

## Chapter 2

# Network Community Discovery with Evolutionary Single-Objective Optimization

**Abstract** Network community detection is one of the most fundamental problems in network structure analytics. With the modularity and modularity density being put forward, network community detection is formulated as a single-objective optimization problem and then communities of network can be discovered by optimizing modularity or modularity density. However, the community detection by optimizing modularity or modularity density is NP-hard. The computational intelligence algorithm, especially for evolutionary single-objective algorithms, have been effectively applied to discover communities from networks. This chapter focuses on evolutionary single-objective algorithms for solving network community discovery. First this chapter reviews evolutionary single-objective algorithm for network community discovery. Then three representative algorithms and their performances of discovering communities are introduced in detail.

## 2.1 Review of the State of the Art

Complex systems can be modeled by complex networks. Entities of complex systems can be represented by nodes of networks and their relationships are represented by edges of networks. It is found that community structure is an important property of networks. As shown in Chap. 1, a community is a subset of nodes which have dense connections within them and sparse connections between them and other nodes in the network. Community detection is to divide a network into several clusters. Criteria are required to evaluate the partitions of network.

One of the most famous and used criteria is modularity, which is originally introduced by Newman and Girvan [19]. High values of modularity are associated with subjectively good partitions. As a consequence, the partition that has the maximum modularity is expected to be the subjectively best partition, or at least a very good one. However, Fortunato and Barthélemy [9] found that modularity may have the resolution limit and fail to identify modules smaller than a scale. To alleviate the resolution limit, Li et al. [17] introduced a quality function, called modularity density. The authors demonstrated that this quantitative function is superior to the widely used

modularity. A tunable parameter is introduced into a general version of modularity density which can explore the network at different resolutions.

With the introductions of modularity and modularity density, network community discovery can be formulated as a single-objective optimization problem. Several studies have shown the potential of nature-inspired optimization algorithms for community discovery by optimizing modularity or modularity density [10, 12, 13, 25]. For instance, in [13], Gong et al. proposed a memetic algorithm, named as Meme-Net, to uncover communities at different hierarchical levels. Meme-Net shows its effectiveness. However, its high computational complexity makes it impossible to search communities on slightly large networks. Shang et al. [25] and Gong et al. [12] try to adopt simulated annealing as an individual learning procedure to decrease the computational complexity of Meme-Net. However, their computational complexities are still very high relative to classical modularity-based community detection algorithms. The algorithm in [10] adopted the technique in [3] as the local search. Meanwhile, it takes a large amount of time and energy on generating initial population as it directly uses the algorithm in [3] to initialize a population of solutions. Therefore, the algorithms in [10, 12, 13, 25] are difficult to apply to real-world problems or large-scale networks.

To discover communities in large-scale networks, different strategies are adopted, for example, multi-level local learning strategies and greedy local search [5, 18]. Ma et al. [18] proposed a fast memetic algorithm with multi-level learning strategies for community detection by optimizing modularity. The multi-level learning strategies are devised based on the potential knowledge of the node, community and partition structures of networks, and they work on the network at nodes, communities and network partitions levels, respectively. Cai et al. [5] designed a novel discrete particle swarm optimization algorithm with a greedy local search for community discovery.

This chapter is organized into four sections. Section 2.2 introduces a basic memetic algorithm for community discovery. A memetic algorithm with multi-learning strategies is presented in Sect. 2.3. A greedy discrete particle swarm optimization algorithm is introduced in Sect. 2.4.

## 2.2 A Node Learning-Based Memetic Algorithm for Community Discovery in Small-Scale Networks

As shown in Sect. 2.1, the discovery of community structure is formulated as a single optimization problem. Discovering community can be solved by maximizing modularity or modularity density. This section is based on maximizing modularity density.

Memetic algorithms (MAs) are hybrid global–local heuristic search methodologies [20]. The global heuristic search is usually a form of population-based method, while the local search is generally considered as an individual learning procedure for accelerating the convergence to an optimum [20]. In general, the population-

based global search has the advantage of exploring the promising search space and providing a reliable estimate of the global optimum [4]. However, the population-based global search is difficult to discover an optimal solution around the explored search space in a short time. The local search is usually designed for accelerating the search and finding the best solutions on the explored search space. Therefore, this hybridization, which synthesizes the complementary advantages of the population-based global search and the individual-based local search, can effectively produce better solutions. MAs have been proved to be of practical success in NP-complete problem.

This section introduces a basic memetic algorithm for community discovery, named Meme-Net. The advantages of Meme-Net are as follows. Meme-Net combines GAs and a hill-climbing strategy as the local search procedure. The hill-climbing strategy is designed based on node learning. The algorithm does not require the number of communities present in the graph in advance. In addition, by tuning the parameter in the quality function, it is able to explore the network at different resolutions and may reveal the hierarchical structure of the network.

### 2.2.1 Memetic Algorithm with Node Learning for Community Discovery

Here, the framework of Meme-Net is given as Algorithm 1.

---

#### Algorithm 1 The algorithm framework of Meme-Net

---

**Input:** Maximum number of generations:  $G_{max}$ ; Population size:  $S_{pop}$ ; Size of mating pool:  $S_{pool}$ ; Tournament size:  $S_{tour}$ ; Crossover probability:  $P_c$ ; Mutation probability:  $P_m$ .

**Output:** Convert the fittest chromosome in  $\mathbf{P}$  into a partition solution and output.

```

P ← GenerateInitialPopulation( $S_{pop}$ );
repeat
  Pparent ← Selection(P,  $S_{pool}$ ,  $S_{tour}$ );
  Pchild ← GeneticOperation(Pparent,  $P_c$ ,  $P_m$ );
  Pnew ← LocalSearch(Pchild);
  P ← UpdatePopulation(P, Pnew);
until TerminationCriterion( $G_{max}$ )

```

---

In this framework, the GenerateInitialPopulation() procedure is used to create the initial population. The Selection() function is used to select parental population for mating. Here, the deterministic tournament selection is used. The GeneticOperation() function is used to perform crossover and mutation operation. The UpdatePopulation() procedure is used to reconstruct the current population using the population  $\mathbf{P}$  and  $\mathbf{P}_{new}$ . Here, the current population is constructed taken the best  $S_{pop}$  individuals from  $\mathbf{P} \cup \mathbf{P}_{new}$ . The TerminationCriterion() function is the terminate condition.

In what follows, more detailed descriptions about the initialization procedure, genetic operators and the local search procedure will be given.

### 2.2.2 Problem Formation

Li et al. [17] designed a function, named modularity density (Eq. 1.7). The authors also proved the equivalence of modularity density and kernel  $k$  means, and proposed a more general modularity density measure:

$$D_\lambda = \sum_{i=1}^m \frac{2\lambda L(V_i, V_i) - 2(1 - \lambda)L(V_i, \bar{V}_i)}{|V_i|}. \quad (2.1)$$

When  $\lambda = 1$ ,  $D_\lambda$  is equivalent to the ratio association; when  $\lambda = 0$ ,  $D_\lambda$  is equivalent to the ratio cut; when  $\lambda = 0.5$ ,  $D_\lambda$  is equivalent to the modularity density  $D$ . So the general modularity density  $D_\lambda$  can be viewed as a combination of the ratio association and the ratio cut. Generally, optimization of the ratio association algorithm often divides a network into small communities, while optimization of the ratio cut often divides a network into large communities. This general modularity density  $D_\lambda$ , which is a convex combination of these two indexes, can avoid the resolution limits. In other words, by varying the  $\lambda$  value, we can use this general function to analyze the topological structure and uncover more detailed and hierarchical organization of the complex network.

### 2.2.3 Representation and Initialization

Each chromosome is encoded as an integer string

$$x = \{x^1 x^2 \cdots x^n\}. \quad (2.2)$$

where,  $n$  is the number of the vertices in the graph, and  $x^i$  is the integer cluster identifier of vertex  $v_i$ , which can be any integer number between 1 and  $n$ . The vertices having the same cluster identifier are partitioned into the same community. This representation does not need the number of clusters of in the graph. A graph of  $n$  vertices can be partitioned into  $n$  clusters at most, and in this case, each cluster contains only one vertex, which can be denoted as  $\{1 \ 2 \cdots n\}$ . However, it should be noted that there are many different representations corresponding to the same partition. For instance, given a graph of 4 vertices,  $\{3 \ 1 \ 2 \ 3\}$  and  $\{1 \ 2 \ 3 \ 1\}$  represent the same partition  $\{\{1, 4\}, \{2\}, \{3\}\}$  of the graph.

The population initialization procedure is given as Algorithm 2. Initially each vertex is put in a different cluster for all chromosomes, that is, each chromosome in the population is  $\{1 \ 2 \cdots n\}$ . However, this initialised population lacks of diversity and each solution is of low quality. To speed up the convergence, here a simple heuristic algorithm is employed. For each chromosome, a vertex is selected randomly and assign its cluster identifier to all of its neighbors. This operation is repeated  $\alpha \cdot n$  times for each chromosome in the initial population where  $\alpha$  is a parameter and  $\alpha = 0.2$  is used. This operation is very fast and results in local small communities, but the resulting clusterings are still far away from being optimal.

**Algorithm 2** The Population Initialization Procedure**Input:** Population size:  $S_{pop}$ .**Output:** Population  $\mathbf{P}$ .

---

```

1: Generate a population  $\mathbf{P}$ , each chromosome  $\mathbf{x}_k$  of which is set to  $\{1 \ 2 \cdots n\}$ , where  $k = 1, 2, \dots, S_{pop}$ .
2: for each chromosome  $\mathbf{x}_k$  do
3:    $t_{counter} \leftarrow 0$ ;
4:   repeat
5:     randomly select a vertex  $v_i$ ;
6:      $x_k^j \leftarrow x_k^i$  whenever  $(v_i, v_j \in E)$ ;
7:      $t_{counter} \leftarrow t_{counter} + 1$ ;
8:   until  $t_{counter} = \alpha \cdot n$ 
9: end for

```

---

**2.2.4 Genetic Operators**

The genetic operators, including crossover and mutation, are important to change the genetic composing of the chosen solutions. The crossover operator is to generate offspring chromosomes which can inherit their parent chromosomes' communities, and the mutation operator is to generate spontaneous random changes in chromosomes. The crossover and mutation operators can effectively prevent the proposed algorithm from getting into a local optimal network partition.

**Crossover.** In Meme-Net, a two-way crossover operation is employed. The crossover procedure is defined as follows. The two selected chromosomes are called  $x_a$  and  $x_b$ , respectively. We pick a vertex  $v_i$  at random, determine its cluster (i.e.,  $x_a^i$ ) in the chromosome  $x_a$  and make sure that all the vertices in this cluster of  $x_a$  are also assigned to the same cluster in the chromosome  $x_b$  (i.e.  $x_b^k \leftarrow x_a^i, \forall k \in \{k \mid x_a^k = x_a^i\}$ ). Simultaneously, we also determine the cluster of the vertex  $v_i$  in  $x_b$ , and make sure that all the vertices in this cluster of  $x_b$  are also assigned to the same cluster in  $x_a$  (i.e.,  $x_a^k \leftarrow x_b^i, \forall k \in \{k \mid x_b^k = x_b^i\}$ ). This procedure returns two new chromosomes  $x_c$  and  $x_d$ . An example of two-way crossover is given in Table 2.1. This two-way crossing over operation can generate descendants carrying features common to the parents, which represents the exploitative side of the crossover operator; on the other hand, the crossover operation is exploratory, which means it can generate descendants carrying combinations of features taken from the parents. For instance, as shown in one of the descendants  $\mathbf{x}_c$  in Table 2.1,  $v_4$  becomes in the same community with  $v_3$ . These properties make the two-way crossover operation suitable for our algorithm.

**Mutation.** In Meme-Net, one-point mutation is employed on this chromosome: a vertex is picked randomly on the chromosome, then the cluster of the vertex is randomly changed to the cluster of one of its neighbors. This operation is repeated  $n$  times on the chromosome. The specialized mutation operator that only considers the neighbors of the vertex can decrease useless exploration and reduce the search space.

**Table 2.1** Two-way crossover when  $v_3$  is selected

$v$		$\mathbf{x}_a$		$\mathbf{x}_b$		$\mathbf{x}_c$	$\mathbf{x}_d$		$\mathbf{x}_a$		$\mathbf{x}_b$		$v$
1		⑤	→	2	→	⑤	5		5		2		1
2		3		6		6	⑥	←	3	←	⑥		2
③	→	⑤	→	6	→	⑤	⑥	←	5	←	⑥	←	③
4		7		5		5	7		7		5		4
5		2		6		6	⑥	←	2	←	⑥		5
6		⑤	→	3	→	⑤	5		5		3		6
7		3		2		2	3		3		2		7

### 2.2.5 The Local Search Procedure

First the neighbors of a partition are defined. Given a partition  $\Omega = \{V_1, V_2, \dots, V_m\}$  ( $2 \leq m \leq n$ ) of a graph  $G$ , where  $m$  is the number of clusters of this partition and  $n$  is the number of vertices in the graph, a single vertex, chose from a cluster  $V_i (i \in 1, \dots, m)$ , is reassigned into another cluster  $V_j (j \in 1, \dots, m \text{ and } j \neq i)$ . The new partition  $\Omega_{nbr}$  after this reassignment is called a neighbor of the partition  $\Omega$ . In particular, when  $m = 1$ ,  $\Omega = \{V_1\}$ , a neighbor of  $\Omega$  is defined as  $\Omega_{nbr} = \{V_1 - v, \{v\}\}$ , where  $v$  is a vertex from  $V_1$ , and  $\{v\}$  is a cluster including the single vertex  $v$  (a single-vertex cluster).

Then  $N_\Omega$ , the number of all possible neighbors of the partition  $\Omega$  can be figured out. If  $m = 1$ ,  $N_\Omega = n$ ; if  $2 \leq m \leq n$ ,  $N_\Omega = n(m - 1) - \frac{1}{2}s(s - 1)$ , where  $s$  is the number of single-vertex clusters in this partition. For example, when  $m = n$ , the value of  $s$  must be  $n$ , so  $N_\Omega = n(n - 1) - \frac{1}{2}n(n - 1) = \frac{1}{2}n(n - 1)$ .

---

#### Algorithm 3 The Local Search Procedure

---

**Input:**  $\mathbf{P}_{child}$ .

**Output:**  $\mathbf{P}_{child}$ .

```

1:  $\mathbf{n}_{current} \leftarrow \text{FindBest}(\mathbf{P}_{child})$ ;
2:  $y_{islocal} \leftarrow \text{FALSE}$ ;
3: repeat
4:    $\mathbf{L} \leftarrow \text{FindNeighbors}(\mathbf{n}_{current})$ ;
5:    $\mathbf{n}_{next} \leftarrow \text{FindBest}(\mathbf{L})$ ;
6:   if  $\text{Eval}(\mathbf{n}_{next}) > \text{Eval}(\mathbf{n}_{current})$  then
7:      $\mathbf{n}_{current} \leftarrow \mathbf{n}_{next}$ ;
8:   else
9:      $y_{islocal} \leftarrow \text{TRUE}$ ;
10:  end if
11: until  $y_{islocal}$  is TRUE

```

---

The local search procedure used in Meme-Net is a hill-climbing strategy, and the implementation is given as Algorithm 3. In Algorithm 3, the FindBest() function is responsible for evaluating the fitness of each chromosome in the input population,

and return the chromosome having maximum fitness, on which the local search procedure will be performed. The Eval() function is used to evaluate the fitness of a solution. The FindNeighbors() function is responsible for finding the neighbors of a partition, which can be done easily according to the definition of the neighbors of a partition.

In Meme-Net, the local search is applied on  $\mathbf{P}_{child}$ , which is the population after crossover and mutation. Not all of the chromosomes in this population are undergoing the local search procedure. Only the fittest chromosome in  $\mathbf{P}_{child}$  is found and performed local search on it, until no improvement can be made.

### 2.2.6 Experimental Results

In this section, the Meme-Net algorithm is tested on artificial generated networks and 4 real-world networks. To show the performance of Meme-Net, Fast Modularity algorithm is taken as a comparison. Some parameters and their values in Meme-Net are summarized in Table 2.2.

A similarity measure *Normalized Mutual Information* (NMI) [8] is used for estimating the similarity between the true partitions and the detected ones. Given two partitions  $A$  and  $B$  of a network in communities, let  $C$  be the confusion matrix whose element  $C_{ij}$  is the number of nodes of community  $i$  of the partition  $A$  that are also in the community  $j$  of the partition  $B$ . The normalized mutual information  $I(A, B)$  is defined as

$$I(A, B) = \frac{-2 \sum_{i=1}^{c_A} \sum_{j=1}^{c_B} C_{ij} \log(C_{ij}N / C_{i.}C_{.j})}{\sum_{i=1}^{c_A} C_{i.} \log(C_{i.}/N) + \sum_{j=1}^{c_B} C_{.j} \log(C_{.j}/N)} \quad (2.3)$$

where  $c_A(c_B)$  is the number of groups in the partition  $A(B)$ ,  $C_{i.}(C_{.j})$  is the sum of elements of  $C$  in row  $i$  (column  $j$ ), and  $N$  is the number of nodes (note that, some denominations here are different from the ones in previous sections just for convenience). If  $A = B$ , then  $I(A, B) = 1$ ; if  $A$  and  $B$  are completely different, then  $I(A, B) = 0$ .

