Nat Comput (2017) 16:85–98
DOI 10.1007/s11047-016-9545-6

A new genetic algorithm based on modified *Physarum* network model for bandwidth-delay constrained least-cost multicast routing

Mingxin Liang[1] · **Chao Gao**[1,2] · **Zili Zhang**[1,3]

Published online: 29 March 2016

Abstract A mobile ad hoc network is a kind of popular self-configuring network, in which multicast routing under the quality of service constraints, is a significant challenge. Many researchers have proved that such problem can be formulated as a NP-complete problem and proposed some swarm-based intelligent algorithms to solve the optimal solution, such as the genetic algorithm (GA), bees algorithm. However, a lower efficiency of local search ability and weak robustness still limit the computational effectiveness. Aiming to those shortcomings, a new hybrid algorithm inspired by the self-organization of *Physarum*, is proposed in this paper. In our algorithm, an updating scheme based on *Physarum* network model (PM) is used for improving the crossover operator of traditional GAs, in which the same parts of parent chromosomes are reserved and the new offspring by the PM is generated. In order to estimate the effectiveness of our proposed optimized scheme, some typical genetic algorithms and their updating algorithms (PMGAs) are compared for solving the multicast routing on four different datasets. The simulation experiments show that PMGAs are more efficient than original GAs. More importantly, the PMGAs are more robustness that is very important for solving the multicast routing problem. Moreover, a series of parameter analyses is used to find a set of better setting for realizing the maximal efficiency of our algorithm.

Keywords Multicast routing · Genetic algorithm · *Physarum* network model

✉ Chao Gao
cgao@swu.edu.cn

✉ Zili Zhang
zhangzl@swu.edu.cn

1 School of Computer and Information Science, Southwest University, Chongqing 400715, China

2 Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China

3 School of Information Technology, Deakin University, Locked Bag 20000, Geelong, VIC 3220, Australia

1 Introduction

A mobile ad hoc network (MANET) is composed of self-organizing mobiles in dynamic topology networks. All nodes cooperatively maintain network connectivity without the aid of any fixed infrastructure unit. The communication between two nodes is carried out either directly or with the help of intermediate nodes which belong to the same network (Wang et al. 2001). Currently, MANETs are very popular due to the no-restricted mobility and feasible deployment (Sesay et al. 2004).

Multicasting is one type of services in MANETs (Oliveira and Pardalos 2005). With the growing of distributed multimedia application, the efficient and effective support of QoS (i.e., Quality of Service) has became more and more crucial for MANETs. The basic purpose of QoS routing is to find paths with better QoS. Typical QoS properties include the end-to-end delay, packet loss probability, delay jitter, available bandwidth and so on. In general, these QoS properties can be classified into three categories: additive metrics (e.g. end-to-end delay), multiplicative metrics (e.g. packet loss probability), and concave metrics (e.g. available bandwidth) (Yen et al. 2011). And there is a balance when we design a network architecture. First, some constraints of QoS have to be taken into consideration, which make QoS routing more complex than a regular one. Second, resource utilization is the most important metric for the efficiency of a routing network, which affects both the QoS and occupied resources of a routing (Wang et al. 2001). Therefore, the key issue in the design of a network is how to efficiently manage the resources and meet the requirements of QoS during each connection.

Routing problems can be divided into the unicast and multicast routing. The unicast routing is to find a feasible path between a single source and a single destination. Meanwhile, the multicast routing aims to find a tree structure, which is used for efficiently delivering the same data stream to different destinations in a network simultaneously. More importantly, in a QoS routing, some QoS constrains are added to the entire tree. Thus, the object of multicast routing problem is to compute a tree structure, called the multicast tree, which has a minimal communication resources and meets QoS requirements (Peng and Li 2013). Like many other networks designs, the updating rule of "choosing the best" strategy has a detrimental impact on the outcome (Wang et al. 2012). Thereby, an optimization algorithm is necessary. The QoS multicast routing problem is also called the steiner tree problem, which is a typical NP-complete problem (Wang and Crowcroft 1996). It means this problem cannot be solved optimally with a polynomial time complexity, which is very crucial for application in the real world. Therefore, providing QoS guarantees for data traffic and minimizing the cost of whole network on multicast routing are significant challenges.

As the QoS multicast routing problem has draw more and more attentions, many researchers have used different methods to solve this problem. Ratnasamy et al. (2006) have tried to modify the existing algorithms to satisfy QoS constraints by best effort forwarding. But these algorithms cannot guarantee a multicast tree for QoS constraints. Some researchers have used swarm intelligence algorithms to solve this problem, such as genetic algorithms (GA) (Peng and Li 2013; Hwang et al. 2000), bee life-based algorithms (Salim and Abdelhamid 2013). Those algorithms first analyze the fitness of tree structures based on the whole network topology, and then optimize those tree structures by some meta-heuristic operators. But a weak robustness and low efficiency of local search still limit the performances of those algorithms.

Recently, more and more researchers focus on the self-organization capability of a species of plasmodium, which is a 'vegetative' phase of *Physarum*. This plasmodium shows an amazing intelligence in foraging, which can produce a robust protoplasmic network for connecting food sources in order to deliver nutrients to all its body effectively (Nakagaki et al. 2000). Moreover, biological experiments have shown that the plasmodium has the ability to form a self-adaptive and high efficient network without central control mechanism (Adamatzky 2009). The self-organization of *Physarum* is what MANETs need. Taking the self-organization capability of *Physarum*, a new crossover scheme is proposed in this paper. Based on the multi-constrained genetic algorithmic approaches, we aim to optimize the cost and QoS property metrics simultaneously. Utilizing *Physarum* model, our proposed scheme can enhance the robustness and local search ability of genetic algorithms. Simulation results demonstrate that the proposed scheme improves the efficiency and robustness of genetic algorithms.

The organization of this paper is as follows. Sect. 2 introduces the related work including the formulation of multicast routing problem, some typical genetic algorithms and the original *Physarum* network model. Section 3 proposes the hybrid algorithm through optimizing traditional crossover schemes of genetical algorithms based on the *Physarum* network model. Section 4 provides some experiments to estimate the effectiveness of proposed hybrid algorithm and analyzes the parameters of modified *Physarum* network model. Finally, Sect. 5 concludes this paper.

2 Related work

Section 2.1 first formulates the multicast routing problem. And Sect. 2.2 describes the basic idea of traditional algorithms and genetic algorithms for the multicast routing problem. Sect. 2.3 presents the main mechanism of original *Physarum* networks model.

2.1 Formulation of multicast routing problem

A network is usually represented as an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes a set of nodes representing routers or switches and $E = \{e_{ij} = (v_i, v_j) | v_i, v_j \in V, i \neq j\}$ denotes a set of edges representing physical or logical connectivity between nodes. In a multicast routing, there are a source node and destination nodes. Let a node $s \in V$ stand for the source and a set $DE \subseteq V - \{s\}$ represent the set of destinations. And then a multicast tree can be denoted as $T(s, DE)$, which is a connected sub-graph of G and covers node s and every node in DE. The path from a node s to any destination node $d \in DE$ is denoted as $p_T(s, d)$. And the object of multicast routing problem is to find a $T(s, DE)$ with a minimal cost, which has a set of paths with acceptable QoS metrics.

As introduced in Sect. 1, QoS metrics can be classified into three categories, i.e., additive metrics, multiplicative metrics and concave metrics. And additive and multiplicative metrics can be deal with in the same way in our algorithm. For simplifying the QoS routing problem, we focus on additive metrics (e.g. end-to-end delay) and concave metrics (e.g. available bandwidth). Furthermore, the cost is the most important property for the effectiveness of a routing, thus our research focuses on the bandwidth-delay constrained least-cost multicast routing problem.

The delay of a path from a node s to any destination d in DE, denoted by $delay(p_T(s,d))$, is defined as the sum of delays in the $p_T(s,d)$, i.e., $delay(p_T(s,d)) = \sum_{e\in p_T(s,d)} delay(e)$. Meanwhile, the bandwidth of a path from a node s to any destination node d in DE, denoted by $band\ width(p_T(s,d))$, is defined as the minimum of bandwidth along the path, i.e., $bandwidth(p_T(s,d)) = \min\{bandwidth(e)|e \in p_T(s,d)\}$. And, according to existing studies in Hwang et al. (2000); Lu and Zhu 2013), we unify the cost of a multicast tree as the sum of costs in the tree, i.e., $cost(T) = \sum_{e\in T} cost(e)$, for comparing the effectiveness of different GAs. Moreover, the delay and bandwidth of a multicast tree T are defined as Eqs. (1) and (2), respectively.

$$delay(T(s,DE)) = \max\{delay(p_T(s,d))|d \in DE\} \quad (1)$$

$$bandwidth(T(s,DE)) = \min\{bandwidth(p_T(s,d))|d \in DE\} \quad (2)$$

Let Δ_d be the upper bound of delay constraint and Δ_b be the lower bound of bandwidth constraint for every path. And the bandwidth-delay constrained least-cost multicast routing problem is defined as Eq. (3). We aim to minimize the cost under the condition of bandwidth-delay constraints.

$$\begin{aligned} \min \quad & cost(T) \\ st. \quad & \begin{cases} delay(T) \le \Delta_d \\ bandwidth(T) \ge \Delta_b \end{cases} \end{aligned} \quad (3)$$

2.2 Algorithms for solving the multicast routing problem

There are two main approaches, namely, exact methods and heuristics, for solving the QoS multicast routing problem. Exact methods are usually based on mathematical programming, among which the dynamic programming (Chow 1991) and branch-and-bound (Salama et al. 1997) are the two most prevailing exact methods for QoS multicast routing. However, due to the high computational complexity, those exact methods are only viable for small-scale problems (Yin et al. 2014). With the development of networks, heuristics are the mostly approaches for the QoS multicast routing problem.

