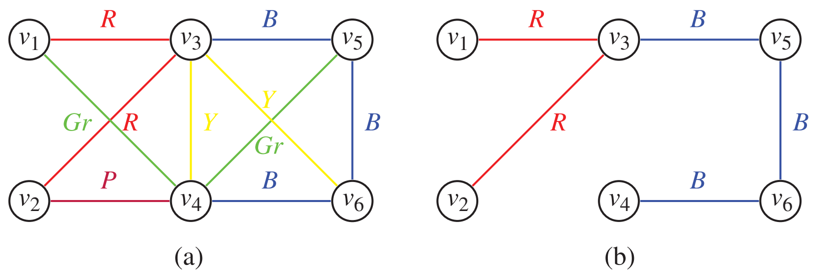


Figure 1: Small example for the MLST. (a) Original graph. (b) Minimum labeling spanning tree

The MLST problem was first introduced by Chang et al. [4]. They showed this problem is NP-hard by a reduction from the set cover problem and presented an exact exponential algorithm based on the A^* algorithm and two heuristics algorithms which are the edge replacement algorithm (ERA) and the maximum vertex cover algorithm (MVCA). Krumke et al. [5] analyzed the result of Chang and Leu from theoretical performance, proposed an approximation algorithm based on the MVCA with logarithmic performance guarantee and showed that there is no constant factor approximation with polynomial time for the MLST problem unless $P = NP$. Since then, researchers have studied the MLST problem and proposed many heuristic algorithms [6–10], approximation algorithms [11–13], integer programming (IP) models and methods [14–17]. In recent years, a wide variety of meta-heuristics methods have been proposed and used to solve NP-hard problems [18–20]. For the MLST problem, the meta-heuristics method is one of the main research methods. Xiong et al. [7,21] and Nummela et al. [22] proposed some efficient genetic algorithms for the MLST problem. Cerulli et al. [23] applied the pilot method to deal with the MLST problem and compared it with several meta-heuristic methods (simulated annealing, reactive tabu search, and variable neighborhood search). Chwatal et al. [24] applied ant colony optimization to solve the problem. Consoli et al. [25] proposed a hybrid local search method combining with variable neighborhood search and simulated annealing. Lai et al. [26] theoretically studied the performances of evolutionary algorithms for the MLST problem. Consoli et al. [27] proposed an intelligent optimization method that integrates variable neighborhood search with some complementary approaches and self-tuning mechanisms. Recently, Da Silva et al. [28] presented a meta-heuristic method, which combines the efficiency of constructing heuristic algorithms with the exploration capacity of local search methods based on MIP.

Nowadays, most meta-heuristic algorithms inspired by the biological process in nature have been developed and shown their power and efficiency in various fields, such as Genetic algorithm [7], Particle Swarm algorithm [29], and Ant Colony algorithm [24]. The swarm intelligent algorithms simulate the interaction and cooperation between individuals and their own independent behavior in real swarm organisms. Therefore, the methods have a good learning mechanism between individuals, which can promote the algorithm to converge faster, and a strong individual self-learning ability, which makes the algorithm to fall into a local optimum solution with lower probability. The firefly algorithm is a swarm intelligent meta-heuristic method developed by

Yang [30,31] to solve continuous optimization problems. It has two major advantages over other evolutionary algorithms: automatical subdivision and the ability of dealing with multimodality. Since the firefly algorithm is based on attraction and attractiveness decreases with distance the whole population can automatically subdivide into subgroups, and each group can swarm around each mode or local optimum. Among all these modes, the best global solution can be found. If the population size is sufficiently higher than the number of modes, this subdivision allows the fireflies to be able to find all optima simultaneously [32]. Although the basic firefly algorithm is easy to operate and implement, it cannot be directly used to solve discrete problems [33]. Thus, discrete firefly algorithms have received extensive attention from many researchers. There are two modifying methods to make the basic firefly algorithm suitable for discrete problems. The first method is to update the position of a firefly in the continuous space and then convert its result to discrete values based on a threshold function [34,35]. This method is suitable for problems with binary variables. However, the solution depends on the threshold function. The second method is to directly update the position of firefly in discrete space based on the nature code of problems [36,37]. The method uses Hamming distance to measure the distance between fireflies, and uses different update methods to update the position of fireflies according to the properties of problems.

In this paper, we study applying the firefly algorithm to solve the MLST problem. The main contributions are: (1) proposed a novel binary firefly algorithm for the MLST problem; (2) proposed a new strategy of updating firefly positions by switching the bits of the position of a firefly, where the changing bit number can be computed based on the attractiveness, distance or other parameters; (3) proposed a local feasible handling method to repair the unfeasible solution, eliminate redundant labels, and make the algorithm more suitable for discrete problems; (4) evaluated the performance of the algorithm, and the computational results show the algorithm is as effective as existing meta-heuristic approaches.

2 Firefly Optimization Algorithm

2.1 Firefly Algorithm

Firefly algorithm is a population-based meta-heuristic inspired by the flashing behavior of fireflies in nature [30,31]. In a swarm of fireflies, each firefly flashes its light, and its brightness can be different. Brighter fireflies will attract darker fireflies. Moreover, the attractiveness between two fireflies decreases with the increase of their distance and the light absorption of medium. In the algorithm, each firefly is a feasible solution randomly distributed in the solution space, the brightness of each firefly depends on the value of the objective function, and the attractiveness will guide fireflies to iteratively move towards more attractive locations to obtain a better solution.

Defining the brightness and attractiveness of fireflies are two important issues for the firefly algorithm to solve optimization problems. For a maximization problem, the brightness is proportional to the value of the objective function. For a minimization problem, however, the brightness is inversely proportional to the value of the objective function. The attractiveness between two fireflies is relative and changes with the distance between them. At the same time, the attractiveness is also related to the absorption coefficient of the medium. Its general form is as follows:

$$\beta\left(d_{ij}\right) = \beta_0 e^{-\gamma d_{ij}^k} \quad (k \geq 1) \tag{1}$$

where d_{ij} is the distance between two fireflies i and j, β_0 is the maximum attractiveness, i.e., the attractiveness at $d_{ij} = 0$, and γ is the light absorption coefficient. The distance d_{ij} between two fireflies i and j at positions $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ is defined the following:

$$d_{ij} = |\mathbf{x_i} - \mathbf{x_j}| = \sqrt{\sum_{k=1}^{dim} \left(x_{ik} - x_{jk}\right)^2} \tag{2}$$

where dim is the dimension number of firefly's position.

The updating position of a firefly is determined by the following formulation.

$$x_i = x_i + \beta\left(d_{ij}\right)\left(x_j - x_i\right) + \alpha\left(rand - 0.5\right) \tag{3}$$

In the left of Eq. (3), $\boldsymbol{x_i}$ is the new position of firefly i. In the right of Eq. (3), $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ are respectively the current positions of fireflies i and j; the second term is the increment of firefly i due to the attraction of the brighter firefly j, which makes the algorithm have global search capability; and the last term represents the random movement of firefly i, which lets the algorithm have local search capability; where α is the random parameter and $\boldsymbol{rand}$ is a vector of random number generator uniformly distributed in the space [0, 1].

2.2 *Binary Firefly Algorithm*

The basic firefly algorithm is proposed to solve continuous optimization problems, and the algorithm cannot be applied directly to discrete problems. To solve the permutation flow shop scheduling problems, Sayadi et al. [34] first proposed a binary firefly algorithm, which was designed by modifying the basic firefly algorithm to adapt to solving discrete problems. There are two methods to modify the original firefly algorithm to get the discrete algorithm. The first method is to update the position of fireflies in the continuous space and then discretize, and the second method directly updates the position of fireflies in discrete space. For a detailed review of binary firefly algorithms, see the recent survey of Tilahun et al. [38]. In this paper, the binary firefly algorithm belongs to the second method, and a novel method is proposed using a complete binary operation for updating the position of fireflies.