**Table 2.2** Parameter settings for the algorithm

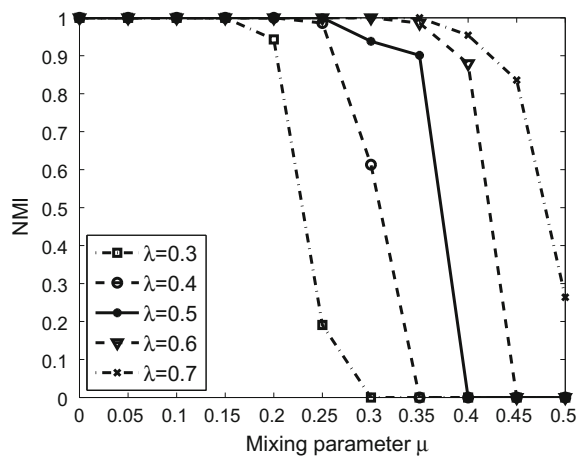
Parameter	Meaning	Value
$G_{max}$	The number of iterations	50
$S_{pop}$	Population size	450
$S_{pool}$	Size of the mating pool	$\frac{S_{pop}}{2}$
$S_{tour}$	Tournament size	2
$P_c$	Crossover probability	0.8
$P_m$	Mutation probability	0.2

### 2.2.6.1 Results on Artificially Generated Network

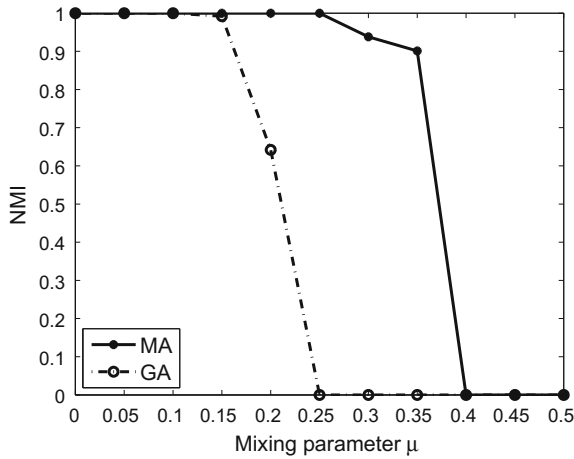
The artificially generated network used is the extension of GN benchmark benchmark network. 11 different networks are generated for the value of mixing parameter  $\mu$  ranging from 0 to 0.5 and NMI is used to measure the similarity between the true partitions and the detected ones. For each network, the average NMI is computed over 10 independent runs. Figure 2.1 shows the average NMI, for different values of  $\lambda$  (the parameter in the objective function  $D_\lambda$ ), when the mixing parameter increases from 0 to 0.5. As shown in Fig. 2.1, for  $\lambda = 0.5$ , when the value of mixing parameter  $\mu$  is small ( $\mu \leq 0.25$ ), which means the fuzziness of the community in the network is low, our algorithm find the true partition correctly (NMI equals 1). When the mixing parameter increases, it is more difficult to detect the true partition, but the detected partition is also close to the true one (NMI is about 0.9 when  $\mu = 0.35$ ). Only when  $\mu \geq 0.4$ , there is no community structure detected. However, the higher value of  $\lambda$  could help to discover smaller communities. For instance, for  $\lambda = 0.7$ , when the mixing parameter  $\mu = 0.45$ , Meme-Net is able to detect about 80% of the community structure information. On the other hand, lower value of  $\lambda$  could help to discover larger communities. For example, for  $\lambda = 0.3$ , when  $\mu \geq 0.3$ , Meme-Net tends to consider the whole network as a large single community (NMI equals 0). Notice that, when  $\mu = 0.5$ , each node has half of the links inside the community and the other half with the rest the network. This means that the community structure is very fuzzy, and any algorithm can hardly find the true partition of the network.

The local search procedure plays a very important role in Meme-Net. A genetic algorithm (GA) version of Meme-Net algorithm is developed by removing the LocalSearch function. Figure 2.2 shows the average NMI obtained by Meme-Net and this GA version algorithm, for  $\lambda = 0.5$ , when the mixing parameter increases from 0 to 0.5. It clearly shows that, with the local search procedure, when  $\mu \leq 0.25$ ,

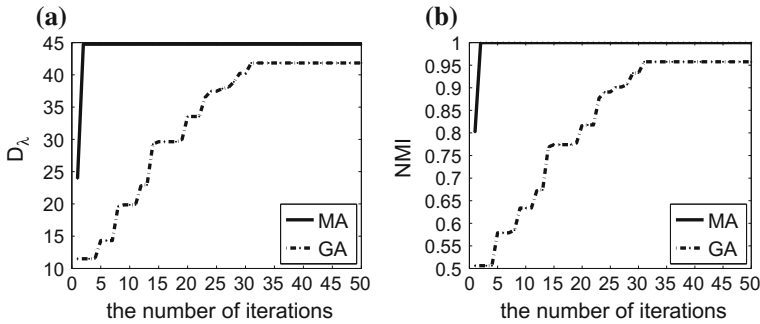
**Fig. 2.1** Average NMI versus mixing parameter  $\mu$ , for different values of  $\lambda$







**Fig. 2.2** Average NMI versus mixing parameter  $\mu$ , for  $\lambda = 0.5$



**Fig. 2.3** (a) The values of  $D_\lambda$ , (b) the corresponding NMI versus the number of iterations, for  $\mu = 0.15$  and  $\lambda = 0.5$

Meme-Net is able to detect the true partition of the network; however, without the local search procedure, it becomes harder for the algorithm to detect the true partition when  $\mu > 0.15$ .

The local search procedure also speeds up the convergence of Meme-Net. Figure 2.3 displays the values of  $D_\lambda$  and the corresponding NMI obtained by Meme-Net and the GA version algorithm in one run, for  $\mu = 0.15$  and  $\lambda = 0.5$ , when the number of generations increases from 1 to 50. The figure shows that with the local search procedure, Meme-Net finds a maximum objective function value of 44.75 in just two generations, and the corresponding NMI is 1, which means the true partition is found. However, without the local search procedure, after 50 generations, the algorithm does not find the true partition of the network.

### 2.2.6.2 Results on Real-World Networks

Meme-Net is also applied to four real-world networks, karate, dolphin, football, and polbooks. The results are also compared against those yielded by the Fast Modularity Algorithm.

$\lambda = 0.5$  is set to the default value for  $\lambda$  in the objective function  $D_\lambda$ . For each network, Meme-Net algorithm is run 30 times, the average value and standard deviation of NMI ( $I_{avg}$  and  $I_{std}$ ) over the 30 runs are computed, and the value of NMI and the number of clusters corresponding the maximum value of  $D_\lambda$  ( $I_{maxD}$  and  $N_{cluster}$ ) in the 30 runs are recorded. Then this is repeated for  $\lambda = 0.2$  to 0.8. The results are reported in Table 2.3. The statistic values of NMI over the 30 runs on the four real-world networks for different values of  $\lambda$  in terms of box plots are shown in Fig. 2.4. From Fig. 2.4, on each of the four networks, the variability of NMI values obtained over the 30 runs is relatively small, especially for some values of  $\lambda$ . For instance, On karate network, this is true for almost all the  $\lambda$  values.

On karate network, for  $\lambda = 0.5$ , in fact in all the 30 runs Meme-Net found a partition which consisted of 3 clusters and the corresponding NMI was 0.699. Additionally, the algorithm converges very fast, which just needs a few iterations (usually less than 10 generations). Although this detected partition is different from the true one, it is very meaningful. In fact, this solution splits one of the two large groups into two smaller ones and never misplace any node. Figure 2.5 displays the detected partitions corresponding  $I_{maxD}$  on this network for different values of  $\lambda$ . As we can see from the table, for  $\lambda = 0.2$ , the whole network is grouped into one cluster; for  $\lambda = 0.3$ , the network is grouped into 2 clusters (Fig. 2.5a), which is exactly the true partition, and the corresponding NMI is 1; for  $\lambda = 0.4$  or 0.5, the network is grouped into 3 clusters, which splits the left part of the network into two smaller ones (Fig. 2.5b); for  $\lambda = 0.6$  or 0.7, 4 clusters are found, and this solution splits each of the two large groups into two smaller ones (Fig. 2.5c); for  $\lambda = 0.8$ , the network is grouped into 5 clusters, which further splits the right part into 3 clusters (Fig. 2.5d). If  $\lambda$  is set to 0.9 or larger, many small clusters containing only two or three vertices are detected. We did not display this network partition in Fig. 2.5.

On dolphin network, for  $\lambda = 0.3$ , 2 clusters are found and the corresponding NMI is 1. This means that the detected partition is exactly the true one, which is shown in Fig. 2.6. For  $\lambda = 0.4$ , the network is grouped into 3 clusters and the corresponding NMI is 0.756. This partition splits the lower right group of the network into two smaller ones and never misplaces a vertex. For  $\lambda = 0.5$ , 5 clusters are found and the corresponding NMI is 0.586, which splits the upper left group of the network into 2 smaller ones, and the lower right group into 3 smaller ones. For  $\lambda = 0.6$  or larger, more smaller clusters are found. The experiments show that, by tuning the parameter  $\lambda$ , the network could be explored at different resolutions. In general, the larger  $\lambda$  value is, the smaller communities Meme-Net tends to find.

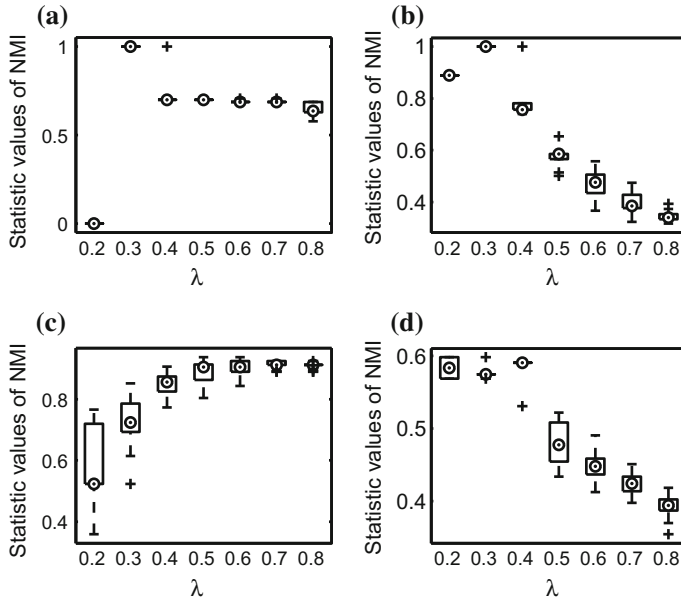
Because of the complexity of the networks themselves, It could not find the “true” partition on the football network (Fig. 2.7a) and the polbooks network (Fig. 2.7b). However, the detected ones are very close to the true partitions. For instance, on the

**Table 2.3** The results of 30 runs of Meme-Net on four real-world networks for different values of  $\lambda$ 

Network	$\lambda$	$I_{avg}$	$I_{std}$	$I_{maxD}$	$N_{cluster}$
Zachary's karate club	0.2	0	0	0	1
	0.3	1	0	1	2
	0.4	0.740	0.104	0.699	3
	0.5	0.699	0	0.699	3
	0.6	0.690	0.007	0.687	4
	0.7	0.688	0.004	0.687	4
	0.8	0.651	0.038	0.628	5
Dolphin social network	0.2	0.889	0	0.889	2
	0.3	1	0	1	2
	0.4	0.787	0.073	0.756	3
	0.5	0.569	0.035	0.586	5
	0.6	0.467	0.048	0.477	9
	0.7	0.400	0.034	0.385	11
	0.8	0.346	0.017	0.334	14
American college football	0.2	0.595	0.118	0.359	2
	0.3	0.723	0.078	0.523	3
	0.4	0.851	0.032	0.824	8
	0.5	0.890	0.033	0.911	11
	0.6	0.904	0.022	0.924	12
	0.7	0.912	0.011	0.911	13
	0.8	0.911	0.010	0.911	13
Books about US politics	0.2	0.583	0.015	0.598	2
	0.3	0.576	0.008	0.574	3
	0.4	0.588	0.011	0.590	4
	0.5	0.481	0.029	0.455	7
	0.6	0.449	0.018	0.434	9
	0.7	0.423	0.013	0.402	11
	0.8	0.394	0.014	0.378	14

football network, for  $\lambda = 0.5$ , 11 clusters are found and the corresponding NMI is 0.911, only a few vertices are misplaced. On the polbooks network, the results are also competitive with other popular community detection algorithms.

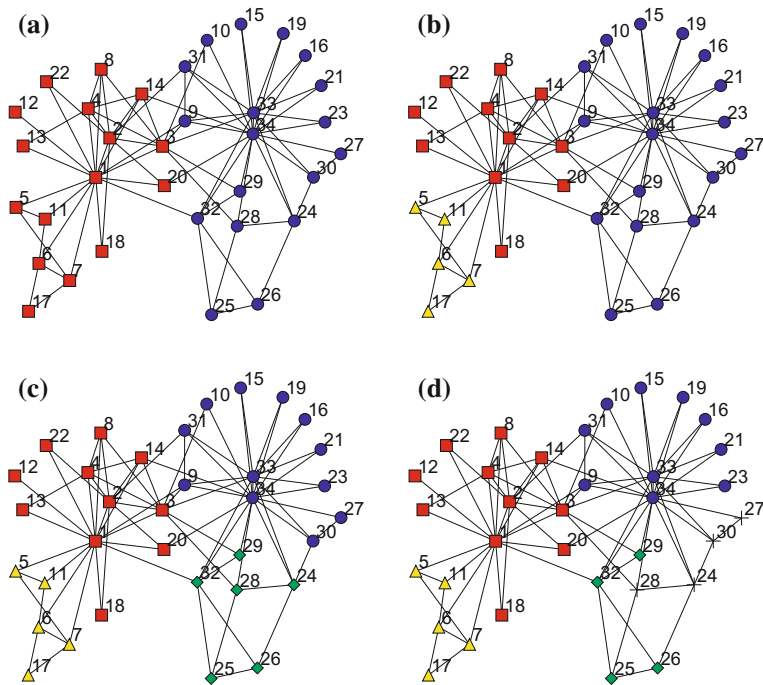
The results obtained by Fast Modularity algorithm are given in Table 2.4. Now the results obtained by Meme-Net for  $\lambda = 0.5$  are considered. On karate network, the Fast Modularity algorithm found a solution with  $NMI = 0.693$ , while in fact for all the 30 runs Meme-Net found a solution with a NMI value of 0.699. On the



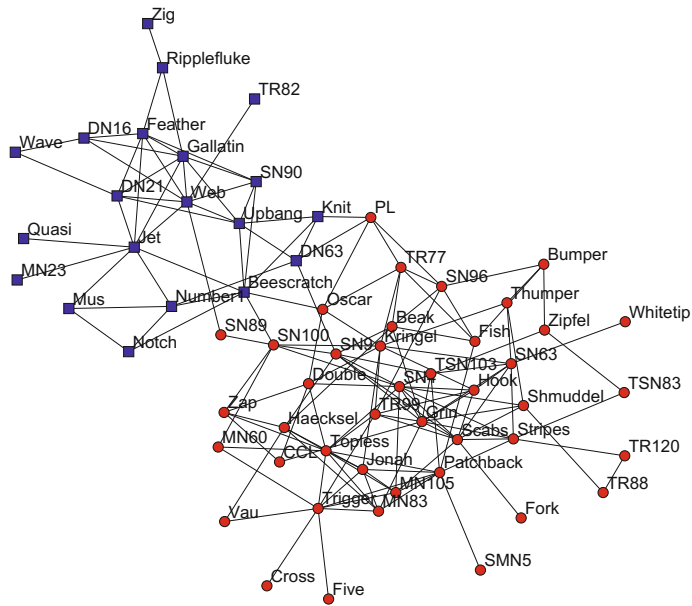
**Fig. 2.4** The statistic values of NMI over the 30 runs on (a) Zachary's karate club, (b) Dolphin social network, (c) American college football, (d) Books about US politics for different values of  $\lambda$ . Here, box plots are used to illustrate the distribution of these samples. On each box, the central mark  $\odot$  is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme datapoints the algorithm considers to be not outliers, and the outliers are plotted individually. Symbol + denotes outliers

football network, the average NMI value found by Meme-Net is 0.890, while Fast Modularity algorithm found the NMI value of 0.762. On these two networks the results obtained by Meme-Net are better than those of Fast Modularity algorithm. On the dolphin network and polbooks network, the average values of NMI found by Meme-Net are 0.569 and 0.481 respectively, while the Fast Modularity algorithm found the NMI values of 0.573 and 0.531, which are slightly better than Meme-Net. However, by tuning the parameter  $\lambda$ , we can also get better results. For example, on the dolphin network, when  $\lambda = 0.3$ , the solution found by Meme-Net is exactly the true partition. On polbooks network, when  $\lambda = 0.4$ , the solutions with the average NMI value of 0.588 found by Meme-Net are more closer to the true partition. This comparison clearly shows the very good performance of Meme-Net with respect to the Fast Modularity algorithm.