Capturing the features of genetic evolution, GA is a powerful tool for solving NP-complete problems. By setting an appropriate maximal iterative steps of GA, an approximate optimal solution can be obtained within a reasonable time. In genetic algorithms, candidate solutions are coded as chromosomes. Moreover, the idea of natural selection and genetic operators, such as the crossover and mutation, are employed for searching better chromosomes. Many researchers have applied GAs to solve the multicast routing problem with various coding and genetic operators (Peng and Li 2013; Hwang et al. 2000; Lu and Zhu 2013; Karthikeyan and Baskar 2015; Mahmoud et al. 2014). In the following, we will take some typical genetic algorithms [i.e., GAMRA (Hwang et al. 2000), EEGA (Lu and Zhu 2013), ISGSA (Zhang et al. 2009)] as examples to describe the solving process of different genetic algorithms for the multicast routing problem.

To deal with the end-to-end delay constraint, Hwang et al. (2000) have proposed an optimized GA, denoted as GAMRA, where a routing table that stores some paths with acceptable delays, has to be constructed first. But constructing such a routing table needs a lot of time. And the crossover operator of GAMRA is a classical two-point crossover, which leads to a weak robustness and lower convergence rate. Wang et al. (2001) have proposed that it is helpful to reserve the same links of two parent chromosomes for the convergence of genetic algorithm. And GAISA (Zhang et al. 2009) and EEGA (Lu and Zhu 2013) have applied this idea to optimizing their crossover operators. Because the same links of parent chromosomes may be in some separated connected components, different schemes have been proposed to reconnect those connected components, in GAISA and EEGA. For GAISA, the reconnection is building a random path from destination nodes, which are separated with the source node, to the connected component including the source node. And for EEGA, two separated components are reconnected by the least-delay path. Although different genetic algorithms have been proposed to solve the multicast routing problem, weak robustness and a lower efficiency of local search capability still limit the performances of genetic algorithms. And making use of knowledge of experts or data, mathematic models are valid methods for improve performances of algorithms (Yu et al. 2011, 2014, 2015).

In order to compare the effectiveness and robustness of different algorithms, some measurements are defined as follows. S_{min}, $S_{average}$ and $S_{variance}$ stand for the minimum, average, and variance of results, respectively. Results are all based on C times repeated computations in order to wipe off the computational fluctuation. For example, S_{min} is calculated as $\min\{S_{i,steps(k)}, i = 1, 2, \ldots, C\}$, where $S_{i,steps(k)}$ represents the optimal solution of the multicast routing problem in the k step for the ith time computation.

2.3 Original *Physarum* model

Physarum has shown an amazing intelligent and self-organization capability. For example, Tero et al. have reported that *Physarum* is cultivated to simulate the Tokyo rail system on a substrate (Tero et al. 2010). And the network generated by *Physarum* has a strong robustness, great fault tolerance and high transport effectiveness on the transportation (Watanabe et al. 2011). And Nakegaki et al. have shown that *Physarum* has the ability to find the minimum-length solution between two points in a maze (Nakagaki et al. 2000).

The *Physarum* network model is inspired by the maze-solving experiment (Nakagaki et al. 2000). Tero et al. have captured the positive feedback mechanism of *Physarum* in foraging, and built the *Physarum* network model (PM) (Tero et al. 2007). In addition, The model, designed for solving maze problems, has been used for the network design (Liu et al. 2013) and complex problems solving (Liu et al. 2014). Moreover, this model can be used for building multicast trees in our study.

The main idea of *Physarum* network model is summarized as follows. We assume the edges of a network are pipelines with fluid inside. Q_{ij} and D_{ij} stand for the flux and conductivity of a pipeline respectively, which connects nodes i and j. When the conductivity of a pipeline is enhancing, the flux in the pipeline will be enhanced correspondingly, vice versa.

In detail, we use a node s and a set DE to present the inlet and outlets of flux respectively. According to the Kirchhoff′s law, the flux of input at the node s is equal to the total flux of output at all noes in DE. And, at any other nodes, the sum of flux flowing in is equal to the sum of flux flowing out. This process can be represented in Eq. (4) where N stands for the cardinality of set DE.

$$\sum_i Q_{ij} = \begin{cases} I_0 \times (N-1), & j == s \\ -I_0, & j \in DE \\ 0, & others \end{cases} \tag{4}$$

In each iterative step, Q_{ij} and p_i can be calculated according to Poiseuille′s law based on Eqs. (4) and (5), where L_{ij} represents the length of pipeline contacting nodes i and j, and p_i represents the pressure of node i. Equation (6) is called adapted equation. As the iteration going on, the conductivities of pipelines adapt to the flux based on Eq. (6). Then, the conductivities will feed back to the flux based on Eq. (5) at the next iterative step. Two typical functions of $f(Q)$ are shown in Eqs. (7) and (8) based on Tero et al. 2007.

$$Q_{ij} = \frac{D_{ij}}{L_{ij}} |p_i - p_j| \tag{5}$$

$$\frac{dD_{ij}}{dt} = f(|Q_{ij}|) - D_{ij} \tag{6}$$

$$f(Q) = Q^u, \quad u > 0 \tag{7}$$

$$f(Q) = \frac{(1+\alpha)Q^u}{1+\alpha Q^u}, \quad u > 1, \alpha > 0 \tag{8}$$

After above processes, an iterative step is completed. This process will continue a loop iteration until the terminal condition is satisfied. After the loop iteration, based on the positive feedback mechanism between the conductivity and flux, critical pipelines will be reserved, and others will disappear. Finally, we obtain a *Physarum* spanning tree.

3 Genetic algorithm for the multicast routing problem

In this section, a modified *physarum* networks model is proposed in Sect. 3.1 first. And then, Sect. 3.2 introduces a new crossover scheme based on PM for the multicast routing problem. Finally, Sect. 3.3 presents the genetic algorithms with the proposed crossover scheme.

3.1 A modified *Physarum* networks model for multicast routing problem

In the original *Physarum* model, an equation system should be solved by the Gaussian elimination at every iterative step in the original *Physarum* model, i.e., each iteration runs in $O(N^3)$. Assuming the maximal iteration of *Physarum* model is G. For an iterative algorithm, an overall computational complexity, $O(G \times N^3)$, is too high. In Liu et al. (2015), an approximate expression of p_i is derived as Eq. (9) to reduce the computational complexity.

$$p_i{}^{t+1} = \begin{cases} \dfrac{I_0 + \sum_j \frac{D_{ij}}{L_{ij}} (p_j{}^t)}{\sum_j \frac{D_{ij}}{L_{ij}}}, & v_i \in DE \\ 0, & v_i == s \\ \dfrac{\sum_j \frac{D_{ij}}{L_{ij}} (p_j{}^t)}{\sum_j \frac{D_{ij}}{L_{ij}}}, & otherwise \end{cases} \tag{9}$$

But in our numerical experiments, the pressures calculated by Eq. (9) fluctuate violently and cannot converge sometimes. Based on Eqs. (9), (10) is used to eliminate the fluctuation of pressures, where λ controls the fluctuations of pressures and conductivities.

$$p_i^{t+1} = \begin{cases} \dfrac{I_0 + \sum_j \frac{D_{ij}}{L_{ij}}(p_j{}^t)}{\sum_j \frac{D_{ij}}{L_{ij}}} * \lambda + (1-\lambda) * p_i{}^t, & v_i \in DE \\ 0, & v_i == s \\ \dfrac{\sum_j \frac{D_{ij}}{L_{ij}}(p_j{}^t)}{\sum_j \frac{D_{ij}}{L_{ij}}} * \lambda + (1-\lambda) * p_i{}^t, & otherwise \end{cases} \tag{10}$$

And the adapted equation in our paper is represented in Eq. (11), where k is a parameter relative to the amplitude of conductivities in the *Physarum* network.

$$D_{ij}{}^{t+1} = \frac{D_{ij}^t + Q_{ij}^t}{k} \tag{11}$$

Based on Eqs. (5) and (10), the adapted equation is equivalent to Eq. (12). And Eqs. (4) and (5) in the original *Physarum* networks model do not have to be solved.

$$D_{ij}^{t+1} = \frac{L_{ij}D_{ij}^t + D_{ij}^t|p_i^t - p_j^t|}{k} \tag{12}$$

By setting more than one destination node, i.e., $|DE| > 1$, the *Physarum* network model can find the shortest paths from the source node to every destination node in DE simultaneously. Therefore, in the multicast routing problem, PM can be used for building multicast trees. The detailed description of modified *Physarum* network model for the shortest path tree is shown in Algorithm 1.