3 Binary Firefly Algorithm for the MLST Problem

In the MLST problem, a given labeled graph $G = (V, E, L)$ needs to be connected, where $|V| = n$, $|E| = m$ and $|L| = l$, otherwise there doesn't exist a solution to the problem. The MLST problem can be equivalently defined as a connected spanning subgraph instead of a spanning tree. Since if a spanning subgraph G' of graph G by induced all edges with labels in the label set R is connected, then any spanning tree of graph G' has at most $|R|$ labels. Moreover, if R is the minimum label set such that graph G' is connected, then any spanning tree of G' is a minimum labeling spanning tree in graph G'. Thus, for the MLST problem, a feasible solution can be defined as a subset R of label set L such that the graph G' induced by all edges with the labels in R is connected. In the following, we discuss some important definitions and operations of the binary firefly algorithm for the MLST problem in detail.

3.1 *Preprocessing*

To speed the algorithm, given instance is preprocessed first. In a connected graph, if a bridge (cut edge) is deleted, then the graph becomes unconnected. Thus, the corresponding label of the bridge must be in all feasible solutions of the MLST problem. Therefore, we first find all bridges in input graph G, and the corresponding label set is denoted by L_f. In the algorithm, we only require to find a label subset L_s from set $L - L_f$.

3.2 Initializing Population of Fireflies

The position of each firefly in a population represents a feasible solution, where the feasible solution is defined by a label subset L_s from set $L_u = L - L_f$ such that all edges with labels in $L_s \cup L_f$ construct a connected subgraph of G. For each firefly i, its position is encoded as a vector $\boldsymbol{x_i} = [x_{i1},\ x_{i2},\ \ldots,\ x_{il_u}]$, where $x_{ij} \in \{0,\ 1\}$, $l_u = |L_u|$ and $j \in \{1,\ 2,\ \ldots,\ l_u\}$, if $x_{ij} = 1$, then the j-th label in set L_u is in the feasible solution which the firefly stands for, otherwise it isn't in the solution. In initializing the population, the position $\boldsymbol{x_i}$ of each firefly i is generated by randomly assigning 1 to an element of $\boldsymbol{x_i}$ which is originally a zero vector until a feasible solution arises. Therefore, the initial population can be easily generated.

3.3 Defining Attractiveness and Distance

For the MLST problem, the light intensity of firefly i is defined as $I(\mathbf{x_i}) = l_u - \sum_{k=1}^{l_u} |x_{ik}$, which represents the number of unselected labels in set L_u. The less the number of selected labels is, the brighter the firefly flashes. The position of firefly is represented in binary code. Thus, Eq. (2) is no longer suitable for measuring the distance between two fireflies. In our algorithm, we use Hamming distance d_{ij} to measure the distance between two fireflies i and j at positions $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ respectively.

$$d_{ij} = |x_i - x_j| = \sum_{k=1}^{l_u} |x_{ik} \oplus x_{jk}| \tag{4}$$

where $\oplus$ denotes the XOR operation.

The attractiveness $\beta(d_{ij})$ between the two fireflies i and j is defined as follows.

$$\beta\left(d_{ij}\right) = \frac{\beta_0}{1 + \gamma d_{ij}} \tag{5}$$

3.4 Binary Movement Operator of Fireflies

In the algorithm, the distance between two fireflies is measured by the hetero-elements of two fireflies. Changing some hetero-elements in the darker firefly may improve its light intensity. Thus, the movement of a firefly is achieved by changing some elements of position x from 0 to 1 or from 1 to 0. In each iteration, each firefly has two ways of moving: the attraction movement guided by the brighter fireflies which is called β-step in the basic firefly algorithm and is regulated by the attractiveness, and the random movement. In the algorithm, we define the number l_{ij} of changing hetero-elements of firefly i due to the attraction of the brighter firefly j.

$$l_{ij} = round\left(\beta\left(d_{ij}\right) \times |\boldsymbol{x_i} - \boldsymbol{x_j}|\right) = round\left(\frac{d_{ij}\beta_0}{1 + \gamma d_{ij}}\right) \tag{6}$$

where $round(x)$ is the function that round x to the nearest integer. $\beta(d_{ij})$ is a percent of hetero-elements between fireflies i and j. Thus, the β-step operation is randomly selecting l_{ij} elements in the hetero-elements of firefly i with firefly j, and changing the values of the elements from 0 to 1 or from 1 to 0.

The random movement, also called α-step in the basic firefly algorithm, is controlled by parameter α. We define the number l_i of changing elements of firefly i caused by its random movement as follows.

$$l_i = round(\alpha \times |rand - 0.5| \times l_u) \tag{7}$$

where *rand* is a random number generator uniformly distributed in the space [0, 1]. The α-step operation is randomly selecting l_i elements in the position of firefly i, and changing the values of the elements from 0 to 1 or from 1 to 0.

After the β-step and α-step operations, the position of firefly i is updated.

3.5 Local Feasible Handling

In a connected labeled graph, a label is redundant if the remaining graph is still connected after deleting the corresponding edges of the label from the graph. After updating the position of each firefly, the graph represented by the position of a firefly can be unconnected, or has some redundant labels. Thus, we apply an additional local operation for each firefly such that we can build a feasible solution and eliminate redundant labels. The unfeasible solutions are repaired by randomly selecting an element whose value is 0 and changing its value from 0 to 1, until the graph which the firefly stands for becomes connected. Then, checking each label whether being redundant or not, and deleting all redundant labels.

3.6 Algorithm and Running Time Analysis

The binary firefly algorithm for the MLST is described in the pseudocode as Algorithm 1. In the following, we analyze the time complexity of Algorithm BFA. In the preprocessing (line 2), we call Tarjan's bridge-finding algorithm [39] to search all bridges of graph G with running time $O(n^2)$. β-step operation (line 10) and α-step operation can perform in time $O(l)$. In line 13, we can add or delete at most $O(l)$ labels and use depth-first search (DFS) with running time $O(m+n)$ to determine whether a solution is feasible or redundant. Thus, the worst running time of the local feasible handing (line 13) is $O(l(m+n))$. In line 6, it takes $O(Slg(S))$ time to sort the population in ascending order according to light intensity. Hence, it takes $O(TS^2l(m+n))$ time to run the programming from line 5 to line 17. In line 19, we can use the DFS method to get a spanning tree at $O(m+n)$ time. Therefore, the total running time of BFA is $O(TS^2l(m+n))$.

Algorithm 1: BFA

Input: A connected labeled undirected graph $G(V, E, L)$;

Output: A labeling spanning tree Tr and label set L_{Tr}.

```
1. Set parameters: $\alpha$, $\beta$, $\gamma$, $T$ (maximum iteration number) and $S$ (size of the population), $t = 1$;
2. Preprocess: find all bridges in graph $G$ and the corresponding label set $L_f$;
3. $L_u = L - L_f$;      % $L_f$ must be in the solution, while $L_u$ is uncertain.
4. Generate the initial population of fireflies $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_S}\}$;
5. while $t \leq T$ do
6.  Sort the population in ascending order by light intensity $I(\mathbf{x_i}) = l_u - \sum_{k=1}^{l_u} |x_{ik}|$;
7.  for $i = 1$ to $S$ do
8.   for $j = 1$ to $i$ do
9.    if $I(x_j) > I(x_i)$ then
```

(Continued)

Algorithm 1 (Continued)

```
10.       $\beta$-step operation: $l_{ij} = round(\frac{d_{ij}}{1+\gamma d_{ij}})$, randomly select $l_{ij}$ elements from the hetero-elements
          of firefly $i$ with firefly $j$, and change its values $0 \to 1$ or $1 \to 0$;
11.       $\alpha$-step operation: $l_i = round(\alpha \times |rand - 0.5| \times l_u)$, randomly select $l_i$ elements of firefly $i$,
          and change its values $0 \to 1$ or $1 \to 0$;
12.     end if
13.     Local Handling: if $G[\boldsymbol{x_i}]$ is unconnected, then randomly select an element $x_{ik} = 0$, let
        $x_{ik} = 1$, until $G[\boldsymbol{x_i}]$ is connected; Delete all redundant labels;
14.   end for
15.  end for
16.  $t = t + 1$;
17. end while
18. Get the brightest position $\boldsymbol{x_{best}}$;
19. Find a spanning tree $Tr$ of graph $G[\boldsymbol{x_{best}}]$ and label set $L_{Tr}$;
20. return the labeling spanning tree $Tr$ and label set $L_{Tr}$;
```