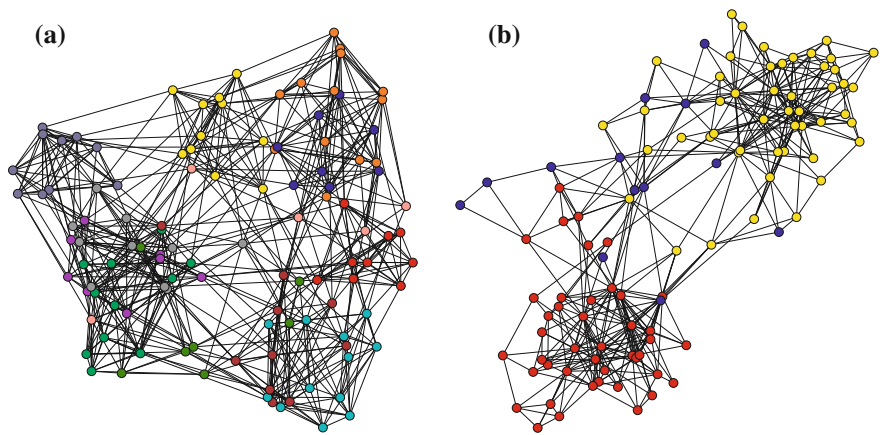
In practice,  $\lambda = 0.5$  is set as the default value for  $\lambda$  in the objective function  $D_\lambda$ , because this value indeed produces good results as shown in the experiments.



**Fig. 2.5** The detected partitions on karate network for (a)  $\lambda = 0.3$ , (b)  $\lambda = 0.4$  or  $0.5$ , (c)  $\lambda = 0.6$  or  $0.7$ , (d)  $\lambda = 0.8$



**Fig. 2.6** The Dolphin social network



**Fig. 2.7** a The football network, b The polbooks network

**Table 2.4** The results obtained by fast modularity algorithm

Network	The number of clusters	NMI
Karate	3	0.693
Dolphin	4	0.573
Football	7	0.762
Polbooks	4	0.531

2.2.7 Conclusions

In this section, a basic memetic algorithm for community discovery is introduced, named Meme-Net. Meme-Net is a synergy of genetic algorithm with a hill-climbing strategy as the local search procedure. Experiments show that combining the local search procedure, Meme-Net performs well in community discovery. By tuning the parameter  $\lambda$  in modularity density, Meme-Net can uncover communities at different hierarchical levels.

2.3 A Multilevel Learning-Based Memetic Algorithm for Community Discovery in Large-Scale Networks

<sup>1</sup>In Sect. 2.2, a basic memetic community discovery, Meme-Net, was introduced. Meme-Net is a synergy of a genetic algorithm with a hill-climbing strategy as the local search procedure. Note that, the local search in Meme-Net has a high computational

<sup>1</sup>Acknowledgement: Reprinted from Applied Soft Computing, 19, Ma, L., Gong, M., Liu, J., Cai, Q., Jiao, L., Multi-level learning based memetic algorithm for community detection, 121–133, Copyright (2014), with permission from Elsevier.

complexity, which makes it impossible to discover communities on large scale of networks.

This section introduces a memetic algorithm with multi-level learning strategies (MLCD) to detect communities. MLCD aims to maximize modularity. The advantages of MLCD are as follows. The proposed multi-level learning strategies work on the network at node, cluster, and network partition levels, respectively. By iteratively executing GA and multi-level learning strategies, a network partition with high modularity can be accurately and stably obtained. A modularity-specific label propagation rule is employed to update the cluster identifier of each node at each operation. The simple update rule guarantees the rapidity of the proposed algorithm.

### 2.3.1 *Memetic Algorithm with Multi-level Learning for Community Discovery*

The framework of MLCD is similar to that of Meme-Net, as shown in Algorithm 1. In what follows, more detailed descriptions about the initialization procedure, genetic operators and the multi-level learning strategies will be given.

### 2.3.2 *Representation and Initialization*

Like Meme-Net, the string-based representation is also adopted in MLCD, but the initialization is different. The initialization process of MLCD can be described as follows: firstly, each gene in chromosome is put in a different cluster (i.e.,  $x_a^i \leftarrow i$ ,  $1 \leq i \leq n$ ,  $1 \leq a \leq N_p$ ). Then, generate a random sequence. Next, for each vertex  $v_i$  chosen in that sequence, update its cluster identifier with one of its neighbors' (i.e.,  $x_a^i \leftarrow x_a^j$ ,  $\exists j \in \{j | A_{ij} = 1\}$ ). The above processes independently run  $N_p$  times, and a population of solutions are generated. The solution, which has maximal modularity in the population, is chosen as  $x_g$ .

### 2.3.3 *Genetic Operators*

**Crossover.** The traditional crossover operations, including the uniform crossover [22] and the two-point crossover [14], have great randomness and large uncertainty, and thus the resulting descendants can hardly inherit useful communities from their parents. In MLCD, a two-way crossover operation is employed. The crossover procedure is given as Algorithm 4. First, randomly choose two chromosomes  $x_a$  and  $x_b$  from  $X_C$ . Then, randomly choose a vertex  $v_i$  and identify the cluster identifiers of node  $v_i$  in  $x_a$  and  $x_b$  ( $x_a^i$  and  $x_b^i$ , respectively). Next, generate a random value in

the range of 0–1. If the generated value is larger than the crossover probability  $p_c$ , then assign the cluster identifier of the corresponding vertices which have the same cluster identifier as  $x_a^i$  in  $x_a$  with  $x_a^i$  in  $x_b$  (i.e.,  $x_b^j \leftarrow x_a^i, \forall j \in \{j \mid x_a^j = x_a^i\}$ ). At the same time, assign the cluster identifier of the corresponding vertices which have the same cluster identifier as  $x_b^i$  in  $x_b$  as  $x_b^i$  in  $x_a$  (i.e.,  $x_a^j \leftarrow x_b^i, \forall j \in \{j \mid x_b^j = x_b^i\}$ ). The above processes repeatedly operate  $\lfloor N_c/2 \rfloor$  times.

This two-way crossover operation can generate descendants which inherit most communities from their parents. As shown in Fig. 2.8, a toy network  $G$  has two communities  $s_1 = \{v_1, v_2, v_3, v_4\}$  and  $s_2 = \{v_5, v_6, v_7\}$ . The parent chromosomes  $x_a = \{1, 1, 1, 1, 2, 3, 4\}$  and  $x_b = \{1, 2, 3, 4, 5, 5, 5\}$  have the community structure  $s_1$  and  $s_2$ , respectively. Assuming that the vertex  $v_5$  is chosen, in the following operations, the nodes  $v_5, v_6$  and  $v_7$  in  $x_a$  are assigned with the cluster identifier of the corresponding nodes in  $x_b$ , and thus produce a descendant  $x_c = \{1, 1, 1, 1, 5, 5, 5\}$ .

---

**Algorithm 4** Two-way crossover
 

---

**Input:**  $X_C$  and the crossover probability  $p_c$ .

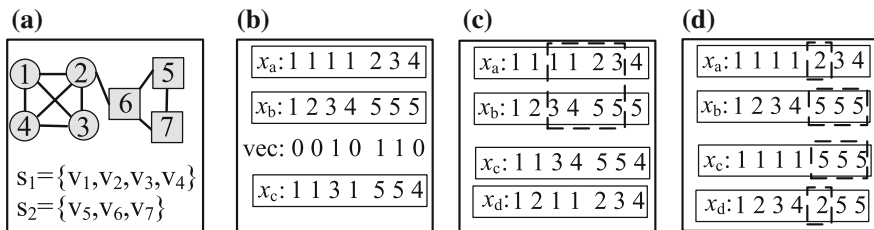
**Output:** the offspring chromosome.

```

1:  $r \leftarrow 1$ 
2: while  $r \leq \lfloor N_c/2 \rfloor$  do
3:   Randomly select two chromosomes  $x_a$  and  $x_b$  ( $a \neq b$ ) from  $X_C$  and randomly select a
      vertex  $v_i$ .
4:   Randomly generate a value  $f$  within  $[0, 1]$ .
5:   if  $f \leq p_c$  then
6:      $x_b^j \leftarrow x_a^i, \forall j \in \{j \mid x_a^j = x_a^i\}$ .
7:      $x_a^j \leftarrow x_b^i, \forall j \in \{j \mid x_b^j = x_b^i\}$ .
8:   end if
9:    $r \leftarrow r + 1$ 
10: end while
  
```

---

However, as for the uniform and two-point crossover operations, they can hardly inherit the communities from their parents.



**Fig. 2.8** Illustration of the crossover operations. **a** a toy network with two communities  $s_1$  and  $s_2$ . **b** the Uniform crossover. The resulting solution  $x_c$  is generated by selecting the genes where the binary vector  $vec$  is 0 from  $x_a$ , and the genes where the binary vector  $vec$  is 1 from  $x_b$ . **c** Two-point Crossover. The resulting solutions  $x_c$  and  $x_d$  are generated by swapping the cluster identifier between the two chosen nodes (eg.  $v_3$  and  $v_6$ ) in  $x_a$  and  $x_b$ . **(d)** Two-way crossover



**Mutation.** In order to decrease useless exploration, a neighbor-based mutation operator is used which considers the effective connections among nodes. The neighbor-based mutation procedure works on the generated population  $X_C$ , and it works as follows. For each node  $v_i$  in a chromosome  $x_c$  ( $x_c \in X_C$ ), first, generate a random value in the range of 0–1. If the generated value is smaller than the mutation probability  $p_m$ , then the cluster identifier of the node  $v_i$  is randomly mutated to one of its neighbors' (i.e.,  $x_c^i \leftarrow x_c^j, \exists j \in \{j | A_{ij} = 1\}$ ). Among the generated offspring chromosomes, the chromosome, which has the maximal modularity value, is chosen as  $x_l$ .

### 2.3.4 Multi-level Learning Strategies

In MLCD, a multi-level learning based local search is proposed to refine the individual  $x_l$ . The multilevel learning based local search is proposed based on the potential knowledge of the node, community and partition structures of networks, and it is composed of three learning techniques from low level to high level. Each level learning technique can converge to a local optimal solution rapidly and the higher level learning techniques have the ability to escape from the local optima obtained by the lower level learning techniques. In the realization of the proposed multi-level learning techniques, in order to avoid generating unnecessary divisions containing unconnected nodes and communities and to find the best solution as soon as possible, the cluster identifier of nodes and communities with their neighbors' are updated when the increment of modularity  $\Delta Q$  is maximal. Moreover, a modularity-specific label propagation rule is employed to update the cluster identifier of each node and community at each operation. This simple update rule guarantees the rapidity of the proposed multi-level learning techniques. In addition, in order to reduce the sensitivity of the proposed learning techniques to the optimized order, the cluster identifiers of nodes and communities are updated in a random order.

#### 2.3.4.1 The First-Level Learning

A node-level learning strategy, which is similar to the algorithm LPAm [2], is chosen as the first-level learning technique, and it takes  $x_l$  as the initial solution. The node-level learning strategy works on each node of the network and its processes is shown in Algorithm 5. Given a network  $G$  with  $n$  nodes, the first-level learning strategy works as follows. Firstly, generate a random sequence (i.e.,  $\{r_1, r_2, \dots, r_n\}$ ). Then, for each vertex  $v_{r_i}$  chosen in that generated sequence, update its cluster identifier  $x_l^{r_i}$  with one of its neighbors' using the modularity-specific label propagation rule described in Sect. 2.3.4.4. The above processes end when the cluster identifier of every node has no change. The first-level learning strategy can help GA quickly converge to an optimal solution around one of its search space.

**Algorithm 5** Node-level learning strategy.**Input:**  $x_l$  and  $G$ . **Output:**  $x_l$ .

---

```

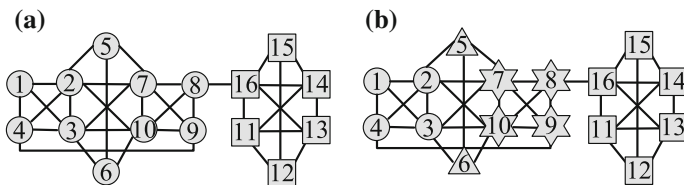
1: repeat
2:   Generate a random sequence (i.e.,  $\{r_1, r_2, \dots, r_n\}$ ).
3:   for each node  $v_{r_i}$  of  $G$  do
4:     Update  $x_l^{r_i}$  using the modularity-specific label propagation rule.
5:   end for
6: until the cluster identifier of each node is not changed.

```

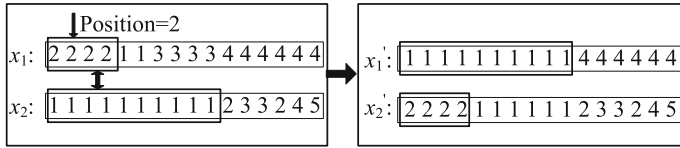
---

Note that, the first-level learning strategy tends to uncover a network partition with similar communities in total degree, and thus it is easy to fall into a local optimum. For example, as is shown in Fig. 2.9, for a toy network  $G_1$  with 16 nodes, the intuitive community division divides the network into two communities (drawn in circle and square shapes respectively) with  $Q = 0.4127$ . The first-level learning strategy tends to divide this network into four communities with  $Q = 0.3988$ , as is shown in Fig. 2.9b. The modularity value of this network partition is not increased by removing any node from its community and placing it in that of its neighbors'. Evidently, the community division obtained by the first-level learning strategy falls into a local optimal solution. Therefore, it is necessary to take measures to help the first-level learning strategy escape from its local optimum.

Actually, for small size networks, the hybrid technique, which iteratively employs GA and the first-level learning strategy, can escape from the local optimum obtained by the first-level learning strategy. A schematic illustration of how GA helps the first-level learning strategy escape from its local optimum is shown in Fig. 2.10. Assuming that the local optimal solution obtained by the first-level learning strategy is  $x_1$  and there is one solution like  $x_2$  in the population, the offspring chromosome  $x'_1$  which corresponds to the true community division of the toy network  $G_1$  is generated after employing the two-way crossover operation on the two chromosomes  $x_1$  and  $x_2$ . Therefore, the hybrid algorithm, termed as M-LPAm, which iteratively performs GA and the first-level learning strategy, has the ability to help the first-level learning strategy escape from its local optimum. However, for the larger scale networks, M-LPAm is difficult to solve this problem. This is because the probability of the existence of the chromosome like  $x_2$  in the population becomes smaller. A community-level learning strategy is devised to help the first-level learning strategy further escape from its local optimum.



**Fig. 2.9** A schematic description of the partitions of a toy network  $G_1$ . **a** The best community division with  $Q = 0.4127$  divides the network into two communities. **b** The first-level learning strategy divides this network into four communities with  $Q = 0.3988$



**Fig. 2.10** A schematic illustration of how the two-way crossover operation helps the first-level learning strategy escape from its local optimum on a toy network  $G_1$ . Chromosome  $x_1$  is obtained by the first-level learning strategy and chromosome  $x_2$  comes from the chosen population. After employing the two-way crossover  $x_1$  and  $x_2$ , an offspring chromosome  $x_1'$  which is the same as the true community division of the toy network is generated

### 2.3.4.2 The Second-Level Learning

A community-level learning technique which is similar to the second phase of the BGLL algorithm [3] is devised as the second-level learning strategy, and it is shown in Algorithm 6. The second-level learning strategy works on each community of the solution  $x_l$  generated by the first-level learning strategy. Assuming that the chromosome  $x_l$  has  $k$  communities (e.g.,  $s_1, s_2, \dots, s_k$ ), the second-level learning strategy works as follows. Firstly, a new network, whose nodes are now the communities of  $x_l$ , is generated. In this case, the links between the new nodes  $v'_i$  and  $v'_j$  are the total links between the nodes in the corresponding communities  $s_i$  and  $s_j$  of  $x_l$ . A schematic illustration of the above operation is given in Fig. 2.11a with a toy network  $G_1$ . The original partition of the toy network  $G_1$  is composed of 4 communities, which have 4, 2, 4, and 6 nodes respectively. In the generated new network, there are four nodes and each node corresponds to one community of the original network. The weight of links between the new nodes is the total weight of links between nodes in the corresponding communities. Second, assign each new node with a unique cluster identifier, i.e.,  $X^i = i$ ,  $1 \leq i \leq k$ , where  $X^i$  represents the cluster identifier of the new node  $v'_i$ . Thirdly, employ the first-level learning strategy on  $X$  to find a better partition of the new network  $G'$ . In Algorithm 6, the function of NodeLearning() is used to represent the node-level learning technique. Finally, decode the partition of new network  $G'$  to the community division of the original network  $G$ . The second-level learning procedure returns a new chromosome  $x_e$ . Compared with the second phase of the BGLL algorithm, the community-level learning technique has different strategies in terms of the optimization order and the cluster identifier update rule.

---

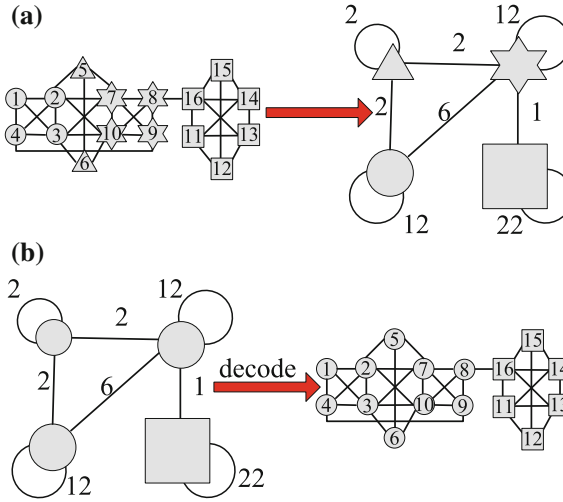
#### Algorithm 6 Community-level learning strategy.