Algorithm 1 *Physarum* network model

```
Input: A graph $NG$, a source node $s$, a set of destination nodes $DE$.
Output: The shortest path tree connecting the source node $s$ to every node in $DE$.
Step 1: Initializing with $D_{ij} = (0, 1]$, $p_i = 0$.
Step 2: Computing $p_i$ based on Eq. (10).
Step 3: Updating $D_{ij}$ based on Eq. (12).
Step 4: If the terminal condition is not satisfied,
            going to step 2.
        Else going to step 5.
Step 5: Outputting the shortest path tree.
```

3.2 A new crossover scheme based on PM for multicast routing problems

Generally speaking, there are two main genetic operators in a genetic algorithm. The first is the crossover operator and the other is the mutation operator. These two genetic operators allow genetic algorithms to search for the global optimum based on an evolutionary process. In an evolution process, the crossover operator is the main way by which chromosomes exchange informations. And the major purpose of mutation operator is to help algorithms avoid falling into the local optimum. Thereby the crossover operator plays a key role in searching better chromosomes. In the crossover operator, two chromosomes with greater fitness values are selected from the chromosomes pool as parent chromosomes. And the crossover operator combines these genes in the parent chromosomes into new ones. In other words, it is one of genetic operators in which the genotypes of two selected parent chromosomes are merged to generate new offsprings. The new offsprings will be put back into the chromosomes pool. Furthermore, the mutation operator is one kind of random change of genes in chromosomes. The mutation operator aims to extend the search range and reduce the possibility of falling into the local optimum. Through these operators, new offsprings with a higher fitness will be produced.

For overcoming the shortcomings of GAs, a universal scheme is proposed for optimizing the crossover operator of GAs, taking advantages of PM. The new crossover operator is called PMcrossover. And the novel hybrid algorithms with PMcrossover (denoted as PMGAs) are used for solving the multicast routing problem. The main idea of PMcrossover is to reserve the common links between parent chromosomes and integrate the offspring by PM based on the reserved links. Other parts of PMGAs are same as the original ones. An example of crossover process is shown in Fig. 1. The details of PMcrossover are described as follows.

First, for utilizing PM, a new graph, denoted as NG, needs to be created with the same topological structure as the network graph in the multicast routing problem. Because GAs may generate some unadaptable chromosomes, which do not satisfy the constraints. There are two strategies to fit for different GAs. For the GAs, which do not generate unadaptable chromosomes, such as GAMRA (Hwang et al. 2000), L_{ij} in NG is set equating to the delay of e_{ij}. For others, such as EEGA (Lu and Zhu 2013), ISGSA (Zhang et al. 2009), L_{ij} in NG is set equating to the product of cost and delay of e_{ij}.

And then, PMcrossover selects the same links of parent chromosomes and reserves them in the offspring chromosomes. According to the definition of fitness function, the selected parent chromosomes with higher fitness values are more likely to satisfy the constraints and have lower costs. Therefore, it is helpful to reserve the same links of two parent chromosomes for the convergence of algorithms. Because these same links may be in some separated sub-trees, PM is used to transform these sub-trees into a multicast tree. In order to reserve the same links in PM model, the length of reserved links is set to 0 in NG. And then, inputting the graph NG, source and destinations into PM, a complete multicast

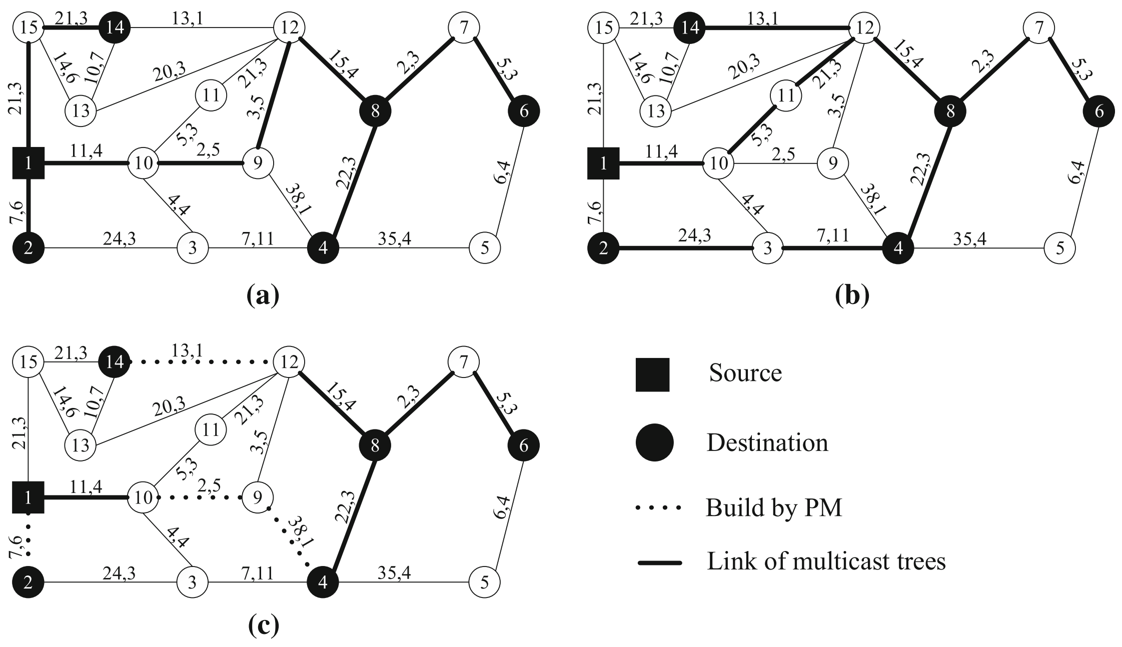


Fig. 1 An example of PMcrossover operation. The numbers in edges denote cost and delay respectively. **a**, **b** are selected as parent chromosomes. And the common links of selected parent chromosomes are reserved in the offspring chromosome as shown in (**c**). But the source and destination nodes are not in a connected component with reserved links only. Therefore, *Physarum* model builds the necessary links to reconnect source and destination. **a** Parent chromosome A. **b** Parent chromosome B. **c** A new offspring which is generated by parent chromosomes A and B based on PMcrossover nodes

tree will be constructed. Algorithm 2 describes detailed steps of PMcrossover operator in PMGAs.

Algorithm 2 PMcrossover

```
Input: A network graph $G$, a source node $s$, a set of destination nodes $DE$,
       parent chromosomes $T_a$ and $T_b$.
Output: An offspring chromosome $T_c$.
Step 1: Creating a graph $NG$ with the same topological structure as
        the network graph $G$ in a multicast routing problem.
Step 2: If the original crossover operator does not generate unadaptable
            chromosomes, Let $L_{ij}$ in $NG$ equal to the delay of $e_{ij}$.
        Else let $L_{ij}$ in $NG$ equal to the product of cost and delay of $e_{ij}$.
Step 2: Let the length of same links between $T_a$ and $T_b$ equal to 0, in $NG$.
Step 3: Inputting the $NG$, $s$ and $DE$ nodes into PM.
Step 4: Outputting a new multicast tree, $T_c$, based on PM.
```

3.3 Genetic algorithms with PMcrossover operator

In order to decrease the searching space and improve the effectiveness of genetic algorithms. A refining operator is employed to deal with the bandwidth constraint (Wang et al. 2001). The links with a bandwidth less than the QoS requirement will be removed by the refining operator. On the one hand, if the source and all destinations are not in a connected component in a refined graph, all paths in a network cannot satisfy the bandwidth constraint, which means the bandwidth constraint should be relaxed. On the other hand, if the source node and all destination nodes are in a connected component, the paths in a refined graph must satisfy the bandwidth constraint. The flow chart of typical genetic algorithms with the PMcrossover and refining operator is shown in Fig. 2.

Figure 2 shows that the new proposed PMcrossover scheme only changes the crossover part of original genetic algorithms. As mentioned above, different genetic algorithms have different crossover operators. For verifying the universality of our proposed crossover scheme, we integrate PMcrossover into three different genetic algorithms [i.e., GAMRA (Hwang et al. 2000), EEGA (citeal-t2Lu2013) ISGSA (Zhang et al. 2009)] and estimate the efficiency in the next section.

4 Experiments

Section 4.1 introduces the datasets used in this paper. And Sect. 4.2 shows and analyzes the results of different algorithms on these datasets. Parameters analysis is shown in Sect. 4.3, for a better performance. Finally, Sect. 4.4 reports the complexity analysis of PMcrossover.

4.1 Datasets

In order to estimate the effectiveness of PMcrossover scheme, we implement GAMRA, EEGA, ISGSA and their

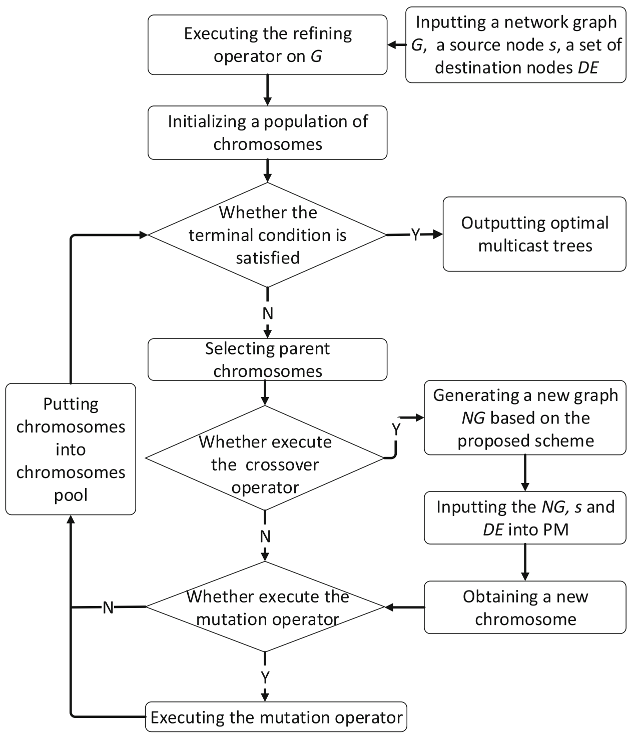

Inputting a network graph G, a source node s, a set of destination nodes DE
Executing the refining operator on G
Initializing a population of chromosomes
Whether the terminal condition is satisfied
Y
Outputting optimal multicast trees
N
Selecting parent chromosomes
Putting chromosomes into chromosomes pool
Whether execute the crossover operator
Y
Generating a new graph NG based on the proposed scheme
Inputting the NG, s and DE into PM
N
Obtaining a new chromosome
Whether execute the mutation operator
N
Y
Executing the mutation operator

Fig. 2 The flow chart of typical PMGAs

updating algorithms on four datasets. The first dataset (denoted as $D1$) is a 20-node random graph which is constructed based on Hwang et al. (2000). In $D1$, costs and delays of links are uniformly distributed between 0.3 and 1. The second (denoted as $D2$) is shown in Fig. 1, which is cited from Lu and Zhu (2013). For verifying the scalability of PMcrossover scheme, the third dataset (denoted as $D3$) is adopted. $D3$ is a synthetic graph generated by the Salama graph generator (Salama 1996), where there are 50 nodes and costs of links are generated between 3 and 10. Moreover, for comparing with other meta-heuristic algorithms, we implement EEGA, ISGSA and their updated algorithms on the fourth dataset (denoted as $D4$), which is came from Salim and Abdelhamid (2013) and constructed by the network simulator[1]. The details and thresholds of $D1$, $D2$, $D3$ and $D4$ are shown in the "Appendix".