4 Experiments and Computational Results

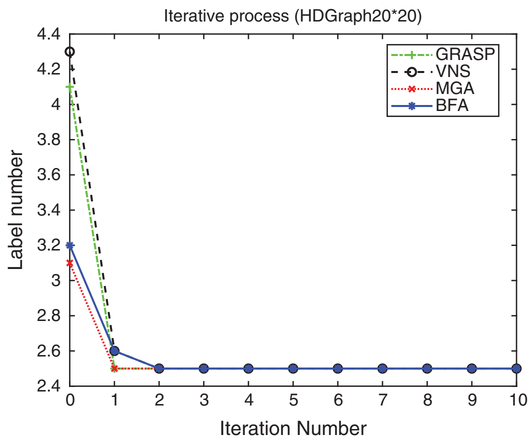
To evaluate the performance of the algorithms, we perform computational experiments on the set of benchmark instances of Cerulli et al. [23] which has been used in MLST literatures [7–10,21,24–28]. In our experiments, we choose 36 different datasets from the benchmark set which cover small, middle, and large instances, and each dataset contains 10 different graphs of the MLST problem with the same parameters which are the number of vertices (n), the density of edges (d), and the number of labels (l). In each graph, the number of edges $m = d \cdot \frac{n \cdot (n-1)}{2}$, where the value of d is 0.8, 0.5, and 0.2, respectively. For each dataset the solution quality is evaluated as the average objective function value of the 10 different graphs with the same parameters. Our method is compared in terms of solution quality, computational time, and iterative process with the methods: Exact method (EXACT), the MVCA [5], the VNS [8], the GRASP [8], and the MGA [7]. For the EXACT, we obtain the optimal solution by backtrack search which checks all possible subsets of the label set and finds all feasible solutions. To reduce the number of subsets, we use the label number of the MVCA solution in the initial step to prune some solution space. In the GRASP, we also use the above approach to reduce the number of possible subsets. The running time of the EXACT method grows exponentially, but it is reasonable if the problem size is small and the optimal solution is small. In our experiments, if it takes more than 3 hours to test the EXACT, the exact solution and running time are reported "-". The MVCA is a greedy heuristics that starts with an empty graph, then successively adds one label which minimize the number of connect components, until the graph becomes one connected graph. The VNS is a meta-heuristic method based on dynamic changes of the neighborhood structure during the search process. All of the methods were implemented on the Matlab platform and performed on a computer with Intel(R) Core(TM) I5-8265U CPU, 1.60 GHz, 8G of RAM, and Windows 10 as the operating system.

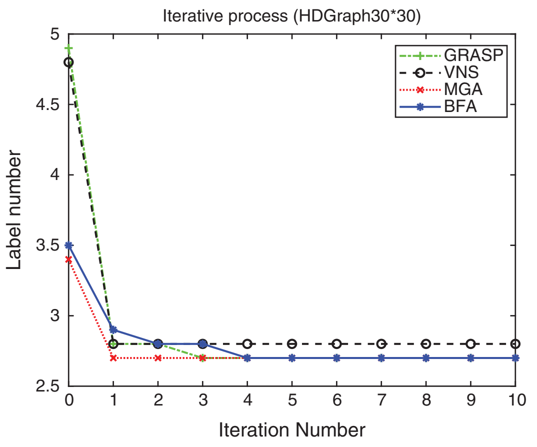
We first test small instances with $n = l = 20$, 30, 40, 50 and $d = 0.8$, 0.5, 0.2. The population size is 10 in the MGA and BFA, and the number of iterations is 10 except for the EXACT and

MVCA. Computational results are presented in Tab. 1, and iterative processes are described in Figs. 2–4. In Tab. 1, the values of the last row are the sum of the corresponding columns above. Observing the table and figures, the results and running time of the EXACT are better for small instances, but the running time increases exponentially as the size of instance becomes larger. All heuristic methods performed well for the instances. The MGA can converge to the best results after one iteration, but it is slower than other heuristic methods. Although the methods MVCA, VNS and GRASP have less running time than the MGA and BFA, their solutions are not very well. The BFA can yield the best solutions, and its running time is between the MGA and the MVCA, VNS and GRASP.

Table 1: Computational results for the MLST problem with $n = l = 20$, 30, 40, 50

Parameters			Objective function values						Time (seconds)					
d	*n*	*l*	EXACT	MVCA	VNS	GRASP	MGA	BFA	EXACT	MVCA	VNS	GRASP	MGA	BFA
0.8	20	20	2.5	2.6	2.5	2.5	2.5	2.5	0.141	0.029	0.325	0.221	4.519	1.953
	30	30	2.7	2.8	2.8	2.7	2.7	2.7	0.423	0.051	0.541	0.347	7.017	2.086
	40	40	2.6	2.7	2.6	2.6	2.6	2.6	0.669	0.065	0.807	0.492	8.561	2.490
	50	50	3	3	3	3	3	3	1.596	0.086	1.275	0.789	11.004	2.758
0.5	20	20	3.1	3.1	3.1	3.1	3.1	3.1	0.210	0.035	0.557	0.294	5.683	2.286
	30	30	3.9	4.0	3.9	3.9	3.9	3.9	2.652	0.063	1.099	0.581	8.801	2.769
	40	40	4	4	4	4	4	4	8.071	0.089	1.507	0.877	11.133	3.125
	50	50	4.1	4.5	4.1	4.2	4.1	4.1	29.039	0.125	2.537	1.281	14.231	3.702
0.2	20	20	7.2	7.4	7.2	7.2	7.2	7.2	16.431	0.078	2.084	0.812	22.482	4.615
	30	30	7.4	8.0	7.5	7.5	7.4	7.4	1026.321	0.150	3.653	1.245	31.877	7.001
	40	40	–	8.5	7.9	8.1	7.7	7.7	–	0.182	5.179	2.203	45.281	9.817
	50	50	–	9.6	9	9.1	8.9	8.9	–	0.325	8.748	2.838	62.374	10.116
Total			–	60.2	57.6	57.9	57.1	57.1	–	1.278	28.312	11.98	232.963	52.718





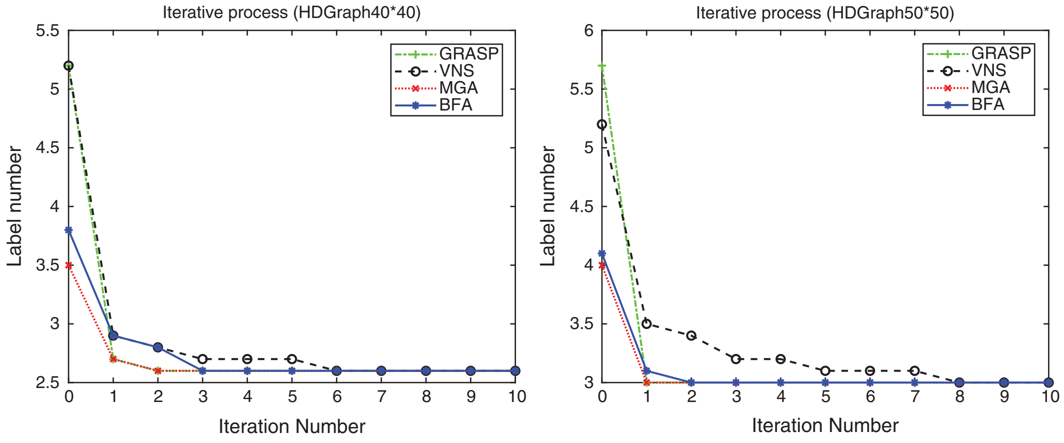


Figure 2: Iterative process of instances ($n = l = 20, 30, 40, 50$ and $d = 0.8$)