---

**Input:**  $x_l$  and  $G$ .

**Output:**  $x_e$ .

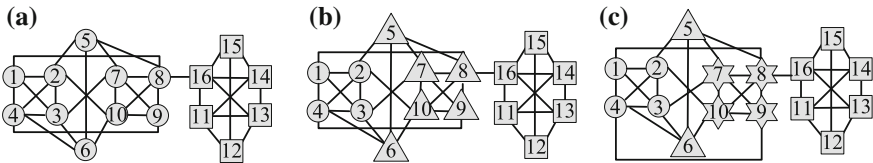
- 1: Generate a new network  $G' = (V', E')$ , where  $V' = \{v'_1, v'_2, \dots, v'_k\}$ .  $v'_i$  is the corresponding community  $s_i$  in  $x_l$ ,  $1 \leq i \leq k$ . The connections between the new nodes can be represented as an adjacency matrix  $A'$ , whose elements  $A'_{ij}$  is computed as  $A'_{ij} \leftarrow \sum_{i \in s_i} \sum_{j \in s_j} A_{ij}$ .
  - 2: Set  $X^i = i$ ,  $1 \leq i \leq k$ .
  - 3:  $X \leftarrow \text{NodeLearning}(X, G')$ .
  - 4: **while**  $i \leq k$  **do**
  - 5:    $x_e^j \leftarrow X^i, \forall j \in \{j | v_j \in s_i\}$ .
  - 6: **end while**
-



**Fig. 2.11** A schematic illustration of the second-level learning technique. **a** Generate a new network. *Left* the original network  $G_1$ . *Right* the new network. **b** Generate a new partition of the toy network  $G_1$  by employing the second-level learning strategy. The new partition is the same as the true one

The second-level learning strategy can help the first-level learning strategy jump out of its local maximum. As is shown in Fig. 2.11b, after merging the upper left three communities, the generated partition is the same as the true one with  $Q = 0.4238$ .

Note that, the first two level learning strategies are also easy to fall into local optimal solutions. This is because the merged communities can hardly be separated again. Therefore, the new partition may not always be good enough (although it is better than the previous local maximum). The above view is confirmed by Rosvall and Axelsson [23]. For example, as is shown in Fig. 2.12a, the new toy network  $G_2$  with 16 nodes is intuitively divided into three communities (drawn in circle, triangle, and square shapes respectively), with  $Q = 0.4298$ . After performing the first-level learning and the second-level learning strategies, the obtained partitions tend to



**Fig. 2.12** A schematic description of the partitions of a new toy network  $G_2$ . **a** The best network partition with modularity  $Q = 0.4298$  divides the network in three communities. **b** The first-level learning strategy is easy to a local maximum where the network is divided into 4 communities with  $Q = 0.4094$ . **c** The technique, which combines the first-level learning strategy and the second-level learning strategy, is easy to get into a local maximum where the network is divided into 2 communities with  $Q = 0.4127$

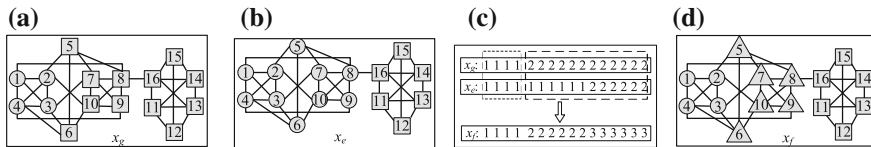
divide the toy network into four communities with  $Q = 0.4094$  (Fig. 2.12b) and two communities with  $Q = 0.4127$  (Fig. 2.12c), respectively. Evidently, these divisions correspond to local optima in the modularity space. As is shown in Fig. 2.12c, the left large community can hardly be separated again by performing the first two level learning strategies.

In this study, a structural learning strategy, which works on the network partition layer, is proposed to help the first two level learning strategies escape from their local optima.

### 2.3.4.3 The Third-Level Learning

The third-level learning strategy is the proposed partition-level learning procedure, and it is shown in Algorithm 7. It works on two chromosomes,  $x_g = \{g_1, g_2, \dots, g_{k_1}\}$  and  $x_e = \{s_1, s_2, \dots, s_{k_2}\}$ , where  $k_1$  and  $k_2$  are the number of clusters of  $x_g$  and  $x_e$ , respectively. The partition-level learning strategy consists of two phases, and its first phase works as follows. For each cluster  $g_i$ ,  $1 \leq i \leq k_1$ , of  $x_g$ , the corresponding nodes in  $g_i$  are divided into a set of clusters according to the following two principles: (1) If the nodes are in the same cluster in both  $x_g$  and  $x_e$ , they are divided into the same cluster; (2) If the nodes are in the same cluster in  $x_g$ , while they are in different clusters in  $x_e$ , they are divided into different clusters. The first phase returns a basic consensual network partition  $x_f = \{g'_1, g'_2, \dots, g'_{k_3}\}$ , where  $k_3$  is the number of clusters of  $x_f$  and its value depends on the difference between  $x_g$  and  $x_e$ . The second phase is employing the first two level learning strategies on  $x_f$  to find a better community division of networks. In Algorithm 7, the functions of NodeLearning() and CommunityLearning() are used to represent the node-level learning and the community-level learning techniques, respectively. The partition-level learning procedure returns a new chromosome  $x_l$ .

A schematic illustration of the partition-level learning strategy on two individuals  $x_g$  and  $x_e$  is given in Fig. 2.13c. As is shown in Fig. 2.13c, individual  $x_g$  has two communities  $g_1 = \{v_1, v_2, v_3, v_4\}$  and  $g_2 = \{v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}\}$ , and individual  $x_e$  has two communities  $s_1 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$  and  $s_2 = \{v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}\}$ . For the nodes of the first



**Fig. 2.13** A schematic illustration of the partition-level learning technique. **a** and **b** are the corresponding network partitions of the individuals  $x_g$  and  $x_e$ , respectively. **c** Generate an individual  $x_f$  by performing the partition-level learning technique on  $x_g$  and  $x_e$ . **d** is the corresponding network partition of the generated individual  $x_f$

cluster  $g_1$  of  $x_g$ , they are in the same cluster in  $x_e$ . Therefore, in the generated individual  $x_f$ , the nodes  $v_1, v_2, v_3$  and  $v_4$  form a cluster. For the nodes of the second community  $g_2$  of  $x_g$ , the nodes  $v_5, v_6, v_7, v_8, v_9$  and  $v_{10}$  have the same cluster identifier 1 in  $x_e$ , and the nodes  $v_{11}, v_{12}, v_{13}, v_{14}, v_{15}$  and  $v_{16}$  have the same cluster identifier 2 in  $x_e$ . Therefore, in the generated individual  $x_f$ , the nodes  $v_5, v_6, v_7, v_8, v_9$  and  $v_{10}$  form a cluster and the nodes  $v_{11}, v_{12}, v_{13}, v_{14}, v_{15}$  and  $v_{16}$  form a new cluster.

The partition-level learning strategy can help the first two level learning strategies to escape from their local maxima. As is shown in Fig. 2.13, after performing the partition-level learning strategy, the large community  $s_1$  of  $x_e$  has been broken into two communities and the true partition has been uncovered.

---

**Algorithm 7** Partition-level learning strategy
 

---

**Input:**  $G, x_g = \{g_1, g_2, \dots, g_{k_1}\}$  and  $x_e = \{s_1, s_2, \dots, s_{k_2}\}$ . **Output:**  $x_{f'}$ .

```

1: Set  $x_f^i \leftarrow 0, i = 1, 2, \dots, n$ , and  $C \leftarrow 0$ ;
2: for each cluster  $g_i$  of  $x_g$  do
3:   Identify the nodes in  $g_i$  and set them as  $g_i = \{v_{i_1}, v_{i_2}, \dots, v_{i_{|g_i|}}\}$ , where  $|g_i|$  is the size of
      $g_i$ .
4:   for each node  $v_{i_j}$  of  $g_i$  do
5:     if  $x_f^{i_j} = 0$  then
6:        $C \leftarrow C + 1$ ;
7:        $x_f^q \leftarrow C$ , where  $\forall q \in \{q | v_q \in g_i \& x_e^{i_j} = x_e^q\}$ ;
8:     end if
9:   end for
10: end for
11:  $x_f \leftarrow \text{NodeLearning}(x_f, G)$ .
12:  $x_{f'} \leftarrow \text{CommunityLearning}(x_f, G)$ .
  
```

---

The reasons why the partition-level learning strategy can improve the accuracy of the first two level learning strategies are as follows. First, the first two level learning strategies tend to get into local optimal network partitions, mainly because it can hardly be separated again after merging two or more nodes and communities together and forming a single one. The partition-level learning strategy makes it possible for the merged nodes and communities to be separated again. After reconstructing the separated nodes and communities, a good network partition with high modularity value can be obtained, as is shown in Fig. 2.13. Moreover, the partition-level learning strategy is a consensus clustering technique working on the solutions  $x_g$  and  $x_e$ . The same divisions between  $x_g$  and  $x_e$  are preserved, and the differences between them are divided. Therefore, the partition-level learning strategy collects the structure property from two or more solutions, rather than one solution. It makes it possible to generate better solutions than that uncovered by the first two level learning strategies which just collect the structure property from one possible solution. Finally, the structural learning strategy can generate new solutions which are different from  $x_g$  and  $x_e$ . Therefore, it can enhance both the population diversity and the global searching capability of GA.

### 2.3.4.4 Modularity-Specific Label Propagation Update Rule

In MLCD, in order to decrease the computation complexity, a simple way is adopted to calculate the  $\Delta Q$  obtained by moving an isolated node  $v_i$  into its neighbor cluster  $s_j$ . The  $\Delta Q$  can easily be computed by:

$$\begin{aligned}\Delta Q &= \left[ \frac{l_{s_j} + l_{i,s_j}}{m} - \left( \frac{k_{s_j} + k_i}{2m} \right)^2 \right] - \left[ \frac{l_{i,i}}{m} - \left( \frac{k_{s_j}}{2m} \right)^2 - \left( \frac{k_i}{2m} \right)^2 \right] \\ &= \frac{l_{i,s_j}}{m} - \frac{k_i k_{s_j}}{2m^2}\end{aligned}\quad (2.4)$$

where  $l_{s_j}$  is the number of links inside cluster  $s_j$ ,  $l_{i,s_j}$  is the number of links from node  $v_i$  to nodes in  $s_j$ ,  $k_i$  represents the degree of the node  $v_i$  and  $k_{s_j}$  is the total degree of nodes in  $s_j$ . Let  $x^i$  and  $C_i$  denote the cluster identifier of the node  $v_i$  and the cluster  $s_i$ , respectively, the above equation also can be written in terms of the adjacency matrix  $A$  as follow

$$\Delta Q = \frac{1}{m} \left( \sum_{q \neq i} A_{iq} \delta(x^q, C_j) \right) - \frac{1}{2m} k_i \sum_{q \neq i} k_q \delta(x^q, C_j), \quad (2.5)$$

where  $\delta$  is the Kronecker delta. In this study, the maximum  $\Delta Q$  is chosen as the update rule for updating the cluster identifier of each isolated node  $v_i$ . Therefore, the new cluster identifier  $X^i$  of the node  $v_i$  can be written as

$$X^i = \arg \max_C \left( \sum_{q \neq i} A_{iq} \delta(x^q, C) - \frac{1}{2m} k_i \sum_{q \neq i} k_q \delta(x^q, C) \right). \quad (2.6)$$

The rule of Eq. (2.6) is the modularity-specific label propagation update rule. As is known from Eq. (2.6), when  $X^i = x^i$ , it indicates that updating the cluster identifier of  $v_i$  with any one of its neighbors' cannot increase the value of modularity. When  $X^i \neq x^i$ , it indicates that updating the cluster identifier of  $v_i$  with  $X^i$  produces the maximal increment of modularity.

### 2.3.5 Complexity Analysis of MLCD

Here, the time complexity of the proposed algorithm MLCD is analyzed. Given a network with  $n$  nodes and  $m$  edges, at each generation, firstly, the crossover operator is performed  $\lfloor N_c/2 \rfloor$  times and the mutation operator  $N_c$  times at most, where  $N_c$  is the size of the chosen population. The time complexity of the calculation of modularity is  $O(m)$ . Therefore, the time complexity of the genetic operator is  $O(N_c(n + m))$ . Then, the multi-level learning strategies are performed. The time complexity of the first-level learning strategy is  $O(rm)$ , where  $r$  is the number of steps required to reach

a local maximum in modularity space. The second-level learning algorithm needs to merge pairs of communities which requires  $O(m \log n)$  basic operations. Therefore, the time complexity of the second-level learning algorithm is  $O(hm \log n)$ , where  $h$  is the number of steps required to reach a local maximum in modularity space. The time complexity of the first step and the second step of the partition-level learning strategy are  $O(n)$  and  $O(rm + hm \log n)$ , respectively. Generally,  $r = \log n$  and  $h \approx \log n$ . Therefore, the overall time cost for the multi-level learning strategies at each generation is  $O(m(\log n)^2)$ . Finally, the population is updated, which needs  $O(N_p + N_c)$  basic operations. In practical applications,  $N_p + N_c \leq N_c(n + m) \leq m(\log n)^2$ . Therefore, the time complexity of the proposed MLCD algorithm at each generation is  $O(m(\log n)^2)$ .

### 2.3.6 Comparisons Between MLCD and Meme-Net

By comparing MLCD with Meme-Net, it can be seen that both of them model the community detection in networks into an optimization problem, and use the framework of memetic algorithm to solve the modeled optimization problem. Moreover, both of them have excellent performance in discovering the potential community structure of networks. However, they are different in the following aspects.

First, the motivation of Meme-Net is to find a set of hierarchical community structures on small-scale networks by optimizing an objective with a resolution parameter  $\lambda$ . To reveal a community structure at a certain hierarchical level, it is necessary to set the value of  $\lambda$  in advance. The motivations of MLCD are to find the best community division of networks quickly and stability without knowing the number of clusters in advance, and to extend Meme-Net to handle larger scale networks.

Secondly, the modeled optimization problems are different. Meme-Net and MLCD model the community detection of networks into an optimization problem based on modularity density and modularity, respectively. The optimization of modularity density tends to reveal a community structure of a network which has greater internal link density than external link density of communities, while the optimization of modularity tends to discover a community structure which has more intra-community links than inter-community links.

Moreover, Meme-Net adopts a hill-climbing strategy based local search which works on a network at nodes level, while MLCD uses the proposed multilevel learning techniques based local search which works on a network at nodes, communities, and network partitions levels, respectively. Compared with the proposed multilevel learning techniques, the hill-climbing strategy has higher computational complexity. Given a network with  $n$  nodes and  $m$  edges, at each step of the hill-climbing technique, it needs to consider all the possible network partitions. Moreover, it requires  $\log n$  steps to reach a local maximum in the modularity density space. In addition, the computation of the modularity density of each network partition requires  $O(m)$  basic



operations. In the worst case, the hill-climbing strategy is necessary to consider  $n^2$  network partitions. Therefore, the overall time cost of the hill-climbing technique at each generation is  $O(mn^2 \log n)$ . For a sparse network  $n \log n \approx m$ , the time complexity of the hill-climbing strategy is  $O(m^2n)$ , which is far greater than that of GA,  $O(N_p m)$ , where  $N_p$  is the size of the population. Due to the high computational complexity ( $O(m^2n)$ ), Meme-Net can only handle small-scale networks. For the proposed algorithm MLCD, its low computational complexity ( $O(m(\log n)^2)$ ) makes it possible to tackle larger scale networks.

In addition, Meme-Net and MLCD have different initialization process. In the initialization of Meme-Net, the cluster identifiers of  $\alpha n$  nodes are assigned to all of their neighbors. The diversity and the quality of the generated initial population are controlled by the parameter  $\alpha$ . For the MLCD, it no longer needs a controlling parameter to adjust the diversity and the quality of the generated initial population. The initialized solutions are generated by assigning the cluster identifier of each node with one of its neighbors'.

Finally, the experimental comparisons between MLCD and Meme-Net demonstrate that MLCD has superior performance than Meme-Net in terms of the quality of the detected community structure, the stability and the time consumption.

### 2.3.7 Experimental Results

MLCD is tested on the extension of GN and LFR benchmarks networks and 12 real-world networks. The comparisons between MLCD and its three variants, M-two-phase, M-LPAm and GA, are made to illustrate the effectiveness of each level learning strategy. The algorithms GA and M-two-phase are the variants of MLCD by removing the multi-level learning strategies and the partition-level learning strategy, respectively. The algorithm M-LPAm is the variant of MLCD by removing both the partition-level learning and the community-level learning strategies. Moreover, the comparisons between M-two-phase (M-LPAm) and two-phase (LPAm) are given to show the effectiveness of the hybrid technique. The algorithm two-phase (LPAm) is the variant of M-two-phase (M-LPAm) by removing the genetic algorithm. In addition, the results obtained by the representative community detection algorithms FM [6], BGLL [3], Infomap [24], MOGA-Net [22], MODPSO [11] and Meme-Net [13] are also given for comparisons.