All experiments are under the same environment, i.e., all parameters of PMGAs are same as these of original GAs. And the results on $D1$, $D2$ and $D3$ in our experiments are based on 50 repeated experiments in order to wipe off the computational fluctuation. For comparing with the algorithms in Salim and Abdelhamid (2013), some results on $D4$ are based on ten repeated experiments.

[1] http://www.isi.edu/nsnam/ns/.

4.2 Experimental analyze

Figure 3 reports the $S_{\min}$, $S_{average}$ and S_{variance} of results calculated by PM-GAMRA and original GAMRA (Hwang et al. 2000) on $D1$. Although these two GAs can find approximate optimal solutions, $S_{\min}$ and $S_{average}$ of PM-GAMRA are less than that of original GAMRA, which means that PM-GAMRA has a stronger ability to exploit the optimal solution. Moreover, S_{variance} of PM-GAMRA is less than that of GAMRA, which shows that the PM-GAMRA has a stronger robustness.

As EEGA and ISGSA may generate unadaptable solutions in the evolution, we compare the costs and delays of results simultaneously. Figure 4a, b show $S_{\min}$, $S_{average}$ and S_{variance} of costs and delays respectively, where $S_{\min}$, $S_{average}$ and S_{variance} of PM-EEGA and PM-ISGSA are less than those of EEGA and ISGSA in both costs and delays. These results show that the PMcrossover scheme can strength the searching ability for finding the optimal solution and improve the robustness of original GAs.

In order to further verify the accuracy and robustness of PMGAs, Figs. 5 and 6 plot averages and variances of costs and delays with the increment of generations in the convergent process. Results show that the averages and variances of PM-EEGA and PM-ISGSA decrease more obviously than those of EEGA and ISGSA. In detail, in the earlier iteration, there is slight difference between averages of PMGAs and GAs. The initial delays of PM-EEGA and PM-ISGSA are even higher than those of EEGA and ISGSA. With the increment of generations, the averages of PM-EEGA and PM-ISGSA are less than that of EEGA and ISGSA. PM-EEGA and PM-ISGSA exhibit a better accuracy. Furthermore, the variances of PM-EEGA and PM-ISGSA are also less than those of EEGA and ISGSA, which indicates that PM-EEGA and PM-ISGSA have much stronger robustness.

To demonstrate the scalability of PMcrossover scheme, we implement four GAs (i.e., EEGA, PM-EEGA, ISGSA, PM-ISGSA) on $D3$ with the same parameters setting. For a better performance, parameters are set based on Table 1. Figure 7 shows that $S_{\min}$ and $S_{average}$ of PM-GAMRA, PM-EEGA, PM-ISGSA are all less than that of original ones. Furthermore, S_{variance} of PM-GAMRA and PM-ISGSA are less than half those of GAMRA and ISGSA. Although S_{variance} of PM-EEGA is higher than that of EEGA, the difference is too slight to have an effect on the performance of algorithms. These results indicate that GAs with PMcrossover exhibit more stronger ability to find optimal solutions. And PMcrossover scheme still work well when the scale of problem is increasing.

For further verifying the effectiveness of proposed scheme and comparing with other meta-heuristic

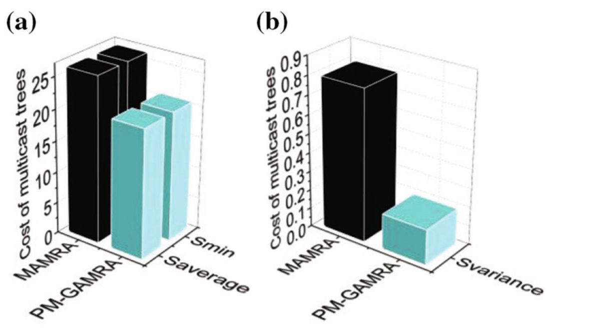


Fig. 3 Comparing costs of results of PM-GAMRA and GAMRA on *D*1. **a**, **b** report the average, minimum and variance of cost, respectively. Compared with GAMRA, PM-GAMRA shows a better search ability and a stronger robustness

algorithms, we implement EEGA, ISGSA and their updated algorithms on *D*4 and add a new constraint on jitter, which is also a typical QoS property. In this paper, the jitter of a path (denoted as $jitter(p_T(s,d))$) and the jitter of a tree (denoted as $jitter(T)$) are defined as Eqs. (13) and (14) respectively. Some meta-heuristic algorithms (i.e., BLA (bees life-based algorithm) (Salim and Abdelhamid 2013), BA (bees algorithm) (Pham et al. 2006), MBO (marriage in honey bees optimization algorithm) (Abbass 2001) are used to compare the efficiency with our proposed methods. And the fitness values of those algorithms (as listed in Fig. 8a) are extracted from Salim and Abdelhamid (2013) directly. Moreover, the fitness and property values in Table 2 are calculated based on *D*4 and topological structures of best results shown in Salim and Abdelhamid (2013), which are generated by those algorithms.

$$jitter(p_T(s,d)) = \sum\nolimits_{e \in p_T} jitter(e) \quad (13)$$

$$jitter(T) = \max\{jitter(p_T(s,d))|d \in DE\} \quad (14)$$

$$f(T) = w_1 f_c(T) + w_2 f_d(T) + w_3 f_j(T) + w_4 f_b(T) \quad (15)$$

As the fitness function is an important part of GA, we retain the original fitness functions, when algorithms run, and calculate new fitness values based on Eq. (14) for a fair comparison. In Eq. (14), $f_c(T), f_d(T), f_j(T)$ and $f_b(T)$ stand for the cost, delay, jitter, bandwidth of a multicast tree T, respectively. Meanwhile, w_i represents the objective weighting coefficient. The objective weighting coefficients

Fig. 4 Comparing results of GAs and PMGAs on *D*2. **a**, **b** report the cost and delay comparisons, respectively. Compared with GAs, PMGAs can find better solutions with stronger robustness

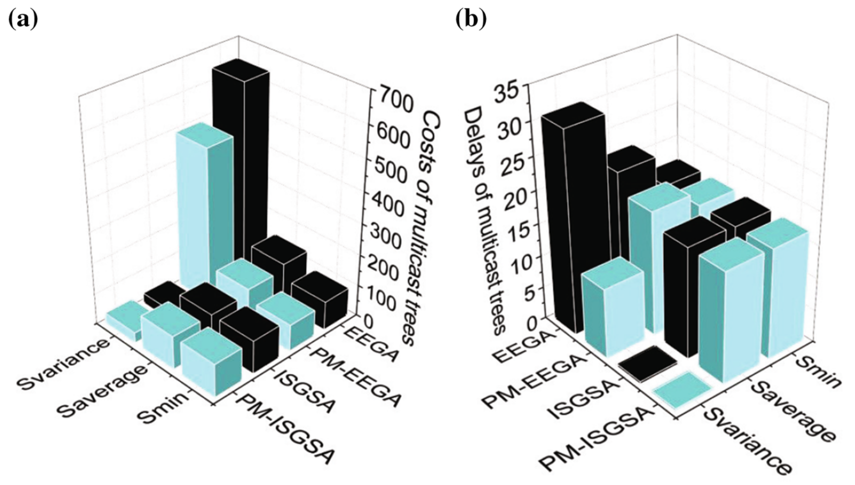
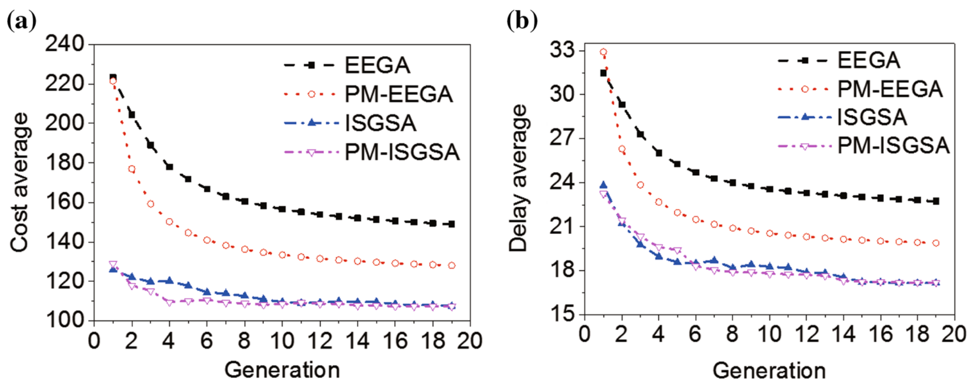