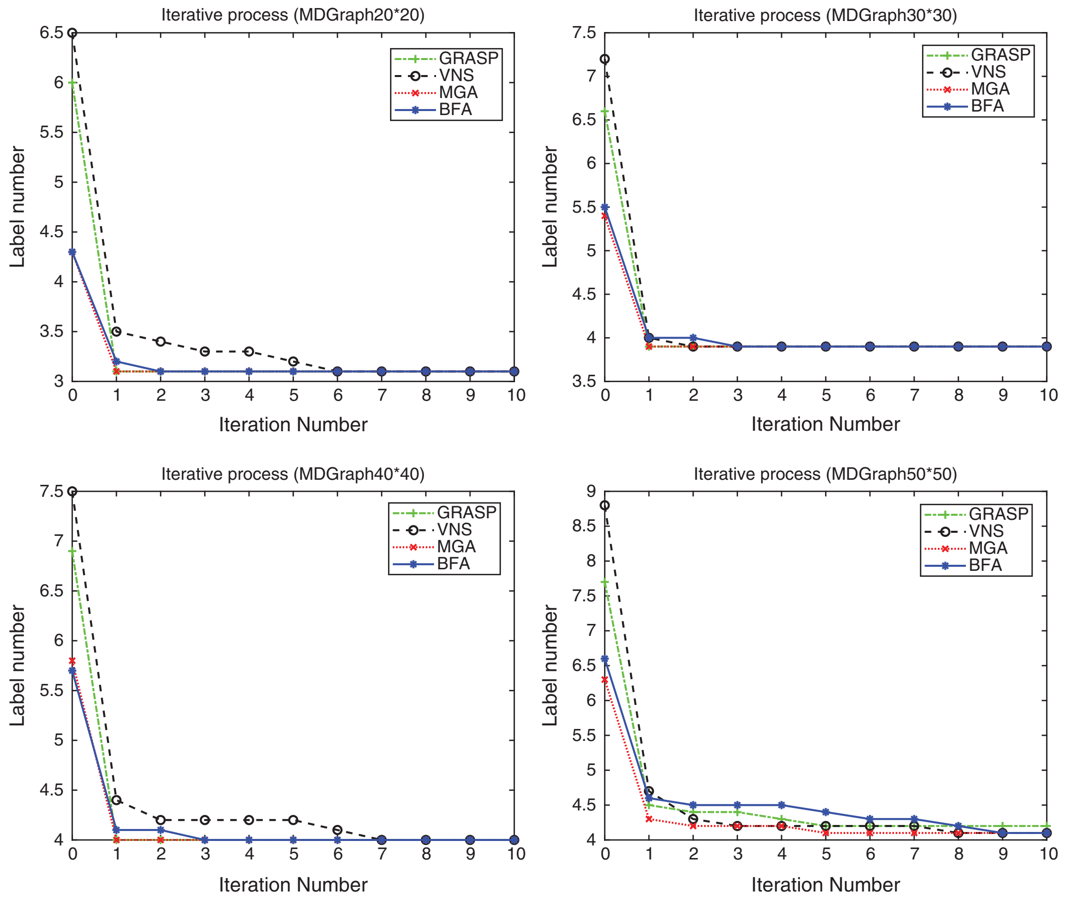


Figure 3: Iterative process of instances ($n = l = 20, 30, 40, 50$ and $d = 0.5$)

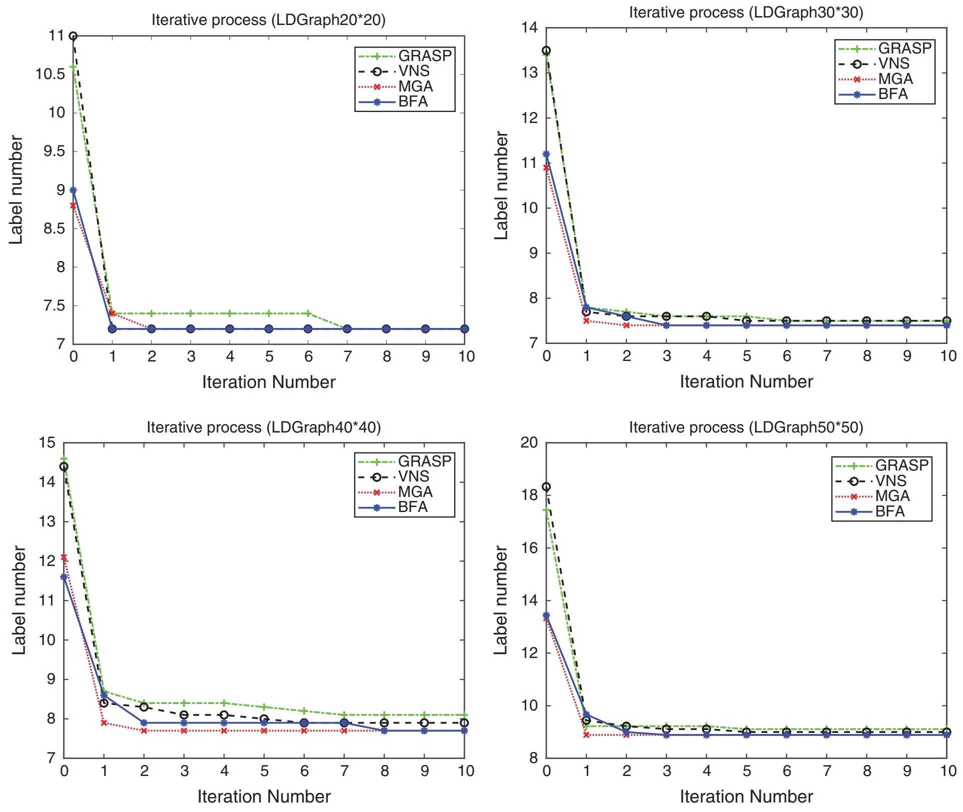


Figure 4: Iterative process of instances ($n = l = 20$, 30, 40, 50 and $d = 0.2$)

Table 2: Computational results for the MLST problem with $n = 200$

Parameters			Objective function values						Time (seconds)					
d	n	l	EXACT	MVCA	VNS	GRASP	MGA	BFA	EXACT	MVCA	VNS	GRASP	MGA	BFA
0.8	200	50	2	2	2	2	2	2	0.049	0.080	0.715	0.758	87.082	12.817
	200	100	2.4	2.6	2.6	2.4	2.5	2.5	2.129	0.217	3.098	2.913	153.992	15.649
	200	200	4	4	4	4	4	4	1235.045	0.713	14.126	9.222	347.396	23.552
	200	250	–	4.7	4.8	4.6	4.5	4.6	–	1.169	25.438	14.639	637.622	68.616
0.5	200	50	2.2	2.4	2.3	2.2	2.2	2.2	0.2745	0.117	1.371	1.477	191.258	29.885
	200	100	3.4	3.8	3.5	3.5	3.5	3.5	66.140	0.346	7.162	5.179	439.808	53.005
	200	200	–	6.1	6.1	5.8	5.9	5.9	–	1.031	36.519	17.921	804.553	63.250
	200	250	–	7	7.1	6.5	6.6	6.6	–	1.373	53.809	23.662	1215.473	75.947
0.2	200	50	5.2	5.5	5.4	5.3	5.3	5.3	981.397	0.219	6.844	3.344	150.340	33.121
	200	100	–	8.7	8.9	8.5	8.4	8.6	–	0.717	35.710	11.794	335.892	50.581
	200	200	–	13.4	13.3	12.8	12.4	12.4	–	2.289	182.592	34.037	837.144	148.797
	200	250	–	15.5	15.6	14.9	14.8	14.8	–	3.222	286.044	58.492	1041.374	236.309
Total			–	75.7	75.6	72.5	72.1	72.4	–	11.493	653.428	183.438	6241.934	811.529

The second dataset group is middle instances with $n = 200$, $l = 0.25n$, $0.5n$, n, $1.25n$, and $d =$ 0.8, 0.5, 0.2. The population size is 20 in the MGA and BFA, and the number of iterations is 15 except for the EXACT and MVCA. Computational results and iterative processes are described in Tab. 2 and Figs. 5–7. For instances with small density and a high number of labels, it is difficult for the EXACT to calculate the results in a limited time. All heuristic methods can perform for the instances, but their performances are different. The MGA can converge to the best results, but its speed is the slowest. Although the MVCA and VNS have fast running speeds, they are easy to fall into a local optimal solution. The GRASP and BFA can yield better solutions, but the BFA runs slower than the GRASP.