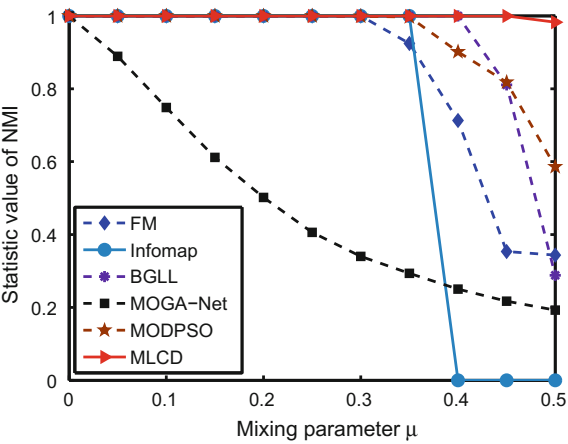
The parameter settings of MLCD and some compared algorithms are listed in Table 2.5. The key parameters, including the population size, the chosen population size, the crossover rate, the mutation rate and the maximum generation, are set to the same values. For each network, all algorithms are independently run 50 times.

As the two classes of benchmarks networks have known partitions, the evaluation criterion Normalize Mutual Information ( $NMI$ ) (Eq. 2.3) is also adopted to estimate the similarity between the true partition and the detected one.

**Table 2.5** Parameter settings of the compared algorithms

Parameter	Meaning	MLCD	MOGA-Net	MODPSO	Meme-Net
$S_{pop}$	The population size	300	300	300	300
$S_{pool}$	Size of the mating pool	15	15	15	15
$P_c$	Crossover rate	0.9	0.9	–	0.9
$P_m$	Mutation rate	0.15	0.15	0.15	0.15
$G_{max}$	The number of iterations	200	200	200	200

**Fig. 2.14** Performance comparisons between FM, Infomap, BGLL, MOGA-net, MODPSO, and MLCD on the GN benchmark networks with different mixing parameters

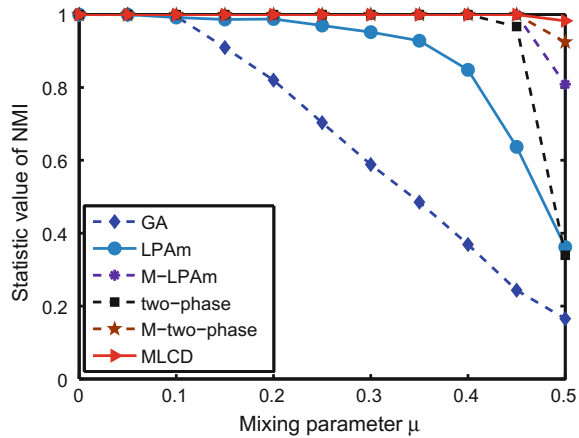


**2.3.7.1 Experiments on Artificial Generated Network**

The extension of GN benchmark network model is introduced in Chap. 1. For GN benchmark networks, the maximum  $NMI$  values averaged over 50 independent runs obtained by MLCD and the compared algorithms are recorded in Figs. 2.14 and 2.15.

As is shown in Figs. 2.14 and 2.15, LPAm, MOGA-Net, FM, Infomap, and BGLL cannot reveal the true community structure of these networks when the mixing parameter  $\mu$  is larger than 0.10, 0.10, 0.30, 0.35, and 0.40, respectively. The proposed algorithm MLCD can detect the true community division of networks when  $\mu \leq 0.45$ . When  $\mu = 0.5$ , the algorithm MLCD uncovers a community structure which is the closest to the true one. These indicate that compared with the classical community detection algorithms, MLCD has remarkable performance on detecting the community structure of the GN benchmark networks.

**Fig. 2.15** Performance comparisons between MLCD, GA, LPAm, M-LPAm two-phase, and M-two-phase on the GN benchmark networks with different mixing parameters



The results in Fig. 2.15 also show that GA cannot detect the true community structure of networks when  $\mu \geq 0.10$ . MLCD, M-two-phase and M-LPAm are able to uncover the true community division of these networks when  $\mu \leq 0.45$ . When  $\mu = 0.5$ , communities detected by the algorithms which have higher level learning strategies are closer to the true one.

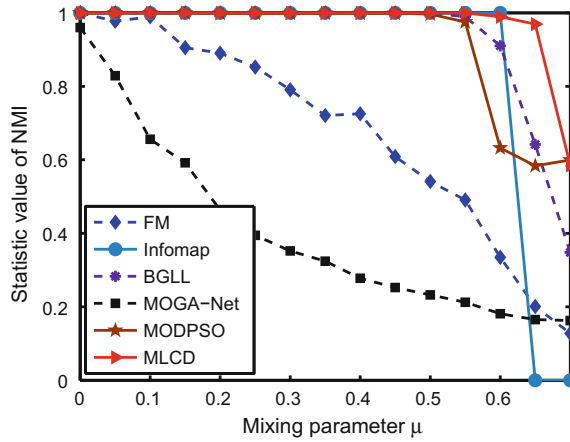
As is shown in Fig. 2.15, GA and LPAm cannot discover the true community structure of networks even if these networks have clear community structures ( $0.1 \leq \mu \leq 0.4$ ). The algorithm M-LPAm, which combines GA and LPAm, can effectively find their true community structures. With  $\mu$  increased, the ratio of the intra-community links to the inter-community links decreases, and thus the community structure of networks becomes increasingly fuzzier. When  $\mu = 0.45$ , two-phase and LPAm cannot find the true community structure of the network, while M-two-phase and M-LPAm can still uncover its true community division. These comparisons demonstrate the effectiveness of the hybrid techniques.

For the LFR benchmark networks, the maximum *NMI* values averaged over 50 independent trials obtained by MLCD and the compared algorithms are recorded in in Figs. 2.16 and 2.17.

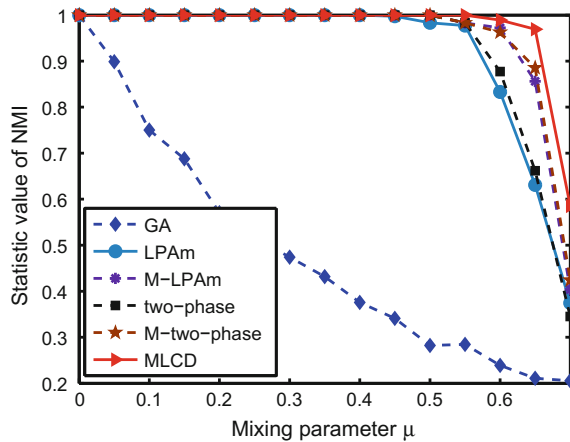
The results in Figs. 2.16 and 2.17 show that except  $\mu = 0.6$ , the partitions obtained by MLCD are the closest to the real ones in the compared algorithms. Therefore, compared with the classical community detection algorithms, the proposed algorithm MLCD has a competitive advantage in discovering the best community divisions of the LFR benchmark networks.

The results in Fig. 2.17 also show that LPAm and two-phase can discover the true community division of networks when  $\mu \leq 0.4$ . With  $\mu$  increased, they become more and more difficult to discover the true community divisions. The hybrid algorithms M-LPAm and M-two-phase can effectively find the true partition of these networks when  $\mu \leq 0.50$ . Even for  $\mu \geq 0.55$ , the values of *NMI* obtained by M-LPAm and M-two-phase are larger than those by LPAm and two-phase, respectively. These

**Fig. 2.16** Performance comparisons between FM, Infomap, BGLL, MOGA-net, MODPSO, and MLCD on the LFR benchmark networks with different mixing parameters



**Fig. 2.17** Performance comparisons between MLCD, GA, LPAm, M-LPAm two-phase and M-two-phase on the LFR benchmark networks with different mixing parameters



indicate that the community divisions revealed by the hybrid techniques M-LPAm and M-two-phase are closer to the true ones than those by LPAm and two-phase.

Compared with the results in Fig. 2.17, it is found that MLCD, M-two-phase and M-LPAm can effectively uncover the true community division of these networks when  $\mu \leq 0.50$ . When  $\mu > 0.50$ , the algorithm which has higher level learning strategies can detect a community division with a higher value of *NMI*. This demonstrates that the higher level learning strategies can effectively escape from the local optima obtained by the lower level learning strategies.

### 2.3.7.2 Experiments on Real-World Networks

In this part, the proposed algorithm MLCD is tested on 12 real-world networks coming from different application fields. These networks include karate, dolphins,

football, polbooks, jazz, elegans, e mail, power, Geom, ca\_grqc, pgp, and Internet. These networks are considered as undirected and unweighted. The basic information of these real-world networks is shown in Table 1.1.

First, the comparisons between MLCD, M-two-phase, M-LPAm and GA are given to illustrate the effectiveness of each level learning strategy. The comparison results are shown in Table 2.6. The results in Table 2.6 show that for the karate, dolphins, football and polbooks networks, GA can hardly discover the optimal community divisions of these networks. The algorithm M-LPAm, which incorporates the node-level learning strategy into GA, has the ability to find the optimal community divisions of these networks. However, the algorithm M-LPAm is easy to fall into poor network partitions. After incorporating the community-level learning strategy into M-LPAm, the algorithm M-two-phase can always find the optimal network partitions for these networks. These demonstrate the effectiveness of the node-level and the community-level learning strategies. The results in Table 2.6 illustrate that for the elegans, e-mail, power, ca\_grqc, Geom, pgp and Internet networks, although the maximal and mean modularity values obtained by M-two-phase are larger than those by M-LPAm, M-two-phase are also easy to trap into poor network partitions. The algorithm MLCD, which incorporates the partition-level learning strategy into M-two-phase, can find the community divisions of these networks with higher modularity values than M-two-phase. This demonstrates the effectiveness of the partition-level learning strategy. The comparison results demonstrate that each level learning strategy can improve the effectiveness of the proposed algorithm.

Moreover, the comparisons between M-LPAm, LPAm, M-two-phase and two-phase are made to illustrate that the hybrid algorithms, M-LPAm and M-two-phase, can enhance both the accuracy and stability of LPAm and two-phase, respectively. The comparison results are recorded in Table 2.6. The results in Table 2.6 show that for most small-scale networks which have less than 5000 nodes, the  $Q_{max}$  and  $Q_{avg}$  values obtained by M-LPAm and M-two-phase are larger than those by LPAm and two-phase, respectively. Meanwhile, the  $Q_{std}$  values obtained by M-LPAm and M-two-phase are smaller than those by LPAm and two-phase, respectively. These indicate that the hybrid techniques, M-LPAm and M-two-phase, can enhance both the accuracy and stability of LPAm and two-phase, respectively, on the small-scale networks. The results in Table 2.6 also illustrate that for these networks which have more than 5000 nodes, the  $Q_{max}$  and  $Q_{avg}$  values obtained by M-LPAm are larger than those by LPAm. However, the  $Q_{max}$  and  $Q_{avg}$  values obtained by M-two-phase are smaller than those by two-phase. This is possible because that with the network size increased, the searching spaces generated by GA are more and more difficult to cover all the preferable ones. Moreover, in the optimization of modularity, the number of local maxima is exponentially growing with the size of networks. Therefore, it is possible that the optimal solution around the search space generated by GA is worse than that by a random way. This can result in this situation in which the community divisions generated by M-two-phase are worse than those by two-phase when they work on the networks with more than 5000 nodes. Note that, after incorporating the partition-level learning strategy into M-two-phase, the community divisions of these networks uncovered by MLCD are better than those by M-two-phase and two-phase.

**Table 2.6** The maximum, average, and standard deviation of modularity values ( $Q_{max}$ ,  $Q_{avg}$  and  $Q_{std}$ ) obtained by GA, LPAm, M-LPAm, two-phase, M-two-phase, MLCD, MOGA-Net, and MODPSO on the real-world networks. Values are recorded under fifty independent trials. “—” indicates that the corresponding algorithm cannot tackle it

$P$	Network	Criterion	GA	LPAm	M-LPAm	two-phase	M-two-phase	MLCD	MOGA-Net	MODPSO
1	Karate	$Q_{max}$	<b>0.4198</b>	0.4052	<b>0.4198</b>	<b>0.4198</b>	<b>0.4198</b>	<b>0.4198</b>	0.4159	<b>0.4198</b>
		$Q_{avg}$	0.4176	0.3564	<b>0.4198</b>	0.4165	<b>0.4198</b>	<b>0.4198</b>	0.3945	0.4182
		$Q_{std}$	0.0047	0.0285	<b>0</b>	0.0077	<b>0</b>	<b>0</b>	0.0089	0.0079
2	Dolphins	$Q_{max}$	0.5258	0.5071	<b>0.5285</b>	<b>0.5285</b>	<b>0.5285</b>	<b>0.5285</b>	0.5034	0.5265
		$Q_{avg}$	0.5138	0.4938	0.5281	0.5232	<b>0.5285</b>	<b>0.5285</b>	0.4584	0.5255
		$Q_{std}$	0.0070	0.0114	0.0013	0.0029	<b>0</b>	<b>0</b>	0.0163	0.0061
3	Polbooks	$Q_{max}$	0.5267	0.5145	<b>0.5272</b>	<b>0.5272</b>	<b>0.5272</b>	<b>0.5272</b>	0.4993	0.5264
		$Q_{avg}$	0.5213	0.4976	0.5271	0.5267	<b>0.5272</b>	<b>0.5272</b>	0.4618	0.5263
		$Q_{std}$	0.0037	0.0158	0.0001	0.0017	<b>0</b>	<b>0</b>	0.0129	0.0007
4	Football	$Q_{max}$	0.4577	0.6032	<b>0.6046</b>	<b>0.6046</b>	<b>0.6046</b>	<b>0.6046</b>	0.4325	<b>0.6046</b>
		$Q_{avg}$	0.4530	0.5777	0.6044	0.6043	<b>0.6046</b>	<b>0.6046</b>	0.3906	0.6038
		$Q_{std}$	0.0210	0.0199	0.0004	0.0009	<b>0</b>	<b>0</b>	0.0179	0.0011
5	Jazz	$Q_{max}$	0.3641	0.4448	<b>0.4451</b>	<b>0.4451</b>	<b>0.4451</b>	<b>0.4451</b>	0.2929	0.4421
		$Q_{avg}$	0.2873	0.4360	0.4450	0.4443	<b>0.4451</b>	<b>0.4451</b>	0.2952	0.4419
		$Q_{std}$	0.0312	0.0098	0.0012	0.0013	0.0001	<b>0</b>	0.0084	0.0001
6	Elegans	$Q_{max}$	0.2750	0.4121	0.4439	0.4498	0.4507	<b>0.4532</b>	0.2977	0.3963
		$Q_{avg}$	0.2139	0.3781	0.4362	0.4411	0.4473	<b>0.4527</b>	0.2805	0.3829
		$Q_{std}$	0.0105	0.0155	0.0042	0.0046	0.0019	<b>0.0004</b>	0.0084	0.0181
7	E-mail	$Q_{max}$	0.3137	0.5415	0.5724	0.5814	0.5820	<b>0.5828</b>	0.3007	0.5193
		$Q_{avg}$	0.2866	0.5349	0.5650	0.5756	0.5805	<b>0.5822</b>	0.2865	0.3493
		$Q_{std}$	0.0090	0.0024	0.0039	0.0036	0.0008	<b>0.0006</b>	0.0075	0.0937
8	Power	$Q_{max}$	0.6345	0.5397	0.7663	0.9382	0.9401	<b>0.9408</b>	0.6914	0.8543
		$Q_{avg}$	0.6208	0.5344	0.7551	0.9370	0.9377	<b>0.9401</b>	0.6864	0.8510
		$Q_{std}$	0.0045	0.0022	0.0051	0.0006	0.0004	<b>0.0002</b>	0.0022	0.0056
9	Ca_grqc	$Q_{max}$	0.4229	0.7146	0.7114	0.8655	0.8635	<b>0.8682</b>	0.6839	—
		$Q_{avg}$	0.4049	0.7074	0.7003	0.8644	0.8617	<b>0.8679</b>	0.6674	—
		$Q_{std}$	0.0080	0.0029	0.0055	0.0008	0.0010	<b>0.0002</b>	0.0063	—
10	Geom	$Q_{max}$	0.5736	0.6625	0.8046	0.8067	0.8052	<b>0.8110</b>	0.6269	0.7007
		$Q_{avg}$	0.5566	0.6494	0.7957	0.8044	0.8018	<b>0.8102</b>	0.6208	0.6954
		$Q_{std}$	0.0077	0.0066	0.0031	0.0016	0.0016	<b>0.0007</b>	0.0030	0.0077
11	Pgp	$Q_{max}$	0.5988	0.7178	0.8129	0.8848	0.8839	<b>0.8867</b>	0.6180	0.3324
		$Q_{avg}$	0.5863	0.7068	0.8063	0.8834	0.8815	<b>0.8865</b>	0.5866	0.3279
		$Q_{std}$	0.0099	0.0070	0.0030	0.0013	0.0011	<b>0.0002</b>	0.0021	0.0090
12	Internet	$Q_{max}$	0.3892	0.4684	0.5839	0.6759	0.6680	<b>0.6782</b>	—	—
		$Q_{avg}$	0.3846	0.4618	0.5658	0.6741	0.6653	<b>0.6770</b>	—	—
		$Q_{std}$	0.0049	0.0041	0.0075	0.0010	0.0022	<b>0.0006</b>	—	—

**Table 2.7** The modularity values obtained by FM, BGLL and Infomap on the real-world networks

$P$	Networks	FM	BGLL	Infomap
1	Karate	0.3807	0.4188	0.4020
2	Dolphins	0.4955	0.5188	0.3978
3	Polbooks	0.5020	0.4986	0.5268
4	Football	0.5773	0.6046	0.6005
5	Jazz	0.4389	0.4431	0.4423
6	Elegans	0.4062	0.4322	0.4168
7	E-mail	0.5116	0.5412	0.5354
8	Power	0.9341	0.7756	0.8302
9	Ca_grqc	0.8122	0.8405	0.8009
10	Geom	0.7767	0.7775	0.7255
11	Pgp	0.8521	0.8604	0.8127
12	Internet	0.6360	0.6464	0.5755

Finally, the comparisons between MLCD, FM, LPAm, BGLL, Infomap, MOGA-Net and MODPSO on the real-world networks are given, and the comparison results are recorded in Tables 2.6 and 2.7. The results show that for all real networks, the modularity values obtained by MLCD are the largest in the compared algorithms. Moreover, except the algorithms of FM, BGLL and Infomap which can stably reveal a network partition, the  $Q_{std}$  values obtained by MLCD are smaller than those by LPAm, MOGA-Net and MODPSO. These indicate that compared with the classical community detection algorithms, MLCD has remarkable performance on solving the community detection problem of networks.