Fig. 5 The averages of delays and costs calculated by GAs and PMGAs on *D*2 are plotted in (**a**, **b**) respectively. Results show that delay and cost averages of PMGs have a higher descent rate than those of GAs



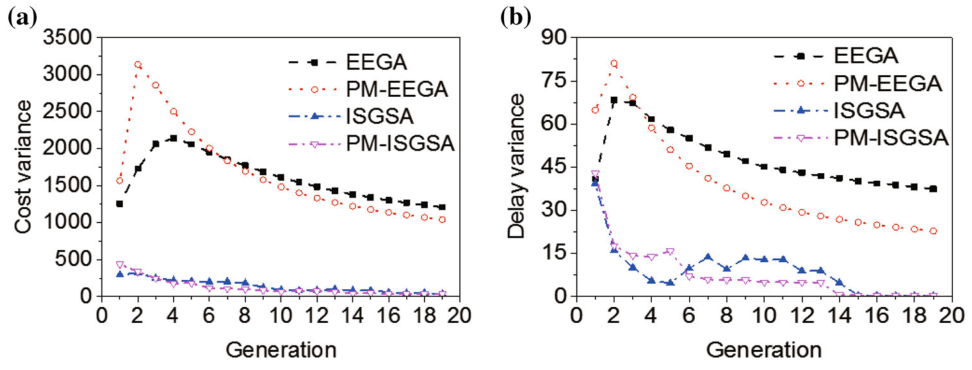


Fig. 6 The variances of delays and costs calculated by GAs and PMGAs on $D2$ are plotted in (**a**, **b**), respectively. Results indicate that PMGAs have a lower variances than those of GAs, which means that PMGAs have a stronger robustness

Table 1 Parameters setting of GAs

Parameter	Explanation	Value
PS	The popular size of generations	64
MG	Maximal generation	50
p_c	Crossover probability	1
p_m	Mutation probability	0.3
NR	The number of routs in the routing table	100

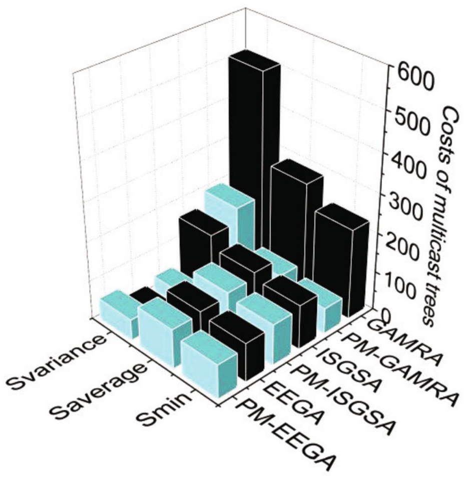


Fig. 7 Comparison costs of results on $D3$. Results show that PMcrossover can reduce the average cost of results and enhance the robustness of GAs

and thresholds of constraints are set based on Salim and Abdelhamid (2013).

Table 2 lists the fitness value, cost, delay, jitter and bandwidth of the best multicast trees generated by different algorithms. As shown in Table 2, BLA, EEGA and PM-EEGA converge to the multicast tree with the lowest cost. Meanwhile, ISGAS and PM-ISGAS converge to the multicast tree with the lowest fitness value.

To further compare the efficiencies of those algorithms, the averages and variances of fitness values, which are generated by 10 repeated experiments, are calculated and shown in Fig. 8a, b respectively. Although the averages of EEGA and ISGSA are close to those of their updated algorithms, variances of EEGA, ISGSA and those of their updated algorithms have a significant difference. As shown in Fig. 8b, variances of PM-EEGA and PM-ISGSA are much lower than those of EEGA and ISGSA. It means PMcrossover still can enhance the robustness of original algorithms, when the original algorithms can find optimal solutions.

4.3 Parameters analysis

As mentioned in Sect. 3.1, there are two parameters, λ and k, which affect the amplitude of conductivities in the *Physarum* network. In order to further analyze the convergence rate of *Physarum* model, we run this model with different λ and k values. Because the computational time is affected by computing environments which are hard to be fully controlled, we use iterative steps instead of computational time to compare the convergence rate. We remark the iterative steps with different k and λ values on $D2$ and $D3$. The iterative steps are recorded when the terminal condition is satisfied. In this paper, the terminal condition is that iterative steps beyond the maximal iterative steps or Eq. (16) is satisfied (Zhang et al. 2014), where $D_{ij}{}^t$ stands for D_{ij} at iterative step t. And the maximal iterative steps are set as 10000 and 50000 when the model runs on $D2$ and $D3$ respectively.

$$Max\{|D^t{}_{ij} - D^{t-1}{}_{ij}||\forall i,j\} < 10^{-6} \quad (16)$$

As shown in Fig. 9, iterative steps of *Physarum* model are more sensitive to λ rather than k. The relationship between k and iterative steps is simple. The higher k is, the lower iterative steps are. But if k is too high, conductivities in the *Physarum* networks will fall sharply, which may lead to the disappearance of some crucial links. For the parameter λ, 0.1 is a special value. When λ is equal to 0.1, iterative steps reach the maximal iterative steps, except when k is equal to 1.2. It means, in most situations, the

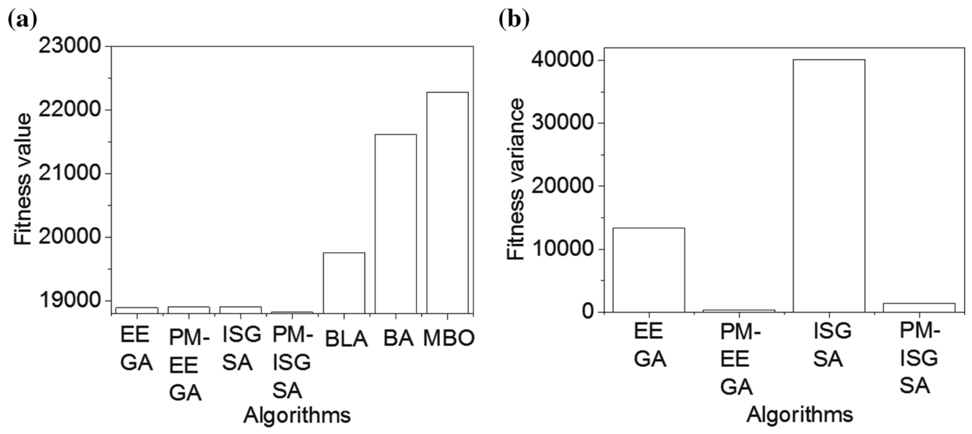


Fig. 8 The averages and variances of fitness calculated by different algorithms on *D*4 are plotted in (**a**, **b**) respectively. The average fitness values of GAs and PMGAs are lower than those of other algorithms. And the gap between variances of PMGAs and GAs is significant

Table 2 Fitness and property values of best results generated by different algorithms on *D*4. The best values of properties are emphasized by boldtype

Algorithms	BLA	BA	MBO	EEGA	PM-EEGA	ISGSA	PM-ISGSA
Fitness	18905.29	20266.64	18987.34	18905.29	18905.29	**18822.64**	**18822.64**
Cost	**1740.38**	1876.53	1762.65	**1740.38**	**1740.38**	1746.19	1746.19
Delay	616.34	616.13	**493.14**	616.34	616.34	**493.14**	**493.14**
Jitter	8.75	8.75	**7.00**	8.75	8.75	**7.00**	**7.00**
Bandwidth	797.54	797.65	797.57	797.54	797.54	797.54	797.54

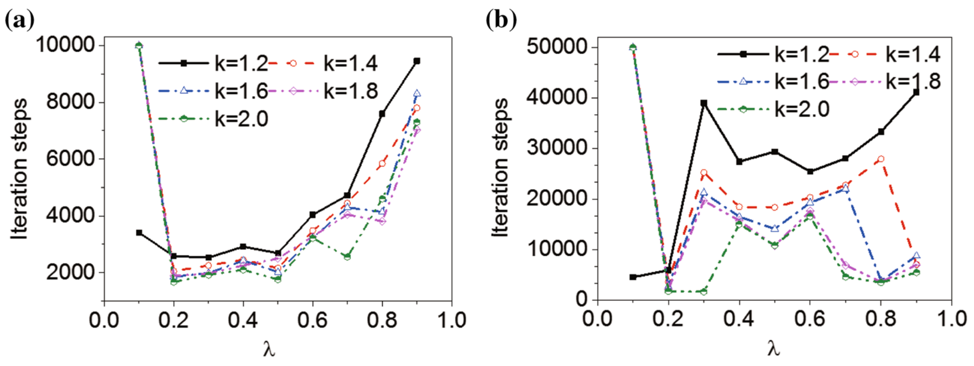


Fig. 9 Iterative steps of *physarum* model with different λ and *k*. Based on those results, best parameters setting of *Physarum* model can be obtained. **a**, **b** report the iterative steps of *physarum* model with the increment of λ under different *k* value on *D*2 and *D*3 respectively

Physarum model cannot converge when the iterative steps reach the maximal iterative steps. As shown in Fig. 9a, the higher λ is, the higher iterative steps are. But in Fig. 9b, iterative steps have a decline at the final stage with the increment of λ. And the reason of this decline is that λ is too high, which results in slight amplitudes of conductivities. Thereby some redundant links have not been cut, when Eq. (16) is satisfied.