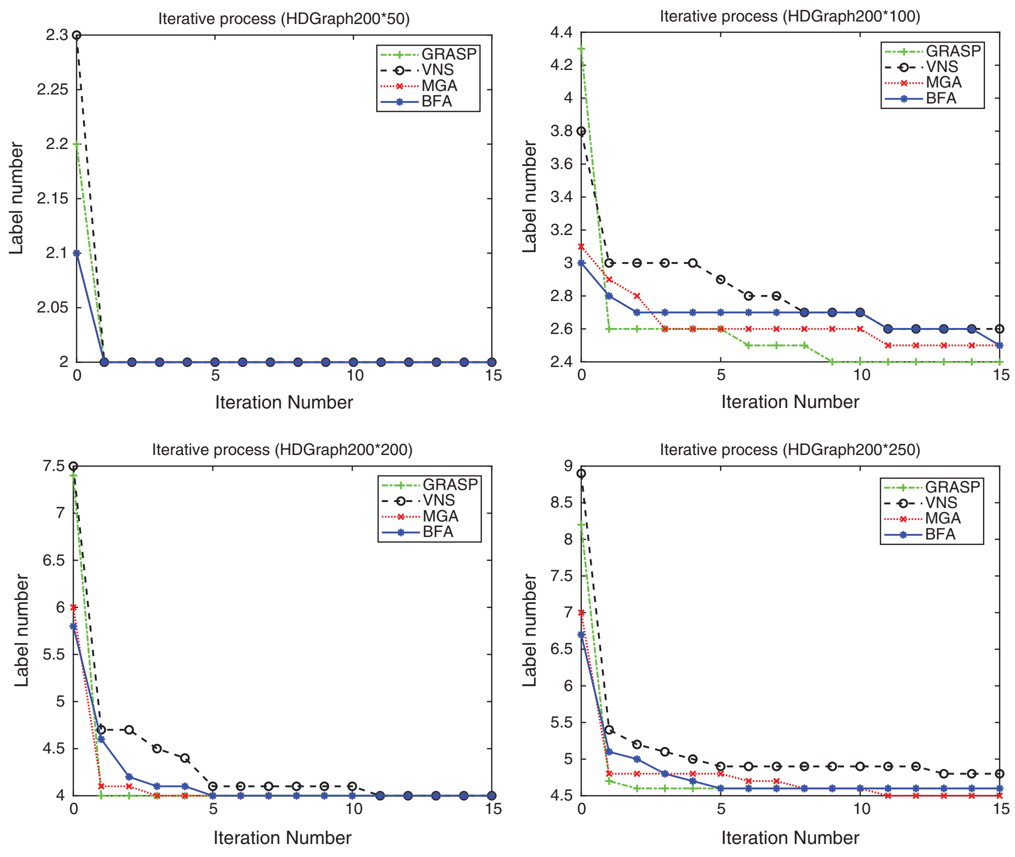
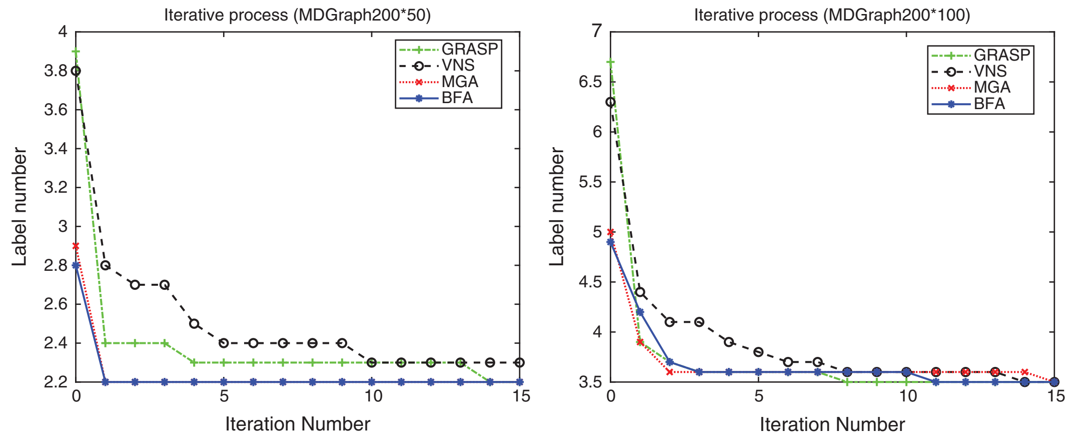


Figure 5: Iterative process of instances ($n = 200$, $l = 0.25n$, $0.5n$, n, $1.25n$, and $d = 0.8$)



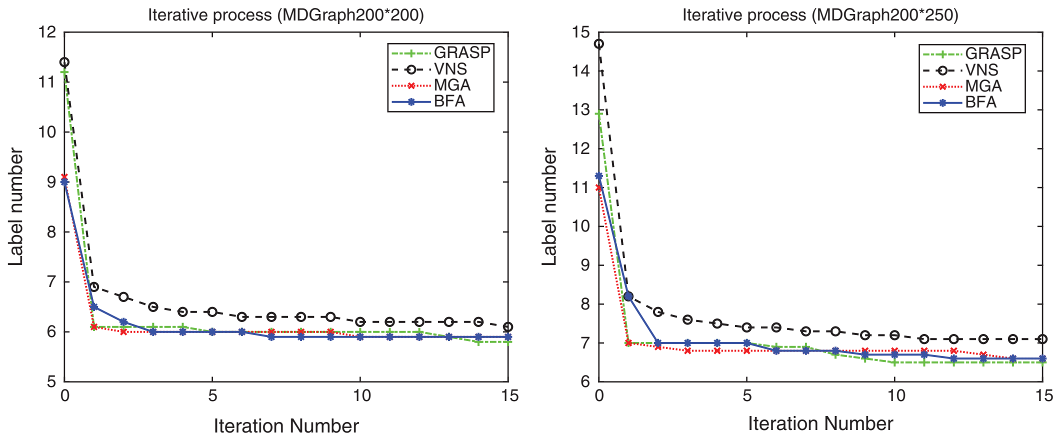


Figure 6: Iterative process of instances ($n = 200$, $l = 0.25n$, $0.5n$, n, $1.25n$, and $d = 0.5$)

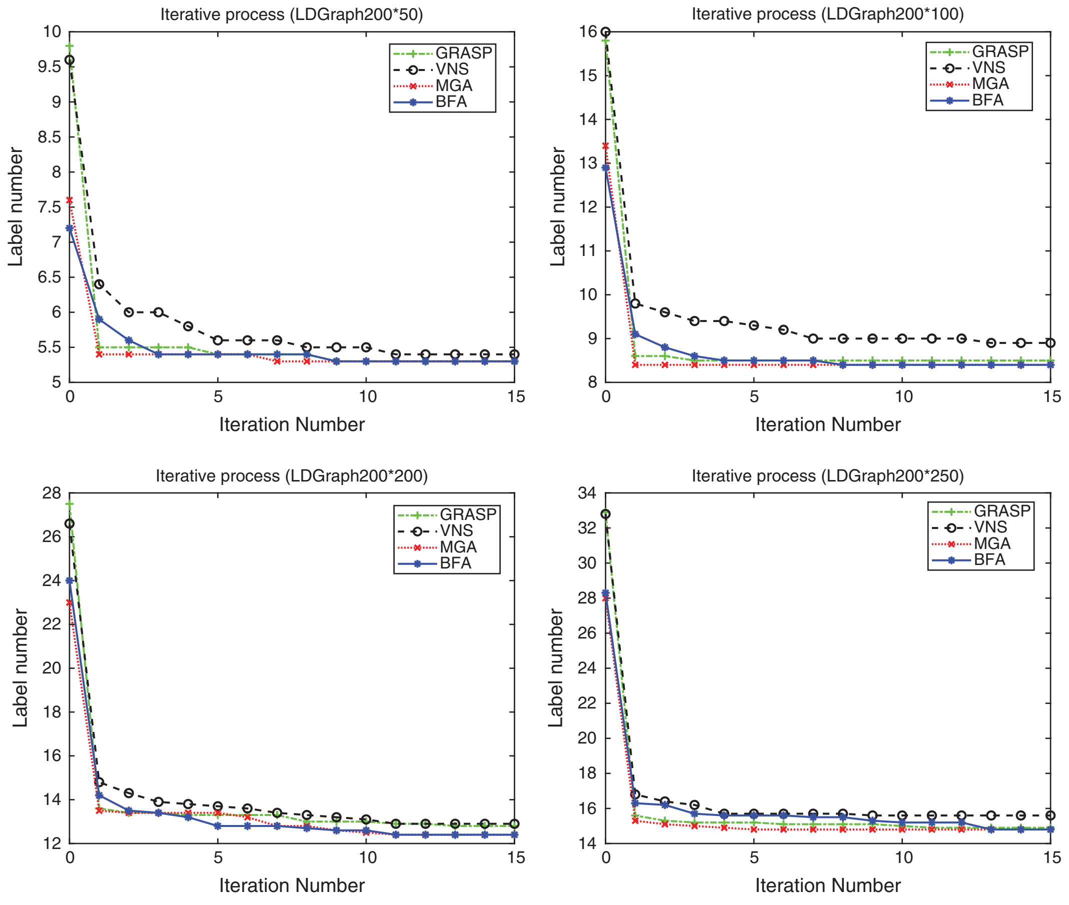
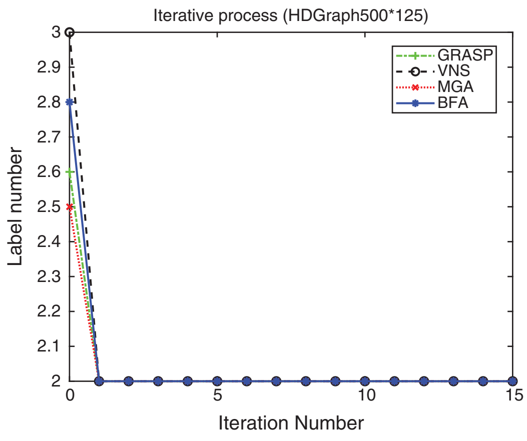