### 2.3.7.3 Experimental Comparisons Between MLCD and Meme-Net

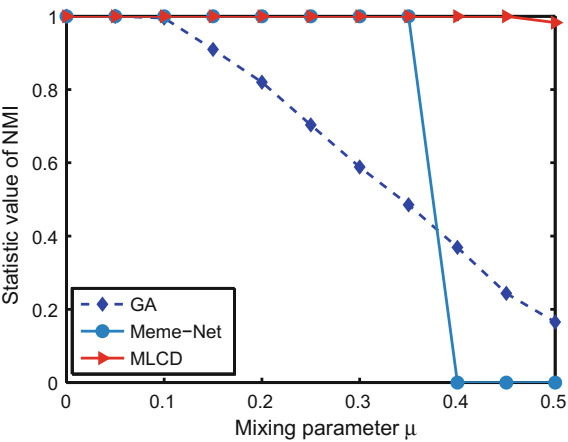
The results of MLCD and Meme-Net on the extension of GN benchmark and 5 real-world networks are recorded in Fig. 2.18 and Tables 2.6 and 2.8. The comparison results show that for the GN benchmark networks, when  $\mu \leq 0.35$ , both Meme-Net and MLCD can find their true network partitions. When  $0.35 < \mu < 0.5$ , Meme-Net cannot detect their community structures while MLCD can find their true network partitions. For the real-world networks, the  $Q_{max}$  and  $Q_{avg}$  values obtained by Meme-Net are smaller than those by MLCD, which means that MLCD has better performance than Meme-Net.

Moreover, the results in Tables 2.6 and 2.8 also show that the  $Q_{std}$  values of these networks obtained by Meme-Net are larger than those by MLCD, which indicates that MLCD has superior performance than Meme-Net in terms of the stability. The box plots of the statistic values of the maximal modularity obtained by GA, Meme-Net and MLCD over 50 runs on the 12 real-world networks are shown in Fig. 2.19.

**Table 2.8** The maximum, average and standard deviation of modularity obtained by Meme-Net on the real-world networks over 50 independent runs

$P$	Networks	$Q_{max}$	$Q_{avg}$	$Q_{std}$
1	Karate	0.4020	0.4020	0
2	Dolphins	0.5185	0.5096	0.0061
3	Polbooks	0.5232	0.5218	0.0031
4	Football	0.6044	0.6023	0.0015
5	Jazz	0.4376	0.4330	0.0011

**Fig. 2.18** Performance comparisons between GA, Meme-Net and MLCD on the GN benchmark networks with different mixing parameters



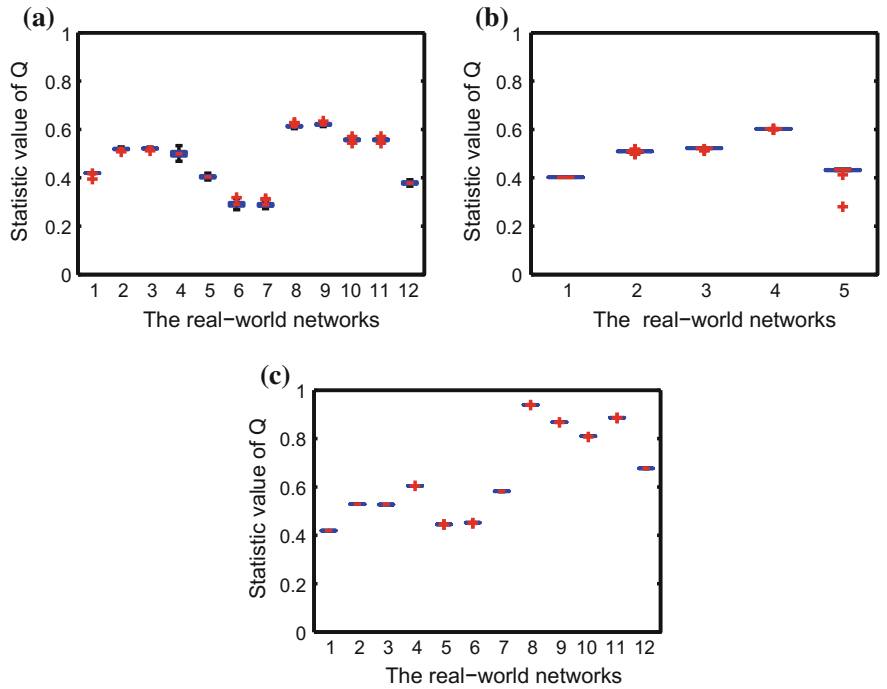
The results in Fig. 2.19 show that the variability of  $Q$  obtained by MLCD is the smallest, which further validates that MLCD is more stable than GA and Meme-Net.

Compared with Meme-Net, the proposed algorithm MLCD has a great advantage in terms of computational complexity. The average elapsed time of Meme-Net and MLCD on real-world networks over 50 times is recorded in Fig. 2.20. The results in Fig. 2.20 clearly show that for the elegans network with 453 nodes, Meme-Net cannot tackle it within a reasonable period of time. However, MLCD can detect communities on the Internet network with 22936 nodes in a reasonable time. This demonstrates that the proposed algorithm has a great advantage over Meme-Net in computing speed.

2.3.8 Conclusions

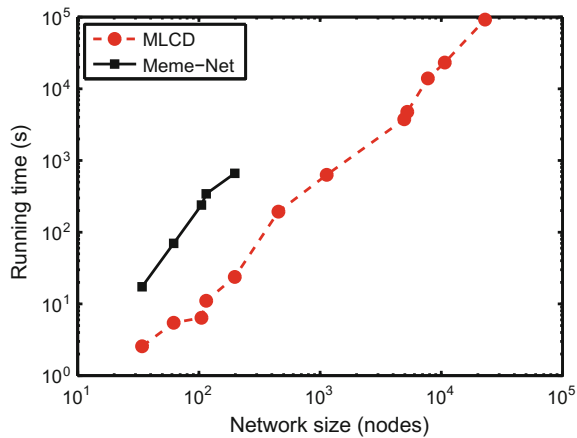
In this section, a fast multi-level learning-based memetic algorithm was proposed to optimize modularity for revealing the potential community structures of complex networks. The proposed algorithm is a hybrid global–local heuristic search methodology





**Fig. 2.19** The box plot of the statistic  $Q$  over the fifty trials on the twelve real-world networks. The box plots are adopted here to explain the distribution of the  $Q$  values obtained by GA, Meme-Net, and MLCD. On each box, the *red line* denotes the median and the symbol *+* represents outliers. The box plot for **a** GA, **b** Meme-Net, and **c** MLCD

**Fig. 2.20** A log-log plot about a comparison of running time between MLCD and Meme-Net in real-world networks with different sizes



and adopts GA as the global search and the proposed multilevel learning strategies as the local search. The operations of GA are redesigned by making full use of the effective connections of networks, and the proposed multilevel learning strategies are devised based on potential knowledge of the node, community and partition structures of networks. The multi-level learning strategies work on the network at nodes, communities and partitions levels, respectively, and they can effectively accelerate the convergence of the proposed algorithm. A large amount of synthetic tests and comparison experiments on artificial and real-world networks demonstrate the effectiveness of the proposed algorithm on uncovering the community structures of networks. They also demonstrate that compared with Meme-Net, the proposed algorithm has superior performance in terms of the quality of the detected community structure, the stability and the computational complexity. The low computational complexity of the proposed algorithm makes it possible to tackle a network with tens of thousands of nodes.

## 2.4 A Swarm Learning-Based Optimization Algorithm for Community Discovery in Large-Scale Networks

<sup>2</sup>In Sects. 2.2 and 2.3, two effective memetic algorithms named Meme-Net and MLCD are introduced for community discovery. They are demonstrated effectively to discover community structures. Here, another computational intelligence algorithm, swarm intelligence optimization, is used for community discovery. Among swarm intelligence optimization, the outstanding paradigm is particle swarm optimization (PSO) [16]. PSO originated from the behavior of social animals, such as fish schooling and birds flocking. PSO optimizes a problem by employing a group of particles. Each particle is a candidate solution to the problem. The candidate solutions are updated with simple rules learnt by the particles. Due to its efficacy and its extremely easy implementation, PSO is prevalent in the optimization field. However, canonical PSO is specially designed for continuous optimization problems.

In this section, a discrete PSO algorithm (GDPSO) for large-scale social network clustering is introduced and GDPSO aims to maximize modularity. The main highlights of GDPSO are as follows. First, the particles velocity and position have been carefully redefined under discrete context so as to make them as easier as possible to encode/decode. Second, for the sake of better exhaustive global searching in the vast searching space, to drive the particles to promising regions, the particle-status-update principles have been thoroughly reconsidered by taking the advantage of the network topology. Third, to avoid being trapped into local optima, a greedy strategy specially designed for the particles to adjust their positions is newly suggested.

---

<sup>2</sup>Acknowledgement: Reprinted from Information Science, 316, Cai, Q., Gong, M., Ma, L., Ruan, S., Yuan, F., Jiao, L., Greedy discrete particle swarm optimization for large-scale social network clustering, 503-516, Copyright(2015), with permission from Elsevier.

### 2.4.1 Greedy Particle Swarm Optimization for Network Community Discovery

The whole framework of the proposed GDPSO algorithm for community discovery is given in Algorithm 8.

---

**Algorithm 8** Framework of the proposed GDPSO algorithm.

---

**Parameters:** particle swarm size  $popsiz$ e, number of iterations  $gmax$ , inertia weight  $\omega$ , learning factors  $c_1$  and  $c_2$ ;

**Input:** network adjacency matrix  $A$ ;

**Output:** best fitness, community structure of the network;

- 1: **Step 1)** Initialize the population: initialize position vectors  $pop[].X$ ; initialize velocity vectors  $pop[].V = 0$ ; initialize the  $Pbest$  vectors  $Pbest[].X = pop[].X$ ;
  - 2: **Step 2)** Evaluate particle fitness  $pop[].fit$ ;
  - 3: **Step 3)** Update the  $Gbest$  particle:  $Gbest = pop[best].X$ ;
  - 4: **Step 4)** Set  $t = 0$ ;
  - 5: **Step 5)** Update particle statuses;
  - 6: **Step 6)** Resorting particle positions:  $resorting(pop[].X)$ ;
  - 7: **Step 7)** Evaluate particles fitness  $pop[].fit$ ;
  - 8: **Step 8)** Update the  $Pbest$  particles: if  $pop[].fit > Pbest[].fit$ , then  $Pbest[].X = pop[].X$ ;
  - 9: **Step 9)** Update the  $Gbest$  particle:  $Gbest = pop[best].X$ ;
  - 10: **Step 10)** If  $t < gmax$ , then  $t++$  and go to **Step 5**); otherwise, stop the algorithm and output.
- 

### 2.4.2 Particle Representation and Initialization

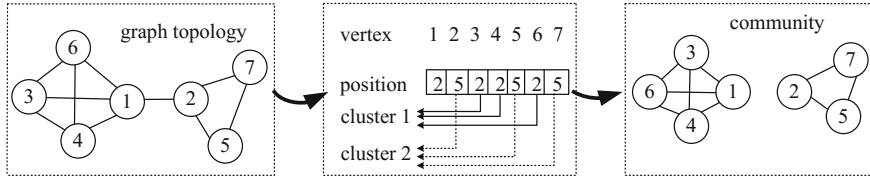
To make the proposed GDPSO algorithm feasible for the network clustering problem, the terms position and velocity under the discrete scenario are redefined.

**Definition 1** (*Position*). The position vector represents a partition of a network. The position permutation of the particle  $i$  is defined as  $X_i = \{x_i^1, x_i^2, \dots, x_i^n\}$ , where  $x_i^j \in [1, n]$  is an integer.

In the above definition,  $x_i^j$  is called a label identifier, which carries the cluster information. If  $x_i^j = x_i^k$ , then nodes  $j$  and  $k$  belong to the same cluster. A graphical illustration of the particle representation is shown in Fig. 2.21.

It can be observed from Fig. 2.21 that the above discrete position definition is straightforward and easy to decode and will reduce the computational complexity, especially in the presence of large-scale networks because the dimensions of the fitness function are the same as the number of nodes in the networks.

**Definition 2** (*Velocity*). The velocity permutation of particle  $i$  is defined as  $V_i = \{v_i^1, v_i^2, \dots, v_i^n\}$ , where  $v_i^j \in \{0, 1\}$ . If  $v_i^j = 1$ , then the corresponding element  $x_i^j$  in the position vector will be changed; otherwise,  $x_i^j$  maintains its original state.



**Fig. 2.21** A schematic illustration of the particle representation

In the canonical version of PSO, there is a threshold  $V_{max}$  that is used to inhibit particles from flying apart because there exists a situation whereby when the speed of a particle is substantial, it will fly out of the boundaries. To define the velocity permutation in the above style, we no longer need this parameter, which is hard to tune.

In the initialization step, the position vectors are initialized using a heuristic method introduced in previous work in [13]. The velocity vectors are initialized as all-zero vectors. The  $pbest$  vectors are initialized in the same manner as the position vectors, and the  $gbest$  vector is set as the best position vector in the original population.

### 2.4.3 Particle-Status-Updating Rules

In the proposed GDPSO algorithm, the particle position and velocity vectors have been redefined in a discrete form; thus, the mathematical operators in the canonical version of PSO no longer fit the discrete context. The mathematical operators have been redefined as follows:

$$V_i = \omega V_i \oplus (c_1 r_1 \times (Pbest_i \ominus X_i) + c_2 r_2 \times (Gbest_i \ominus X_i)) \quad (2.7)$$

$$X_i = X_i \otimes V_i \quad (2.8)$$

In GDPSO, the inertia weight  $\omega$  and the learning factors  $c_1$  and  $c_2$  are set using typical values of 0.7298, 1.4961 and 1.4961. It can be seen that the above equations have the same format as in canonical PSO but that the key components are different. In the next step, these components will be illustrated in detail.

**Definition 3** (*Position  $\ominus$  Position*). Given two position permutations  $P_1 = \{p_1^1, p_1^2, \dots, p_1^n\}$  and  $P_2 = \{p_2^1, p_2^2, \dots, p_2^n\}$ , position  $\ominus$  Position is a velocity vector, i.e.,  $P_1 \ominus P_2 = V = \{v_1, v_2, \dots, v_n\}$ . The element  $v_i$  is defined as

$$\begin{cases} v_i = 0 & \text{if } p_1^i = p_2^i \\ v_i = 1 & \text{if } p_1^i \neq p_2^i \end{cases} \quad (2.9)$$

The inspiration of the above defined operator comes from two aspects. First, from the perspective of swarm intelligence, a particle will adjust its velocity by learning from its neighbors. The learning process is actually a comparison between the positions; in other words, two position vectors generate a velocity vector. Second, from the viewpoint of graph theory, two position vectors represent two types of network community structures. The defined  $\ominus$  operation actually reflects the difference between two network structures.

**Definition 4** (*Coefficient  $\times$  Velocity*). The operator  $\times$  is the same as the basic arithmetical multiplication operator. For instance, given a coefficient  $c \cdot r = 1.3$  and given a velocity vector  $V = \{1, 0, 1, 1, 0\}$ ,  $c \cdot r \times V = \{1.3, 0, 1.3, 1.3, 0\}$ .