Comparing iterative steps between Fig. 9a, b, the average of iterative steps arises from 3348.7 to 13963, but the minimum of iterative steps just arises from 1669 to 1726. Although the average of iterative steps has a significant rise, the minimum of iterative steps rises slightly. These results suggest the modified *Physarum* model with a better parameters setting has a good ability to adapt to the increment of problem scales.

4.4 Complexity analysis of PMcrossover

Computational complexities of GAs with different coding schemes and genetic operators are hardly described uniformly. Hence, we just focus on the computational complexities of crossover operators. For a comparison, we analyze the crossover operators of EEGA, GAMRA, ISGSA and PMcrossover. All experiments are implemented on Matlab 2012b, with Pentium(R) Dual-Core CPU E5700 and 1.8 GB RAM.

- **PMcrossover:** Let *N*, *E* be the numbers of nodes and edges in a network respectively, and *IT* be the iterative steps of *Physarum* model. First, PMcrossover spends $O(E)$ time units on selecting common links between parent chromosomes. In the iterative stage, $O(N)$ time units are used for calculating the pressure at every node

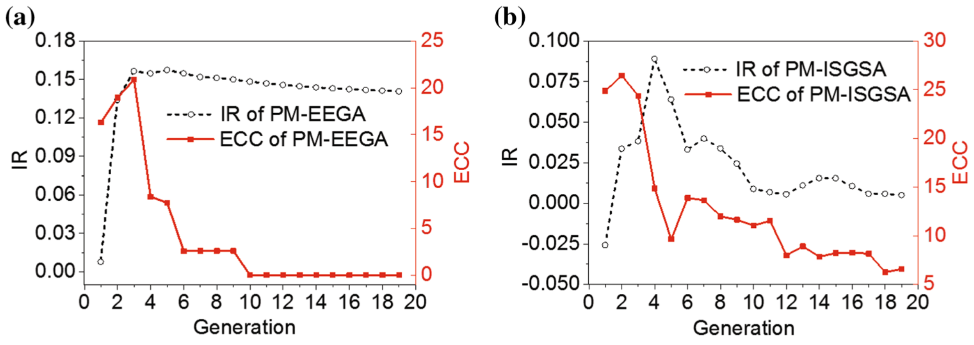


Fig. 10 The dynamic changes of *IR* and *ECC*. **a**, **b** show the *ECC* and *IR* of PM-EEGA and PM-ISGSA on *D*2 respectively. *ECC* values of PM-ISGSA and PM-EEGA both descent with the increment of generations. Comparing with *IR* of PM-EEGA, *IR* of PM-ISGSA declines quickly with the increment of generations. Results suggest the efficiency of PMGAs has the significant improvement for GAs

based on Eq. (10). After that, $O(E)$ time units are spent on calculating the conductivities of links based on Eq. (12). And then PMcrossover goes to the next iteration. Thereby the total computational complexity of PMcrossover is represented in Eq. (17).

$$O(E) + O(IT \times (E + N)) = O(IT \times (E + N)) \tag{17}$$

- **The crossover operator of GAMRA:** Let L be the number of destinations. First, two chromosomes with larger fitness values are picked. After that, $O(1)$ time units are used for selecting star and end points of portion to be exchanged. And then changing the selected portion between two chromosomes spends at most $O(L)$ time units. Thereby the complexity of such crossover operator is measured by Eq. (18).

$$O(1) + O(L) = O(L) \tag{18}$$

- **The crossover operator of ISGSA:** There are three steps in the processing of crossover operator. First, operator reserves the same links of parent chromosomes, which spends $O(E)$ time units. And then, operator adds links randomly from destinations, which are separated by the source node, to the connected component including the source node. This step needs at most $O(E)$ time units. In the third step, excrescent links are deleted, and $O(N^2)$ time units are spent on the this step. Thereby the complexity of such crossover operator is represented by Eq. (19).

$$O(E) + O(E) + O(N^2) = O(E + N^2) \tag{19}$$

- **The crossover operator of EEGA:** The crossover operator of EEGA also spends $O(E)$ time units on selecting the common links between parent chromosomes. We assume that there are S sub-trees with just the common links between parent chromosomes. And then at most $S - 1$ least-delay paths need to be found to connect these sub-trees. In the worst situation, the operator needs $O(N^2)$ time units to find a least-delay path. Thereby the complexity of such crossover operator is represented in Eq. (20).

$$O(E) + O((S - 1) \times (N^2)) \tag{20}$$

Although the crossover operator of GAMRA has the lowest computational complexity, the GAMRA spends a lot of time on constructing the routing table. Therefore, it is less significant to compare GAMRA with others. However, we present the time complexity of these crossover operators. The actual situations are more complex. For further analyzing the computational cost of PMcrossover scheme, we show the extra computational cost (denoted as *ECC*) and the improvement-ratio (denoted as *IR*) of PMGAs in the evolutionary process on *D*2, simultaneously. Based on definitions in Huang and Lee (2007), the IR_i is measured by Eq. (21), where $S^i_{average}$ and $\widehat{S}^i_{average}$ stand for the average cost of results at the *i*th generation of GAs and their updating algorithms respectively. And the ECC_i is defined as Eq. (22), where S^i_{time} and $\widehat{S}^i_{time}$ represent for computational cost which GAs and PMGAs spend on the *i*th generation respectively. And the computational cost is defined as the time spending on computation in scends.

$$IR_i = \frac{(S^i_{average} - \widehat{S}^i_{average})}{S^i_{average}} \tag{21}$$

$$ECC_i = \widehat{S}^i_{time} - S^i_{time} \tag{22}$$

Figure 10 shows the *IR* and *ECC* of PM-EEGA and PM-ISGSA on *D*2, respectively. *ECC* values of PM-ISGSA and PM-EEGA both are higher at the initial stage and then show a downtrend. Meanwhile, *IR* values of PM-ISGSA and PM-EEGA also have a rise at the initial stage. The main difference between PM-ISGSA and PM-EEGA is the decline rate of *IR* with the increment of generations. In detail, *IR* of PM-EEGA descends slightly while *IR* of PM-ISGSA descends rapidly. At the final stage, *IR* of PM-EEGA is still higher while *IR* of PM-ISGSA is not so remarkable, compared with the initial stage. Because PMGAs can find the optimal solution with a fewer generations, their computational costs can be reduced by setting a lower maximal generation. Based on above observations, we can conclude that PMcrossover has the significant improvement for GAs.

5 Conclusion

Inspired by *Physarum* polycephalum forming an optimized network in foraging food sources, a genetic algorithm based on the modified *Physarum* network was proposed to solve the multicast routing NP-complete problem. In the proposed PMGA, a new crossover operator was presented, which can improve the effectiveness of GA. Some experiments on four datasets were used to verify the efficiency of proposed method. Moreover, for further improving the efficiency of PMGA, the original *Physarum* network model is modified by simplifying the calculation of pressures in *Physarum* networks. Furthermore, parameters analyses indicate that the modified *Physarum* model with a better parameters setting has a better scalability.

Acknowledgments Prof. Z.L. Zhang and Dr. Chao Gao are the corresponding authors. This work is supported by the National High Technology Research and Development Program of China (No. 2013AA013801), National Natural Science Foundation of China (Nos. 61402379, 61403315), and Natural Science Foundation of Chongqing (No. cstc2013jcyjA40022), Fundamental Research Funds for the Central Universities (No. XDJK2016A008) and Research Fund for the Doctoral Program of Higher Education (RFDP) (No. 20120182120016).

Appendix

$D1 = \{$(number of endpoint, number of endpoint, cost, delay)| (8, 1, 0.682, 0.853) (10, 1, 0.975, 0.547) (15, 1, 0.860, 0.0305) (17, 1, 0.595, 0.173) (7, 2, 0.820, 0.961) (15, 2, 0.367, 0.744) (4, 3, 0.856, 0.505) (13, 3, 0.758, 0.801) (14, 3, 0.413, 0.913) (16, 3, 0.648, 0.042) (20, 3, 0.456, 0.617) (5, 4, 0.923, 0.549) (9, 4, 0.404, 0.403) (10, 4, 0.480, 0.188) (19, 5, 0.308, 0.190) (7, 6, 0.484, 0.868) (8, 6, 0.757, 0.075) (16, 6, 0.878, 0.096) (8, 7, 0.579, 0.239) (19, 7, 0.684, 0.460) (10, 8, 0.467, 0.625) (12, 8, 0.334, 0.111) (11, 9, 0.464, 0.622) (13, 9, 0.874, 0.237) (14, 9, 0.310, 0.136) (20, 9, 0.615, 0.584) (11, 10, 0.950, 0.587) (12, 10, 0.843, 0.408) (13, 10, 0.640, 0.458) (17, 10, 0.655, 0.627) (20, 11, 0.604, 0.906) (13, 12, 0.721, 0.546) (14, 12, 0.797, 0.653) (19, 12, 0.596, 0.376) (14, 13, 0.462, 0.494) (15, 14, 0.800, 0.028) (18, 15, 0.798, 0.698) (19, 18, 0.999, 0.226)$\}$, $\Delta_d = 5$.