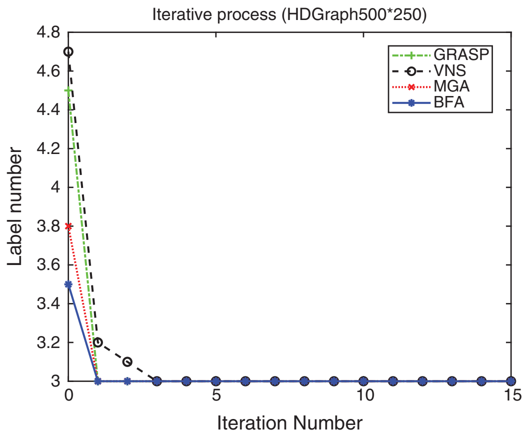
Figure 7: Iterative process of instances ($n = 200$, $l = 0.25n$, $0.5n$, n, $1.25n$, and $d = 0.2$)

The last dataset group is big instances with $n = 500$, $l = 0.25n$, $0.5n$, n, $1.25n$, and $d = 0.8$, 0.5, 0.2. The population size is 40 in the MGA and BFA, and the number of iteration is 15 except for the EXACT and MVCA. Computational results and iterative processes are described in Tab. 3 and Figs. 8–10. For most instances, the EXACT is difficult to calculate results within a limited time. All heuristic methods can perform for the instances, but they vary widely in performance. The MGA can obtain the best results, but its running time is unacceptable. Although the MVCA runs fast, its solution is bad. The performance of the VNS is worst in the methods. The comprehensive performances of the GRASP and BFA are the best in these methods, but the BFA runs slower than the GRASP.

Table 3: Computational results for the MLST problem with $n = 500$

Parameters			Objective function values						Time (seconds)					
d	n	l	EXACT	MVCA	VNS	GRASP	MGA	BFA	EXACT	MVCA	VNS	GRASP	MGA	BFA
0.8	500	125	2	2	2	2	2	2	3.266	0.659	7.458	7.124	657.503	97.505
	500	250	3	3	3	3	3	3	108.128	2.153	28.169	23.975	1266.91	123.841
	500	500	–	5	5	5	5	5	–	7.874	181.237	93.773	2723.955	276.55
	500	625	–	5.9	5.9	5.4	5.3	5.3	–	11.274	309.652	150.193	6090.295	255.61
0.5	500	125	2.9	3.1	2.9	2.9	2.9	2.9	24.193	1.023	15.399	13.125	691.59	118.979
	500	250	–	4.4	4.4	4.3	4.3	4.3	–	3.158	79.369	39.835	1328.951	162.847
	500	500	–	7.3	6.6	6.4	6.4	6.5	–	11.906	419.413	139.958	2718.718	338.509
	500	625	–	8.3	8.6	8.2	7.9	8.3	–	16.984	742.66	230.789	7539.984	432.228
0.2	500	125	–	6.8	6.3	6.3	6.2	6.3	–	2.376	89.763	33.865	1082.705	278.292
	500	250	–	10.6	10.3	10.1	10.1	10.2	–	7.525	475.213	133.82	2032.724	498.895
	500	500	–	16.9	16.5	16.3	16.2	16.2	–	24.188	1794.329	336.643	8637.177	925.178
	500	625	–	19.9	19.7	19.1	19.1	19.2	–	46.997	3546.501	619.628	11536.655	1177.899
Total			–	93.2	91.2	89	88.4	89.2	–	136.117	7689.163	1822.728	46307.17	4686.333





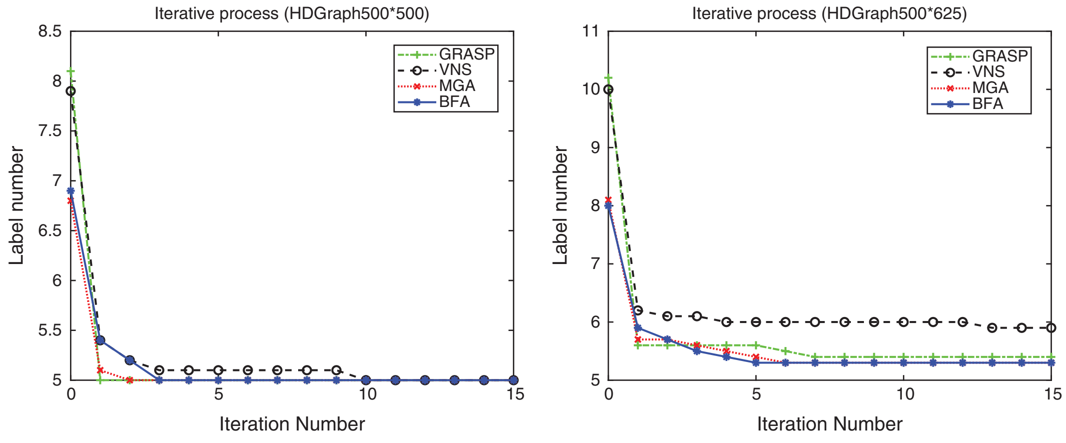


Figure 8: Iterative process of instances ($n = 500$, $l = 0.25n$, $0.5n$, n, $1.25n$, and $d = 0.8$)