**Definition 5** (*Velocity  $\oplus$  Velocity*). Velocity  $\oplus$  Velocity equals a velocity as well. Given two velocity vectors  $V_1 = \{v_1^1, v_1^2, \dots, v_1^n\}$  and  $V_2 = \{v_2^1, v_2^2, \dots, v_2^n\}$ ,  $V_1 + V_2 = V_3 = \{v_3^1, v_3^2, \dots, v_3^n\}$ . The element  $v_3^i$  is defined as

$$\begin{cases} v_3^i = 1 & \text{if } v_1^i + v_2^i \geq 1 \\ v_3^i = 0 & \text{if } v_1^i + v_2^i < 1 \end{cases} \quad (2.10)$$

The definition of the operator  $\oplus$  is straight forward, and the operation is easy to perform. Moreover, it can always make sure that the velocity is binary coded, which is easier for the position to work with.

**Definition 6** (*Position  $\otimes$  Velocity*). The operator  $\otimes$  is the key component. A particle will update its position according to a new velocity, i.e., Position  $\otimes$  Velocity generates a new position. A good operator  $\otimes$  should drive a particle to a promising region. Given a position  $P_{old} = \{p_{old}^1, p_{old}^2, \dots, p_{old}^n\}$  and a velocity  $V = \{v_1, v_2, \dots, v_n\}$ ,  $P_{old} \otimes V = P_{new} = \{p_{new}^1, p_{new}^2, \dots, p_{new}^n\}$ . The element of  $P_{new}$  is defined as follows:

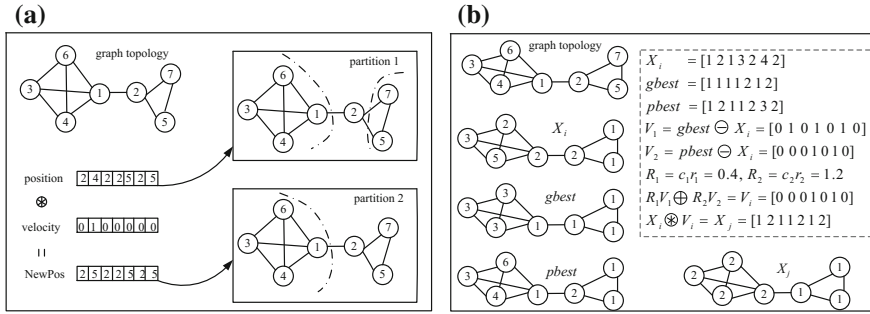
$$\begin{cases} p_{new}^i = p_{old}^i & \text{if } v_i = 0 \\ p_{new}^i = \arg \max_j \Delta Q(p_{old}^i, j \mid j \in L_i) & \text{if } v_i = 1 \end{cases} \quad (2.11)$$

where  $L_i = \{l_1, l_2, \dots, l_k\}$  is the set of label identifiers of vertex  $i$ 's neighbors. The  $\Delta Q$  is calculated using the following equation:

$$\Delta Q(p_{old}^i, j \mid j \in L_i) = \text{fit}(P_{old} \mid p_{old}^i \leftarrow j) - \text{fit}(P_{old}) \quad (2.12)$$

Equation 2.11 can be regarded as a greedy local search strategy because a particle updates its position by choosing the label identifier that can generate the largest fitness increment. A graphical Fig. 2.22b gives a simple illustration of how a particle updates its status.

In Fig. 2.22b,  $X_i$  represents the current particle's position vector,  $V_1$  and  $V_2$  are intermediate velocity vectors determined using Eq. 2.9, and  $V_i$  is the new velocity calculated using Eq. 2.10. Under the guidance of the velocity  $V_i$ , the current particle updates its current position  $X_i$  using Eq. 2.11, and thus, a new position  $X_j$  is obtained.



**Fig. 2.22** A schematic illustration of **a** the  $\otimes$  operator and **b** the Particle-status-updating rules

It can be observed from the above description that (1) the proposed GDPSO algorithm has a concise framework, (2) the newly defined particle representation is direct and easy to decode, and (3) the redefined updating rules are easy to realize. All these merits make this advanced algorithm capable of addressing large-scale networks.

### 2.4.4 Particle Position Reordering

In the proposed algorithm, to avoid unnecessary computing, a particle position reordering operator is designed. Given that  $X = \{x_1, x_2, \dots, x_n\}$  is a position vector of a particle, the particle position reordering operator acts on  $X$  and outputs a new vector  $X'$ . The operator reorders the elements in  $X$  with a starting value of 1. The pseudocode of the reordering operator is given in Algorithm 9.

---

#### Algorithm 9 Pseudocode of the reordering operator.

---

**Input:** an integer permutation  $X = \{x_1, x_2, \dots, x_n\}$ ;

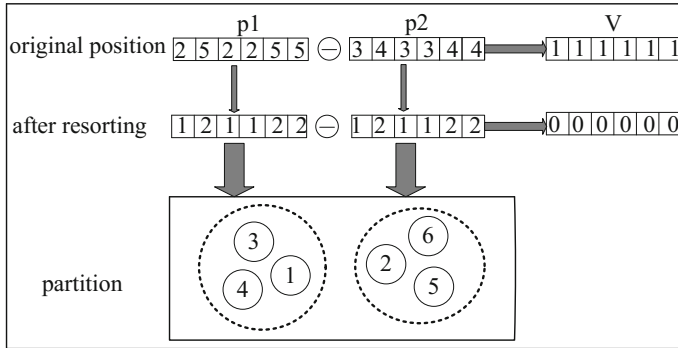
**Output:** reordered permutation  $X' = \{x'_1, x'_2, \dots, x'_n\}$ ;

```

1: set  $counter = 1, X' \leftarrow X$ ;
2: for  $i = 1; i \leq n; i++$  do
3:   if  $x'_i \neq -1$  then
4:     for  $j = i + 1; j \leq n; j++$  do
5:       if  $x'_j = x'_i$  then
6:          $x_j = counter, x'_j = -1$ ;
7:       end if
8:     end for
9:      $x'_i = -1, x_i = counter, counter++$ ;
10:  end if
11: end for
12:  $X' \leftarrow X$ ;

```

---



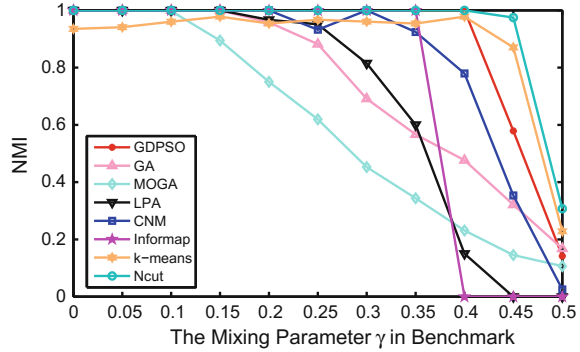
**Fig. 2.23** A schematic illustration of the reordering operation

Figure 2.23 gives a graphical illustration of the reordering operator. As illustrated in Fig. 2.23,  $p1$  and  $p2$  are structure-equivalent (they correspond to the same partition). If we do not design the resorting operation, then according to Eq. 2.9, a non-zero velocity vector would be obtained, and according to Eq. 2.11, a new position would need to be calculated, which would require computing time. However, if we design the operation so that after resorting,  $p1$  and  $p2$  are the same, then according to Eq. 2.9, the obtained velocity will be a zero vector; thus, it would not need to calculate the new position, resulting in reduced computing time.

### 2.4.5 Experimental Results

Here, GDPSO is tested on both artificial generated networks and real-world networks and is compared with several state-of-the-art community detection algorithms: GA [21], MOGA [22], LPA [1], CNM [7] and Infomap [24]. To enable a fair comparison, in the experiments, the objective function used in the GA is replaced by the modularity, and the modularity is used to choose the ultimate solution from the Pareto front for the MOGA. The population size and the maximum number of iterations for the GDPSO, GA, and MOGA are set to 100, the crossover and mutation probabilities for the GA and the MOGA are set to 0.9 and 0.1, respectively. The LPA and Infomap are two iterative methods; the maximum number of iteration is set to 5 and 100, respectively. In addition, a comparison is performed with two canonical data clustering methods: the k-means method proposed in [15] and the normalized cut (Ncut) approach proposed in [26].

**Fig. 2.24** The experimental results from the benchmark networks



### 2.4.5.1 Results on Artificial Generated Network

All the algorithms are performed on the extension of GN benchmark networks. Figure 2.24 shows the statistical results averaged over 30 runs for different algorithms.

As shown in Fig. 2.24, when the mixing parameter is no larger than 0.15, all the methods, except the MOGA and k-means methods, can determine the true community structures of the networks ( $NMI = 1$ ). As the mixing parameter increases, the GA, LPA and CNM fail to discover the correct partitions. For  $\gamma = 0.4$ , our proposed method and Ncut still obtain  $NMI = 1$ , whereas the other methods cannot. When  $\gamma$  is larger than 0.4, from the curve, it can be observed that the  $NMI$  values quickly decrease and that our method fails to detect the ground truths. One reason that can account for this is that when  $\gamma$  is larger than 0.4, the community structure of the network is rather vague, that is, there is no community structure. Moreover, optimizing the modularity has been proven to be NP hard. Therefore, it is very hard for an optimization method to uncover any communities under this scenario.

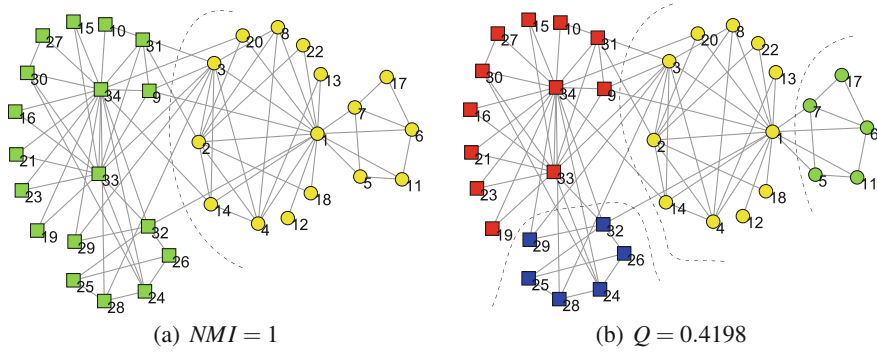
The experiments on the computer-generated benchmark networks prove that the proposed GDP SO framework for network clustering is feasible. The proposed algorithm is capable of uncovering community structures in social networks. Although from the curves, Ncut seems to perform the best; however, Ncut needs to specify the clusters in advance, which is very inconvenient.

### 2.4.5.2 Results on Real-World Networks

Here, the performance of the GDP SO is tested on four small-scale networks and on four large-scale networks: Karate, dolphin, football, SFI, e-mail, netscience, power grid, PGP. The parameters of each network are listed in Table 1.1. For each social network, each algorithm is run 30 times.

First the algorithms are tested on the four small-scale networks. A statistical analysis using the Wilcoxon rank sum test is performed. In the right-tailed rank sum test experiments, the significance level is set to 0.05 with the null hypothesis that





**Fig. 2.25** Community structure of the Karate network. **a** Ground truth. **b** Detected structure

the two independent samples (the two modularity sets) come from distributions with equal means. Tables 2.9 and 2.10 show the experimental results and the hypothesis test  $p$ -values.

From the two tables, it is can noted that GDPSO performs remarkably well in terms of the modularity values and the computational time. The small  $p$ -values indicate that GDPSO is substantially better than the compared methods. GDPSO framework possesses excellent global exploration ability. On the four small-scale networks, GDPSO outperforms the other methods, except for the GA. The GA optimizes the same objective as does the GDPSO. From the experiments, it is shown that the GA and the GDPSO perform well, but the GDPSO converges faster. In the next step, the community structures discovered by the GDPSO on the four small-scale networks will be analyzed.

The Karate network is a network of relations between 34 members of a karate club. Figure 2.25 shows the real community structure and the detected structure of the network. It can be observed that GDPSO has discovered four clusters, which are the subdivisions of the real ones. This phenomenon also occurs in the dolphin network. Figure 2.26 clearly displays the discovered structure of the dolphin network. GDPSO divides one of the two clusters in the real structure into four smaller sections. It cannot determine if this division makes sense to the dolphins; from the perspective of optimization, GDPSO simply finds the best objective function value.

In football network, although GDPSO obtains the largest  $Q$  value, it could find that several vertices, such as the numbers 29, 43, 37, 81, 60, 91 and 63, are misclassified. This is mainly caused by the nuances in the scheduling of games.

Figure 2.27 exhibits the discovered communities in SFI network. From the figure, it could notice that the GDPSO splits the network into eight strong communities, with divisions running principally along disciplinary lines. The sub-communities that are subdivisions of the original three large-scale groups are centered around the interests of leading members.

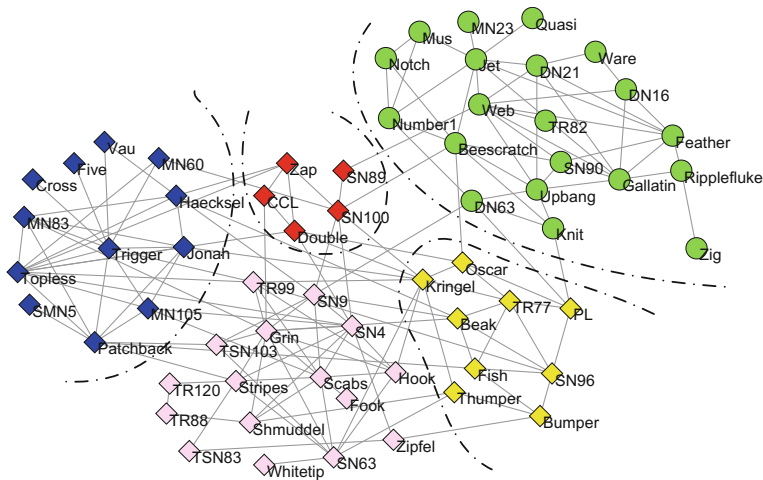
Experiments on the four small-scale social networks indicate that the proposed GDPSO algorithm is effective. The algorithm exhibits an outstanding search abil-

**Table 2.9** Experimental results on the Karate and dolphin networks. The parameter  $k$  used in k-means and Neut is set to 4 for the Karate network and 5 for the dolphin network

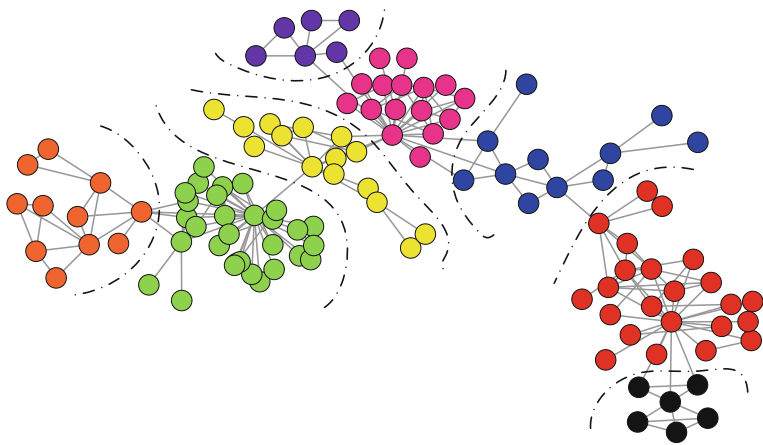
Network	Karate					Dolphin				
Index	$Q_{max}$	$Q_{avg}$	$T_{avg}$	$I_{avg}$	$p$ -value	$Q_{max}$	$Q_{avg}$	$T_{avg}$	$I_{avg}$	$p$ -value
GDP50	0.4198	0.4198	2.5800E-2	0.6881	×	0.5285	0.5284	5.8621E-2	0.8935	×
GA	0.4198	0.4198	1.9871E-1	0.6873	0.5000	0.5285	0.5280	2.3868	0.5892	0.1591
MOGA	0.4198	0.4160	1.3975	1	0.0000	0.5085	0.4098	3.1713	0.9442	0.0000
LPA	0.4151	0.3264	3.3133E-3	0.6623	0.0000	0.5258	0.4964	3.8011E-3	0.6194	0.0000
CNM	0.3800	0.3800	5.0700E-2	0.6920	0.0000	0.4950	0.4950	2.2517E-2	0.5730	0.0000
Informap	0.4020	0.4020	2.1331E-1	0.6995	0.0000	0.5247	0.5247	3.7121E-1	0.4662	0.0000
K-means	0.1429	0.0351	2.6121E-2	0.6059	0.0000	0.4796	0.3787	4.5717E-2	0.4282	0.0000
Ncut	0.4198	0.4198	4.2272E-3	0.6873	0.5000	0.5068	0.5047	8.2015E-3	0.5084	0.0000

**Table 2.10** The experimental results on the football and SFI networks. The parameter  $k$  used in k-means and in Ncut is set to 12 for the football network and to 7 for the SFI network

Network Index	Football					SFI				
	$Q_{max}$	$Q_{avg}$	$T_{avg}$	$I_{avg}$	p-value	$Q_{max}$	$Q_{avg}$	$T_{avg}$	p-value	
GDP SO	0.6046	0.6041	9.4210E-2	0.8889	×	0.7506	0.7449	4.5217E-2	×	
GA	0.5929	0.5829	4.9276	0.8227	0.0000	0.7506	0.7505	6.8536	1.0000	
MOGA	0.5280	0.5173	4.1731	0.7883	0.0000	0.7430	0.7323	1.9742	0.0000	
LPA	0.6030	0.5848	8.2001E-3	0.8735	0.0000	0.7341	0.7095	5.3731E-3	0.0000	
CNM	0.5770	0.5770	1.1292E-1	0.7620	0.0000	0.7335	0.7335	4.5661E-2	0.0000	
Informap	0.6005	0.6005	1.2133	0.9242	0.0000	0.7334	0.7334	7.7563E-1	0.0000	
k-means	0.5783	0.5130	5.4805E-2	0.8512	0.0000	0.4376	0.2780	3.4512E-2	0.0000	
Ncut	0.6031	0.6007	8.8133E-3	0.9233	0.0000	0.7478	0.7470	9.5000E-3	0.9990	



**Fig. 2.26** Our detected community structure of the dolphin network ( $Q = 0.5285$ )



**Fig. 2.27** Community structure of the SFI network ( $Q = 0.7506$ )

ity when addressing moderate-scale optimization problems. To further verify its optimization ability, next the algorithm is performed on four large-scale networks. Tables 2.11 and 2.12 list the statistical results.