$D2 = \{$(number of endpoint, number of endpoint, cost, delay)| (2, 1, 11, 4) (3, 1, 7, 6) (8, 1, 21, 3) (4, 2, 4, 4) (5, 2, 2, 5) (11, 2, 5, 3) (4, 3, 24, 3) (6, 4, 7, 11) (6, 5, 39, 1) (12, 5, 3, 5) (7, 6, 22, 3) (14, 6, 35, 4) (12, 7, 15, 4) (13, 7, 2, 3) (9, 8, 21, 3) (10, 8, 14, 6) (10, 9, 10, 7) (12, 9, 13, 1) (12, 10, 20, 3) (12, 11, 21, 3) (15, 13, 5, 3) (15, 14, 6, 4)$\}$, $\Delta_d = 18$.

$D3 = \{$(number of endpoint, number of endpoint, cost, delay)| (1, 6, 8.197, 0.307) (1, 7, 7.549, 0.367) (1, 11, 4.214, 0.642) (1, 12, 9.743, 0.518) (1, 15, 7.722, 0.655) (2, 5, 5.654, 0.312) (2, 6, 7.986, 0.451) (2, 9, 3.002, 0.3662) (2, 11, 3.935, 0.490) (2, 13, 3.155, 0.583) (3, 4, 9.337, 0.539) (3, 7, 6.643, 0.421) (3, 8, 4.057, 0.360) (3, 10, 5.365, 0.441) (4, 8, 7.970, 0.647) (4, 10, 3.572, 0.413) (4, 14, 8.306, 0.398) (5, 9, 9.688, 0.374) (5, 11, 4.029, 0.610) (5, 13, 8.719, 0.387) (6, 7, 7.679, 0.562) (6, 9, 9.061, 0.490) (6, 11, 6.316, 0.346) (6, 12, 6.882, 0.522) (6, 15, 7.190, 0.502) (6, 16, 3.485, 0.612) (7, 8, 6.436, 0.352) (7, 10, 7.347, 0.631) (7, 12, 7.551, 0.329) (7, 15, 9.843, 0.601) (8, 10, 8.144, 0.291) (8, 12, 6.567, 0.479) (8, 14, 5.807, 0.622) (8, 19, 8.837, 0.631) (8, 22, 8.636, 0.660) (9, 11, 7.803, 0.252) (9, 13, 5.529, 0.315) (9, 15, 5.862, 0.651) (9, 16, 6.463, 0.435) (9, 18, 5.801, 0.461) (9, 20, 6.337, 0.622) (9, 21, 5.509, 0.645) (10, 14, 7.929, 0.336) (10, 17, 9.245, 0.455) (11, 12, 6.445, 0.608) (11, 13, 9.912, 0.531) (11, 15, 4.762, 0.403) (11, 16, 4.007, 0.295) (11, 18, 9.952, 0.503) (11, 21, 5.329, 0.470) (11, 23, 9.146, 0.622) (12, 15, 4.351, 0.295) (12, 19, 9.411, 0.373) (12, 21, 4.566, 0.616) (13, 16, 9.953, 0.572) (13, 18, 6.545, 0.379) (13, 20, 3.090, 0.387) (13, 23, 6.023, 0.638) (14, 17, 9.680, 0.303) (14, 25, 3.959, 0.594) (15, 16, 5.267, 0.371) (15, 19, 9.173, 0.385) (15, 21, 7.194, 0.329) (15, 27, 5.202, 0.584) (15, 29, 7.961, 0.634) (16, 18, 3.80, 0.332) (16, 20, 8.485, 0.606) (16, 21, 4.405, 0.211) (16, 23, 8.201, 0.348) (16, 27, 4.002, 0.470) (16, 30, 7.739, 0.582) (17, 22, 9.816, 0.523) (17, 24, 8.302, 0.398) (17, 25, 6.715, 0.334) (17, 32, 4.940, 0.590) (18, 20, 7.453, 0.274) (18, 21, 8.467, 0.471) (18, 23, 9.232, 0.259) (18, 27, 7.068, 0.589) (18, 28, 8.130, 0.441) (18, 30, 8.697, 0.500) (19, 21, 7.585, 0.531) (19, 22, 7.701, 0.455) (19, 24, 3.276, 0.640) (19, 26, 8.093, 0.360) (19, 27, 3.893, 0.622) (19, 29, 3.688, 0.431) (19, 31, 7.232, 0.494) (20, 23, 4.357, 0.441) (20, 28, 7.091, 0.345) (20, 30, 9.546, 0.596) (21, 23, 3.220, 0.352) (21, 27, 5.448, 0.293) (21, 29, 5.166, 0.515) (21, 30, 6.833, 0.507) (21, 33, 8.709, 0.553) (22, 24, 5.198, 0.185) (22, 25, 7.572, 0.602) (22, 26, 6.844, 0.219) (22, 29, 9.269, 0.629) (22, 31, 6.252, 0.445) (22, 32, 3.272, 0.392) (23, 27, 9.829, 0.352) (23, 28, 6.379, 0.385) (23, 30, 5.975, 0.259) (23, 33, 6.377, 0.500) (23, 35, 7.838, 0.663) (23, 36, 7.603, 0.600) (23, 37, 9.731, 0.613) (24, 25, 3.208, 0.417) (24, 26, 6.701, 0.376) (24, 31, 7.755, 0.564) (24, 32, 9.647, 0.274) (24, 38, 8.590, 0.638) (25, 32, 5.922, 0.428) (25, 34, 7.025, 0.442) (26, 29, 7.727, 0.413) (26, 31, 5.885, 0.241) (26, 32, 3.088, 0.473) (26, 40, 9.290, 0.615) (27, 28, 6.010, 0.661) (27, 29, 9.107, 0.370) (27, 30, 3.222, 0.324) (27, 31, 4.900, 0.651) (27, 33, 9.154, 0.261) (27, 36, 3.054, 0.496) (27, 37, 8.970, 0.656) (27, 39, 5.462, 0.615) (28, 30, 9.940, 0.363) (28, 35, 5.724, 0.349) (28, 36, 6.793, 0.601) (28, 37, 7.897, 0.445) (28, 41, 8.896, 0.631) (29, 31, 3.070, 0.281) (29, 33, 8.962, 0.434) (29, 36, 8.669, 0.692) (29, 39, 6.361, 0.491) (29, 40, 6.408, 0.590) (30, 33, 3.811, 0.313) (30, 35, 3.595, 0.498) (30, 36, 9.168, 0.343) (30, 37, 6.422, 0.376) (30, 41, 8.374, 0.637) (30, 42, 5.428, 0.648) (30, 44, 4.848, 0.694) (31, 32, 8.768, 0.551) (31, 33, 5.957, 0.667) (31, 38, 6.297, 0.617) (31, 39, 3.348, 0.537) (31, 40, 6.643, 0.410) (32, 34, 7.376, 0.464) (32, 38, 6.803, 0.374) (32, 40, 4.431, 0.637) (32, 43, 5.908, 0.668) (33, 36, 5.626, 0.266) (33, 37, 7.402, 0.491) (33, 39, 7.982, 0.417) (33, 42, 8.390, 0.450) (33, 44, 8.534, 0.627) (34, 38, 3.360, 0.499) (34, 43, 4.324, 0.456) (35, 36, 5.521, 0.510) (35, 37, 4.452, 0.251) (35, 41, 9.006, 0.305) (35, 44, 9.669, 0.579) (35, 46, 8.093, 0.655) (36, 37, 7.797, 0.263) (36, 39, 5.230, 0.500) (36, 41, 7.476, 0.458) (36, 42, 3.0563, 0.313) (36, 44, 8.227, 0.379) (36, 49, 4.107, 0.668) (37, 41, 7.264, 0.260) (37, 42, 3.519, 0.507) (37, 44, 6.914, 0.391) (37, 46, 3.308, 0.631) (38, 40, 6.990, 0.435) (38, 43, 5.081, 0.386) (38, 47, 6.347, 0.561) (38, 48, 7.744, 0.681) (39, 40, 4.309, 0.419) (39, 42, 5.630, 0.355) (39, 44, 3.077, 0.654) (39, 45, 7.530, 0.497) (39, 50, 8.132, 0.601) (40, 45, 9.557, 0.526) (40, 47, 9.185, 0.574) (41, 42, 6.132, 0.572) (41, 44, 3.230, 0.319) (41, 46, 6.869, 0.378) (41, 49, 4.788, 0.511) (42, 44, 5.267, 0.300) (42, 45, 6.418, 0.575) (42, 49, 4.340, 0.493) (42, 50, 9.512, 0.429) (43, 47, 3.246, 0.528) (43, 48, 8.404, 0.316) (44, 46, 7.936, 0.438) (44, 49, 8.298, 0.294) (44, 50, 6.236, 0.513) (45, 47, 3.476, 0.530) (45, 50, 4.900, 0.372) (46, 49, 7.593, 0.364) (47, 48, 7.700, 0.553) (49, 50, 8.928, 0.423)}, $\Delta_d = 5$.