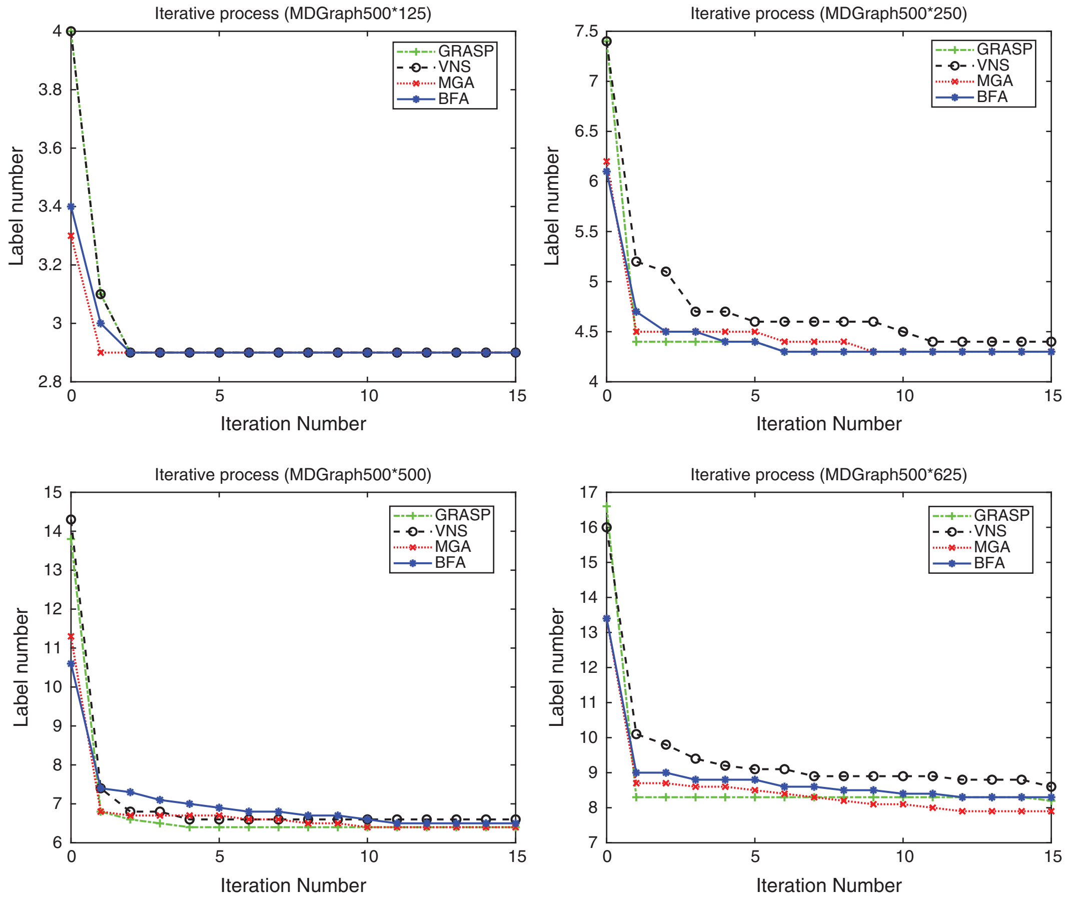


Figure 9: Iterative process of instances ($n = 500$, $l = 0.25n$, $0.5n$, n, $1.25n$, and $d = 0.5$)

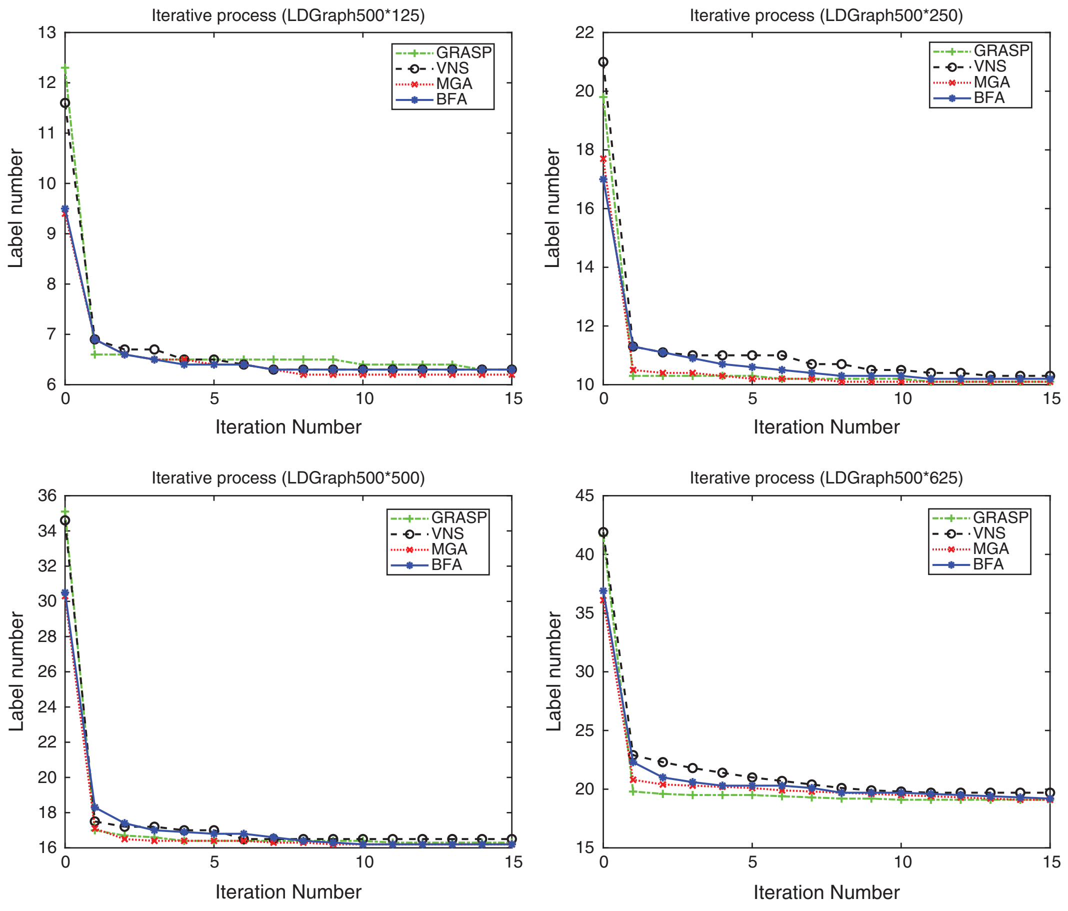


Figure 10: Iterative process of instances ($n = 500$, $l = 0.25n$, $0.5n$, n, $1.25n$, and $d = 0.2$)

By the above results, it is known that the complexity of the instances increases with the number of vertices and labels of the graph, and the reduction of the density in the graph. For the EXACT, the running time is reasonable if the size of instances is small or the optimal solution is small; while it grows exponentially with the problem size increasing and the graph density reducing. The MVCA has the shortest computational running time than other methods, however, it is easy to obtain a local optimal solution in the greedy local search process. The VNS is a meta-heuristic based on dynamically changing neighborhood structures during the local search process. When the density of the graph is different, the structural properties of the graph are also different. Thus, the computational time of the VNS mainly depends on graph density: the more sparse the graph, the larger the time. The MGA can obtain the solution with the best quality, but its running time is long, even unacceptable. This may be because its chromosomes are obtained by calling MVCA in each of its crossover operations, causing the convergence to become fast but the running time to increase. The GRASP is a meta-heuristic combining the power of greedy local search with randomization, such that it has a good diversification capability. The computational results show that the GRASP can obtain solutions with better quality in a shorter time. Although the BFA runs slower than the GRASP, the computational results reveal that the BFA can reliably search the global optimal or suboptimal solutions within a relatively reasonable time, and the BFA algorithm is stable and has good convergence ability. This may be due to the fact that the

BFA is a swarm intelligent algorithm based on multi-individual search, where each individual searches itself solution by interacting with each other during the iterative search process, the algorithm's final result takes the best solution of all individuals. Thus the quality of the solution is better, but the running time of the algorithm increases. Therefore, the BFA algorithm for the MLST problem is an effective method as other methods from the theoretical analysis and experimental results.

5 Conclusions

In this paper we proposed a binary firefly algorithm for the MLST problem. A novel method of updating positions of fireflies is introduced, which makes the algorithm more suitable for solving discrete problems. Moreover, we also applied a preprocessing step to improve the running time of the algorithm, and a local handling step to ensure a feasible solution after updating the position and eliminate redundant labels. Computational results show the algorithm for the MLST problem is as effective as the existing meta-heuristic algorithms. Our BFA algorithm can also be extended to solve other discrete optimization problems. Future research will include trying to improve the performance of the algorithm (such as hybridization with other methods), setting appropriate algorithm parameters and testing large instances of the MLST problem.

Funding Statement: This work is supported by the National Natural Science Foundation of China under Grant 61772179, the Hunan Provincial Natural Science Foundation of China under Grant 2019JJ40005, the Science and Technology Plan Project of Hunan Province under Grant 2016TP1020, the Double First-Class University Project of Hunan Province under Grant Xiangjiaotong [2018]469, the Open Fund Project of Hunan Provincial Key Laboratory of Intelligent Information Processing and Application for Hengyang Normal University under Grant IIPA19K02, and the Science Foundation of Hengyang Normal University under Grant 19QD13.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Tanenbaum, A. S. (1988). *Computer networks,* 2nd edition, Upper Saddle River, NJ, USA: Prentice-Hall, Inc.
2. Van Nes, R. (2002). *Design of multimodal transport networks: a hierarchical approach (Ph.D. Thesis).* TRAIL Research School, Delft University, The Netherlands.
3. Chwatal, A., Raidl, G., Dietzel, O. (2007). Compressing fingerprint templates by solving an extended minimum label spanning tree problem. *Proceedings of the Seventh Metaheuristics International Conference*, Montreal, Canada.
4. Chang, R. S., Shing-Jiuan, L. (1997). The minimum labeling spanning trees. *Information Processing Letters, 63(5),* 277–282. DOI 10.1016/S0020-0190(97)00127-0.
5. Krumke, S. O., Wirth, H. C. (1998). On the minimum label spanning tree problem. *Information Processing Letters, 66(2),* 81–85. DOI 10.1016/S0020-0190(98)00034-9.
6. Voss, S., Cerulli, R., Fink, A., Gentili, M. (2005). Applications of the pilot method to hard modifications of the minimum spanning tree problem. *Proceedings of the 18th MINI EURO Conference on VNS*, Tenerife, Spain.
7. Xiong, Y., Golden, B., Wasil, E. (2006). Improved heuristics for the minimum label spanning tree problem. *IEEE Transactions on Evolutionary Computation, 10(6),* 700–703. DOI 10.1109/TEVC.2006.877147.