The rank sum test results indicate that GDP SO is superior to the GA, MOGA, LPA, and k-means algorithm for the four large networks and that it outperforms Informap for the four big networks, except for the e-mail network. The CNM and Informap methods are two deterministic methods. For the four large networks, the CNM seems to perform the best from the angle of modularity, LPA is the fastest, and the k-means algorithm performs poorly. The MOGA, k-means and Ncut methods can hardly

handle large-scale networks. For the PGP network, the MOGA cannot provide output after four hours, and the k-means and Ncut methods run out of computer memory. Through GDPSO, good modularity values could be found within a reasonable amount of time. From the tables, it can be observed that our method still falls into local optimum solutions; for example, for the netscience and power grid networks, the modularity values obtained by the GDPSO are smaller than those obtained by the CNM. One reason that can account for this is the designed greedy mechanism-based particle-position-update principle possibly causing prematurity. In the next subsection, the particle-position-update principle will further be discussed.

### 2.4.6 Additional Discussion on GDPSO

For the proposed GDPSO algorithm, the population size and the maximum number of iterations affect the performance of the algorithm. Tables 2.13 and 2.14 show the influence of these two parameters on the performance of the algorithm.

The two parameters  $gmax$  and  $popsiz$  are normally set empirically. Small values of the two parameters may not result in convergence, whereas large values will require substantial amounts of computing time. From the above experiments, as shown in Tables 2.13 and in 2.14, it could observe that the impact of the two parameters on the performance of the algorithm is not salient. To obtain a tradeoff between the convergence and the computation time,  $gmax = popsiz = 100$  is set as the default configuration of GDPSO.

The experiments on small networks demonstrate the effectiveness of GDPSO algorithm; however, its performance on large networks remains unsatisfactory. There are two key reasons why the GDPSO obtains unsatisfactory results. On the one hand, for the large scale of networks clustering problem, the diversity preservation is insufficient. However, it needs to find a better strategy that can both preserve diversity and ensure fast convergence. On the other hand, the designed particle-position-update principle (Eq. 2.11) is based on a simple greedy mechanism, which may lead to prematurity.

Equation 2.11 is the key component of GDPSO. It updates the label identifier of a vertex with the neighbor identifier that generates the largest increase in the objective function value. From the perspective of graph theory, it is natural to update the vertex identifier with the two different methods shown in Fig. 2.28a.

The two different label-identifier-update principles shown in Fig. 2.28b, c make sense from the perspective of sociology. For example, the maxD principle is in accordance with the social phenomenon whereby people prefer to learn from or simply imitate the one who is the most attractive and talented amongst their friends, and the dominated principle complies with the social phenomenon that one would like to follow the state that is kept by the majority of his or her friends. Table 2.15 shows a comparison of the three update principles.

From the table, it can observe that the maxD and the dominated principles are computationally faster than that of  $\Delta Q$ , and the maxD principle is the fastest. How-

**Table 2.11** The experimental results for the e-mail and netscience networks. The parameter  $k$  used in k-means and Neut is set to 4 for the e-mail network and to 100 for the netscience network

Network	E-mail				Netscience			
	$Q_{max}$	$Q_{avg}$	$T_{avg}$	$p$ -value	$Q_{max}$	$Q_{avg}$	$T_{avg}$	$p$ -value
GDP SO	0.5487	0.4783	2.4717E+1	×	0.9540	0.9512	3.5212E+1	×
GA	0.3647	0.3500	3.1015E+2	0.0000	0.9086	0.9003	1.9967E+2	0.0000
MOGA	0.3283	0.3037	5.3716E+2	0.0000	0.8916	0.8810	7.0681E+1	0.0000
LPA	0.2055	0.0070	6.5033E-2	0.0000	0.9266	0.9202	3.9117E-2	0.0000
CNM	0.4985	0.4985	1.5255E+1	1.0000	0.9555	0.9555	2.0414E+1	1.0000
Informap	0.5355	0.5355	2.3200	1.0000	0.9252	0.9252	1.5577	0.0000
K-means	0.3681	0.3600	5.2580	0.0000	0.6510	0.6359	5.6187E+1	0.0000
Neut	0.4841	0.4749	4.7000E-2	0.2384	0.9293	0.9268	3.9302E-1	0.0000

**Table 2.12** The experimental results for the power grid and PGP networks. The parameter  $k$  used in k-means and Neut is set to 200 for the power grid network and to 300 for the PGP network

Network Index	Power grid				PGP			
	$Q_{max}$	$Q_{avg}$	$T_{avg}$	$p$ -value	$Q_{max}$	$Q_{avg}$	$T_{avg}$	$p$ -value
GDPSO	0.8382	0.8368	4.7818E+2	×	0.8050	0.8013	6.3609E+2	×
GA	0.7161	0.7124	3.4485E+3	0.0000	0.6576	0.6473	3.7798E+4	0.0000
MOGA	0.7035	0.6949	4.9177E+3	0.0000	—	—	—	—
LPA	0.7602	0.7476	1.7373E-1	0.0000	0.7949	0.7845	7.0391E-1	0.0000
CNM	0.9229	0.9229	4.9121E+2	1.0000	0.8481	0.8481	7.8562E+3	1.0000
Informap	0.8140	0.8140	6.3715	0.0000	0.7777	0.7777	1.2313E+1	0.0000
K-means	Out of Memory				Out of Memory			
Neut	0.8875	0.8866	4.4442	1.0000	Out of Memory			

**Table 2.13** The influence of the population size on the performance of the algorithm. The modularity values over 30 independent runs are recorded in this table

(Gmax = 100)	PopsiZe = 20		PopsiZe = 60		PopsiZe = 100		PopsiZe = 140		PopsiZe = 180	
	Q <sub>max</sub>	Q <sub>avg</sub>	Q <sub>max</sub>	Q <sub>avg</sub>	Q <sub>max</sub>	Q <sub>avg</sub>	Q <sub>max</sub>	Q <sub>avg</sub>	Q <sub>max</sub>	Q <sub>avg</sub>
Karate	0.4198	0.4085	0.4198	0.4174	0.4198	0.4198	0.4198	0.4198	0.4198	0.4198
Dolphin	0.5269	0.5265	0.5277	0.5265	0.5285	0.5284	0.5285	0.5284	0.5285	0.5285
Football	0.6046	0.6027	0.6046	0.6035	0.6046	0.6041	0.6046	0.6041	0.6046	0.6043
SFI	0.7484	0.7408	0.7484	0.7439	0.7506	0.7449	0.7506	0.7456	0.7506	0.7453
E-mail	0.5361	0.4162	0.5384	0.4632	0.5487	0.4783	0.5476	0.4819	0.5437	0.4864
NetScience	0.9323	0.9275	0.9535	0.9492	0.9540	0.9512	0.9537	0.9507	0.9547	0.9511
Power grid	0.7685	0.7627	0.8377	0.8311	0.8382	0.8368	0.8383	0.8357	0.8401	0.8372
PGP	0.8028	0.7974	0.8047	0.8002	0.8050	0.8013	0.8047	0.8011	0.8053	0.8017

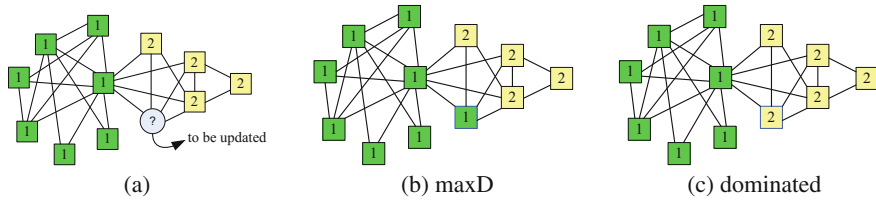


**Table 2.14** The influence of the maximum iteration number  $gmax$  on the performance of the algorithm. The modularity values over 30 independent runs are recorded in this table

$(Popsiz = 100)$	$Gmax = 50$		$Gmax = 100$		$Gmax = 150$		$Gmax = 200$		$Gmax = 250$	
	$Q_{max}$	$Q_{avg}$	$Q_{max}$	$Q_{avg}$	$Q_{max}$	$Q_{avg}$	$Q_{max}$	$Q_{avg}$	$Q_{max}$	$Q_{avg}$
Karate	0.4198	0.4180	0.4198	0.4198	0.4198	0.4198	0.4198	0.4198	0.4198	0.4198
Dolphin	0.5277	0.5267	0.5285	0.5284	0.5285	0.5285	0.5285	0.5285	0.5285	0.5285
Football	0.6046	0.6040	0.6046	0.6041	0.6046	0.6037	0.6046	0.6039	0.6046	0.6043
SFI	0.7487	0.7447	0.7506	0.7449	0.7506	0.7454	0.7506	0.7449	0.7506	0.7441
E-mail	0.5157	0.4731	0.5487	0.4783	0.5237	0.4687	0.5431	0.4864	0.5454	0.4997
Netscience	0.9531	0.9482	0.9540	0.9512	0.9537	0.9507	0.9547	0.9511	0.9507	0.9491
Power grid	0.8365	0.8331	0.8382	0.8368	0.8383	0.8357	0.8401	0.8372	0.8404	0.8380
PGP	0.8041	0.8012	0.8050	0.8013	0.8050	0.7991	0.8053	0.8031	0.8053	0.8008

**Table 2.15** Comparison of different particle-position-update principles

Algorithm index	GDPSO ( $\Delta Q$ )				GDPSO (maxD)				GDPSO (dominated)			
	$Q_{max}$	$Q_{avg}$	$T_{avg}$		$Q_{max}$	$Q_{avg}$	$T_{avg}$	$p$ -value	$Q_{max}$	$Q_{avg}$	$T_{avg}$	$p$ -value
Karate	0.4198	0.4198	2.5800E-2		0.4156	0.4080	1.7724E-3	0.0000	0.4156	0.4089	3.3976E-3	0.0000
Dolphin	0.5285	0.5284	5.8621E-2		0.5268	0.5265	4.4154E-3	0.0000	0.5268	0.5265	2.8781E-2	0.0000
Football	0.6046	0.6041	9.4210E-2		0.6046	0.6038	1.0939E-2	0.3349	0.6046	0.6038	3.6844E-2	0.5000
SFI	0.7506	0.7449	4.5217E-2		0.7484	0.7433	8.7936E-3	0.0153	0.7497	0.7458	2.3563E-1	0.8812
E-mail	0.5487	0.4783	2.4717E+1		0.4941	0.3108	4.3074E-1	0.0000	0.3670	0.2453	1.1583	0.0000
Netscience	0.9540	0.9512	3.5212E+1		0.9522	0.9478	2.2385	0.0000	0.9334	0.9296	1.2988E+1	0.0000
Power grid	0.8382	0.8368	4.7818E+2		0.8382	0.8368	4.5868E+1	0.5000	0.8145	0.8116	2.0348E+2	0.0000
PGP	0.8050	0.8013	6.3609E+2		0.8021	0.7977	1.7526E+1	0.0000	0.8145	0.8093	1.0025E+3	1.0000



**Fig. 2.28** **a** Different vertex-label-identifier-update principles. **b** Choose the identifier of the neighboring vertex that has the largest degree. **c** Choose the dominated identifier from the neighbor vertices

ever, from the modularity index perspective, the maxD and the dominated principles all fall into local optima solutions. The rank sum test results indicate that the  $\Delta Q$  mechanism works better than do the other two particle-position-update principles.

### 2.4.7 Conclusions

In this section, a discrete PSO method designed to discover community structures in social networks was introduced. In GDPSO, first the particle position and velocity are redefined in a discrete form and subsequently redesigned the particle-update rules based on the network topology; consequently, a discrete PSO framework was established. When applying GDPSO to solve the network clustering problem, because the scale of a real-world social network is especially large most of the time, to alleviate prematurity, a greedy local-search-based mechanism was specially designed for the particle-position-update rule. Experiments on both synthetic and real-world networks demonstrated that the proposed algorithm for network clustering is effective and promising.

## References

1. Bagrow, J.P., Boltt, E.M.: Local method for detecting communities. *Phys. Rev. E* **72**(4), 046,108 (2005)
2. Barber, M.J., Clark, J.W.: Detecting network communities by propagating labels under constraints. *Phys. Rev. E* **80**(2), 026,129 (2009)
3. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. *J. Stat. Mech.: Theory Exp.* **2008**(10), P10,008 (2008)
4. Blum, C., Puchinger, J., Raidl, G.R., Roli, A.: Hybrid metaheuristics in combinatorial optimization: a survey. *Appl. Soft Comp.* **11**(6), 4135–4151 (2011)
5. Cai, Q., Gong, M., Ma, L., Ruan, S., Yuan, F., Jiao, L.: Greedy discrete particle swarm optimization for large-scale social network clustering. *Inf. Sci.* **316**, 503–516 (2015)
6. Clauset, A., Newman, M.E., Moore, C.: Finding community structure in very large networks. *Phys. Rev. E* **70**(6), 066,111 (2004)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: *Clifford stein. Introduction to algorithms* (2001)

8. Danon, L., Diaz-Guilera, A., Duch, J., Arenas, A.: Comparing community structure identification. *J. Stat. Mech.: Theory Exp.* **2005**(09), P09,008 (2005)
9. Fortunato, S., Barthelemy, M.: Resolution limit in community detection. *Proc. Nat. Acad. Sci.* **104**(1), 36–41 (2007)
10. Gach, O., Hao, J.K.: A memetic algorithm for community detection in complex networks. In: *International Conference on Parallel Problem Solving from Nature*, pp. 327–336. Springer (2012)
11. Gong, M., Cai, Q., Chen, X., Ma, L.: Complex network clustering by multiobjective discrete particle swarm optimization based on decomposition. *IEEE Trans. Evol. Comput.* **18**(1), 82–97 (2014)
12. Gong, M., Cai, Q., Li, Y., Ma, J.: An improved memetic algorithm for community detection in complex networks. In: *2012 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE (2012)
13. Gong, M., Fu, B., Jiao, L., Du, H.: Memetic algorithm for community detection in networks. *Phys. Rev. E* **84**(5), 056,101 (2011)
14. Gong, M., Ma, L., Zhang, Q., Jiao, L.: Community detection in networks by using multiobjective evolutionary algorithm with decomposition. *Physica A: Stat. Mech. Appl.* **391**(15), 4050–4060 (2012)
15. Jain, B.J., Obermayer, K.: Elkan's k-means algorithm for graphs. In: *Mexican International Conference on Artificial Intelligence*, pp. 22–32 (2010)
16. Kennedy, J.: Particle swarm optimization. In: *Encyclopedia of Machine Learning*, pp. 760–766. Springer (2011)
17. Li, Z., Zhang, S., Wang, R.S., Zhang, X.S., Chen, L.: Quantitative function for community detection. *Phys. Rev. E* **77**(3), 036,109 (March 2008)
18. Ma, L., Gong, M., Liu, J., Cai, Q., Jiao, L.: Multi-level learning based memetic algorithm for community detection. *Appl. Soft Comput.* **19**, 121–133 (2014)
19. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. *Phys. Rev. E* **69**(2), 026,113 (2004)
20. Ong, Y.S., Lim, M.H., Chen, X.: Memetic computation—past, present & future [research frontier]. *IEEE Comput. Intell. Mag.* **5**(2), 24–31 (2010)
21. Pizzuti, C.: Ga-net: a genetic algorithm for community detection in social networks. In: *International Conference on Parallel Problem Solving from Nature*, pp. 1081–1090 (2008)
22. Pizzuti, C.: A multiobjective genetic algorithm to find communities in complex networks. *IEEE Trans. Evol. Comput.* **16**(3), 418–430 (2012)
23. Rosvall, M., Axelsson, D., Bergstrom, C.T.: The map equation. *Eur. Phys. J. Spec. Top.* **178**(1), 13–23 (2009)
24. Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure. *Proc. Nat. Acad. Sci.* **105**(4), 1118–1123 (2008)
25. Shang, R., Bai, J., Jiao, L., Jin, C.: Community detection based on modularity and an improved genetic algorithm. *Phys. A: Stat. Mech. Appl.* **392**(5), 1215–1231 (2013)
26. Shi, J., Malik, J.: Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **22**(8), 888–905 (2000)

Computational Intelligence for Network Structure  
Analytics

Gong, M.; Cai, Q.; Ma, L.; Wang, S.; Lei, Y.

2017, XI, 283 p. 159 illus., 140 illus. in color., Hardcover

ISBN: 978-981-10-4557-8