$D4 =$ {(number of endpoint, number of endpoint, cost, delay, jitter, bandwidth)| (1, 2, 229.2, 123.27, 1.751, 797.457) (1, 5, 277.884, 123.312, 1.7516, 797.403) (1, 16, 153.24, 123.205, 1.75, 797.539) (1, 17, 132.163, 123.187, 1.7497, 797.562) (1, 18, 53.8145, 123.12, 1.7487, 797.648) (1, 19, 168.808, 123.219, 1.7502, 797.522) (2, 3, 215.969, 123.259, 1.7508, 797.471) (2, 4, 267.247, 123.303, 1.7515, 797.415) (2, 5, 63.7104, 123.129, 1.7488, 797.637) (2, 6, 176.894, 123.226, 1.7503, 797.513) (2, 8, 152.6, 123.205, 1.75, 797.54) (2, 9, 239.438, 123.279, 1.7511, 797.446) (2, 11, 202.107, 123.247, 1.7506, 797.486) (2, 12, 266.533, 123.302, 1.7515, 797.416) (2, 16, 126.895, 123.183, 1.7496, 797.568) (2, 18, 197.297, 123.243, 1.7506, 797.491) (3, 5, 260.038, 123.297, 1.7514, 797.423) (3, 6, 63.0259, 123.128, 1.7488, 797.638) (3, 8, 248.496, 123.287, 1.7513, 797.435) (3, 9, 80, 123.143, 1.749, 797.62) (4, 5, 221.549, 123.264, 1.7509, 797.465) (4, 8, 294.157, 123.326, 1.7519, 797.386) (4, 11, 244.492, 123.283, 1.7512, 797.44) (4, 12, 82.8908, 123.145, 1.7491, 797.616) (4, 13, 52.5624, 123.119, 1.7487, 797.65) (4, 16, 224.185, 123.266, 1.7509, 797.462) (5, 6, 210.481, 123.254, 1.7508, 797.477) (5, 8, 116.669, 123.174, 1.7495, 797.579) (5, 9, 266.66, 123.302, 1.7515, 797.416) (5, 11, 144.867, 123.198, 1.7499, 797.549) (5, 12, 208.471, 123.253, 1.7507, 797.479) (5, 13, 255.831, 123.293, 1.7514, 797.427) (5, 16, 147.683, 123.2, 1.7499, 797.545) (5, 18, 238.255, 123.278, 1.7511, 797.447) (6, 8, 186.141, 123.233, 1.7504, 797.504) (6, 9, 64.9019, 123.13, 1.7488, 797.636) (6, 11, 272.403, 123.307, 1.7516, 797.409) (6, 16, 294.214, 123.326, 1.7519, 797.386) (7, 8, 239.9, 123.279, 1.7511, 797.445) (7, 10, 84.9586, 123.147, 1.7491, 797.614) (7, 11, 221.828, 123.264, 1.7509, 797.464) (8, 9, 219.14, 123.262, 1.7509, 797.468) (8, 10, 280.165, 123.314, 1.7517, 797.401) (8, 11, 86.539, 123.148, 1.7491, 797.612) (8, 12, 248.458, 123.287, 1.7513, 797.435) (8, 16, 263.536, 123.3, 1.7515, 797.419) (10, 11, 235.5, 123.276, 1.7511, 797.449) (11, 12, 182.129, 123.23, 1.7504, 797.508) (11, 13, 250.027, 123.288, 1.7513, 797.434) (11, 16, 284.1, 123.317, 1.7517, 797.397) (12, 13, 68.656, 123.133, 1.7489, 797.632) (12, 16, 262.887, 123.299, 1.7515, 797.419) (13, 14, 264.3, 123.3, 1.7515, 797.418) (13, 16, 275.18, 123.31, 1.7516, 797.406) (14, 15, 199, 123.244, 1.7506, 797.489) (16, 17, 235.9, 123.276, 1.7511, 797.449) (16, 18, 102.801, 123.162, 1.7493, 797.595) (16, 19, 297.402, 123.329, 1.7519, 797.382) (17, 18, 146.756, 123.2, 1.7499, 797.547) (17, 19, 71.1157, 123.135, 1.7489, 797.629) (17, 20, 261.873, 123.298, 1.7514, 797.421) (18, 19, 200, 123.245, 1.7506, 797.489) (19, 20, 200, 123.245, 1.7506, 797.489)}, $\Delta_d = 1200, \Delta_b = 797.4, \Delta_j = 17$.

References

Abbass HA (2001) A single queen single worker honey-bees approach to 3-SAT. In: The genetic and evolutionary computation conference, pp 807–814

Adamatzky A (2009) Developing proximity graphs by *Physarum* polycephalum: does the plasmodium follow the Toussaint hierarchy? Parallel Process Lett 19(1):105–127

Chow CH (1991) On multicast path finding algorithms. In: 10th Annual joint conference of the IEEE computer and communications societies, pp 1274–1283

Huang TL, Lee DT (2007) A distributed multicast routing algorithm for real-time applications in wide area networks. J Parallel Distrib Comput 67(5):516–530

Hwang RH, Do WY, Yang SC (2000) Multicast routing based on genetic algorithms. J Inf Sci Eng 16(6):885–901

Karthikeyan P, Baskar S (2015) Genetic algorithm with ensemble of immigrant strategies for multicast routing in ad hoc networks. Soft Comput 19(2):489–498

Liu YX, Gao C, Wu YH, Tao L, Lu YX, Zhang ZL (2014) A *Physarum*-inspired multi-agent system to solve maze. In: The fifth international conference on Swarm intelligence, pp 424–430

Liu YX, Zhang ZL, Gao C, Wu YH, Qian T (2013) A *Physarum* network evolution model based on IBTM. In: The fourth international conference on Swarm intelligence, pp 19–26

Liu L, Song Y, Ma H, Zhang X (2015) Physarum optimization: a biology-inspired algorithm for minimal exposure path problem in wireless sensor networks. IEEE Trans Comput 64(3):819–832

Lu T, Zhu J (2013) Genetic algorithm for energy-efficient QoS multicast routing. Commun Lett 17(1):31–34

Mahmoud TM, El Nashar AI, Eman M (2014) An efficient genetic algorithm based clonal selection and hill climbing for solving QoS multicast routing problem. Int J Comput Sci Issues. 11(3):83–88

Nakagaki T, Yamada H, Tóth Á (2000) Intelligence: Maze-solving by an amoeboid organism. Nature 407(6803):470–470

Oliveira CA, Pardalos PM (2005) A survey of combinatorial optimization problems in multicast routing. Comput Oper Res 32(8):1953–1981

Peng B, Li L (2013) Combination of genetic algorithm and ant colony optimization for QoS multicast routing. In: 14th International symposium on advanced intelligent systems, pp 49–56

Pham DT, Kog E, Ghanbarzadeh A, Otri S, Rahim S, Zaidi M (2006) The bees algorithm—a novel tool for complex optimisation problems. In: The 2nd international virtual conference on intelligent production machines and systems, p 454

Ratnasamy S, Ermolinskiy A, Shenker S (2006) Revisiting IP multicast. ACM SIGCOMM Comput Commun Rev 36(4):15–16

Salama HF (1996) Multicast routing for real-time communication of high-speed networks. Ph.D. Thesis. North Carolina State University

Salama HF, Reeves DS, Viniotis Y (1997) Evaluation of multicast routing algorithms for real-time communication on high-speed networks. IEEE J Sel Areas Commun 15(3):32–45

Salim B, Abdelhamid M (2013) Bee life-based multi constraints multicast routing optimization for vehicular ad hoc networks. J Netw Comput Appl 36(3):981–991

Sesay S, Yang ZK, He JH (2004) A survey on mobile ad hoc wireless network. Inf Technol J 3(2):168–175

Tero A, Kobayashi R, Nakagaki T (2007) A mathematical model for adaptive transport network in path finding by true slime mold. J Theor Biol 244(4):553–564

Tero A, Takagi S, Saigusa T, Ito K, Bebber DP, Fricker MD, Yumiki K, Kobayashi R, Nakagaki T (2010) Rules for biologically inspired adaptive network design. Sci Signal 327(5964):439

Wang ZY, Shi BX, Zhao E (2001) Bandwidth-delay-constrained least-cost multicast routing based on heuristic genetic algorithm. Comput Commun 24(7):685–692

Wang Z, Szolnoki A, Perc M (2012) If players are sparse social dilemmas are too: importance of percolation for evolution of cooperation. Sci Rep 2:369

Wang Z, Crowcroft J (1996) Quality-of-service routing for supporting multimedia applications. IEEE J Sel Areas Commun 14(7):1228–1234

Watanabe S, Tero A, Takamatsu A, Nakagaki T (2011) Traffic optimization in railroad networks using an algorithm mimicking an amoeba-like organism. *Physarum* plasmodium. Biosystems 105(3):225–232

Yen Y, Chao H, Chang R, Vasilakos A (2011) Flooding-limited and multi-constrained QoS multicast routing based on the genetic algroithms. Math Comput Model 53(11):2238–2250

Yin P, Chang R, Chao C, Chu Y (2014) Niched ant colony optimization with colony guides for QoS multicast routing. J Netw Comput Appl 40:61–72

Yu Z, Wong H, Wang D, Wei M (2011) Neighborhood knowledge-based evolutionary algorithm for multiobjective optimization problems. IEEE Trans Evol Comput 15(6):812–831

Yu Z, Chen H, You J, Wong H, Liu J, Li L, Han G (2014) Double selection based semi-supervised clustering ensemble for tumor clustering from gene expression profiles. IEEE/ACM Trans Comput Biol Bioinfor 11(4):727–740

Yu Z, Chen H, You J, Wong H, Liu J, Han G, Li L (2015) Adaptive fuzzy consensus clustering framework for clustering analysis of cancer data. IEEE/ACM Trans Comput Biol Bioinform 12(3):568–582

Zhang L, Cai L, Li M, Wang F (2009) A method for least-cost QoS multicast routing based on genetic simulated annealing algorithm. Comput Commun 32(1):105–110

Zhang ZL, Gao C, Liu YX, Qian T (2014) A universal optimization strategy for ant colony optimization algorithms based on the *Physarum*-inspired mathematical model. Bioinspir Biomim 9(3):036006