8. Consoli, S., Darby-Dowman, K., Mladenović, N., Pérez, J. M. (2009). Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. *European Journal of Operational Research, 196(2),* 440–449. DOI 10.1016/j.ejor.2008.03.014.
9. Cerrone, C., Cerulli, R., Golden, B. (2017). Carousel greedy: a generalized greedy algorithm with applications in optimization. *Computers & Operations Research, 85,* 97–112. DOI 10.1016/j.cor.2017.03.016.
10. Cerrone, C., D'Ambrosio, C., Raiconi, A. (2019). Heuristics for the strong generalized minimum label spanning tree problem. *Networks, 74(2),* 148–160. DOI 10.1002/net.21882.
11. Wan, Y., Chen, G., Xu, Y. (2002). A note on the minimum label spanning tree. *Information Processing Letters, 84(2),* 99–101. DOI 10.1016/S0020-0190(02)00230-2.
12. BrüGgemann, T., Monnot, J., Woeginger, G. J. (2003). Local search for the minimum label spanning tree problem with bounded color classes. *Operations Research Letters, 31(3),* 195–201. DOI 10.1016/S0167-6377(02)00241-9.
13. Xiong, Y., Golden, B., Wasil, E. (2005). Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem. *Operations Research Letters, 33(1),* 77–80. DOI 10.1016/j.orl.2004.03.004.
14. Chen, Y., Cornick, N., Hall, A. O., Shajpal, R., Silberholz, J. et al. (2008). Comparison of heuristics for solving the GMLST problem. In *Telecommunications Modeling, Policy, and Technology,* pp. 191–217. Boston, MA: Springer.
15. Captivo, M. E., Clímaco, J. C., Pascoal, M. M. (2009). A mixed integer linear formulation for the minimum label spanning tree problem. *Computers & Operations Research, 36(11),* 3082–3085. DOI 10.1016/j.cor.2009.02.003.
16. Chwatal, A. M., Raidl, G. R. (2011). Solving the minimum label spanning tree problem by mathematical programming techniques. *Advances in Operations Research, 2011(2),* 1–38. DOI 10.1155/2011/143732.
17. Da Silva, T. G., Gueye, S., Michelon, P., Ochi, L. S., Cabral, L. D. A. F. (2019). A polyhedral approach to the generalized minimum labeling spanning tree problem. *EURO Journal on Computational Optimization, 7(1),* 47–77. DOI 10.1007/s13675-018-0099-5.
18. Lones, M. A. (2014). Metaheuristics in nature-inspired algorithms. *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation,* pp. 1419–1422. New York, NY, USA: ACM.
19. Ma, T., Zhou, H., Jia, D., Al-Dhelaan, A., Al-Dhelaan, M. et al. (2019). Feature selection with a local search strategy based on the forest optimization algorithm. *Computer Modeling in Engineering & Sciences, 121(2),* 569–592. DOI 10.32604/cmes.2019.07758.
20. Venkatakrishnan, G., Rengaraj, R., Salivahanan, S. (2018). Grey wolf optimizer to real power dispatch with non-linear constraints. *Computer Modeling in Engineering & Sciences, 115(1),* 25–45.
21. Xiong, Y., Golden, B., Wasil, E. (2005). A one-parameter genetic algorithm for the minimum labeling spanning tree problem. *IEEE Transactions on Evolutionary Computation, 9(1),* 55–60. DOI 10.1109/TEVC.2004.840145.
22. Nummela, J., Julstrom, B. A. (2006). An effective genetic algorithm for the minimum-label spanning tree problem. *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation,* pp. 553–558. New York, NY: ACM.
23. Cerulli, R., Fink, A., Gentili, M., Voß, S. (2005). Metaheuristics comparison for the minimum labelling spanning tree problem. In *The Next Wave in Computing, Optimization, and Decision Technologies*, pp. 93–106. Boston, MA: Springer US.
24. Chwatal, A. M., Raidl, G. R. (2010). Solving the minimum label spanning tree problem by ant colony optimization. *Proceedings of the 7th International Conference on Genetic and Evolutionary Methods,* pp. 91–97. Las Vegas Nevada, USA: CSREA Press.
25. Consoli, S., Pérez, J. M. (2012). Solving the minimum labelling spanning tree problem using hybrid local search. *Electronic Notes in Discrete Mathematics, 39,* 75–82. DOI 10.1016/j.endm.2012.10.011.

26. Lai, X., Zhou, Y., He, J., Zhang, J. (2013). Performance analysis of evolutionary algorithms for the minimum label spanning tree problem. *IEEE Transactions on Evolutionary Computation, 18(6),* 860–872.
27. Consoli, S., Mladenović, N., Pérez, J. M. (2015). Solving the minimum labelling spanning tree problem by intelligent optimization. *Applied Soft Computing, 28,* 440–452. DOI 10.1016/j.asoc.2014.12.020.
28. Da Silva, T. G., Queiroga, E., Ochi, L. S., Cabral, L. D. A. F., Gueye, S. et al. (2019). A hybrid meta-heuristic for the minimum labeling spanning tree problem. *European Journal of Operational Research, 274(1),* 22–34. DOI 10.1016/j.ejor.2018.09.044.
29. Gao, H., Pun, C. M., Kwong, S. (2016). An efficient image segmentation method based on a hybrid particle swarm algorithm with learning strategy. *Information Sciences, 369,* 500–521. DOI 10.1016/j.ins.2016.07.017.
30. Yang, X. (2009). Firefly algorithms for multimodal optimization. In Watanabe, O., Zeugmann, T. (Eds.) *Stochastic Algorithms: Foundations and Applications*. Berlin Heidelberg: Springer.
31. Yang, X. (2010). *Nature-inspired metaheuristic algorithms*. Frome, UK: Luniver Press.
32. Yang, X. S. (2014). Cuckoo search and firefly algorithm: overview and analysis. *Cuckoo search and firefly algorithm*. pp. 1–26. Cham, Germany: Springer.
33. Yang, X. S., He, X. (2013). Firefly algorithm: recent advances and applications. *International Journal of Swarm Intelligence, 1(1),* 36–50. DOI 10.1504/IJSI.2013.055801.
34. Sayadi, M. K., Ramezanian, R., Ghaffari-Nasab, N. (2010). A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations, 1(1),* 1–10. DOI 10.5267/j.ijiec.2010.01.001.
35. Chandrasekaran, K., Simon, S. P. (2012). Network and reliability constrained unit commitment problem using binary real coded firefly algorithm. *Electrical Power and Energy Systems, 43(1),* 921–932. DOI 10.1016/j.ijepes.2012.06.004.
36. Poursalehi, N., Zolfaghari, A., Minuchehr, A. (2015). A novel optimization method, effective discrete firefly algorithm, for fuel reload design of nuclear reactors. *BMC Bioinformatics, 81,* 263–275.
37. Zhang, J., Gao, B., Chai, H., Ma, Z., Yang, G. (2016). Identification of DNA-binding proteins using multi-features fusion and binary firefly optimization algorithm. *BMC Bioinformatics, 17(323),* 1–12. DOI 10.1186/s12859-015-0844-1.
38. Tilahun, S. L., Ngnotchouye, J. M. T. (2017). Firefly algorithm for discrete optimization problems: a survey. *KSCE Journal of Civil Engineering, 21(2),* 535–545. DOI 10.1007/s12205-017-1501-1.
39. Tarjan, R. E. (1974). A note on finding the bridges of a graph. *Information Processing Letters, 2(6),* 160–161. DOI 10.1016/0020-0190(74)90003-9.