Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-009-9080-2

An autonomy-oriented computing approach to community mining in distributed and dynamic networks

Bo Yang · Jiming Liu · Dayou Liu

Abstract A network community refers to a special type of network structure that contains a group of nodes connected based on certain relationships or similar properties. Our ability to mine communities hidden inside networks will readily enable us to effectively understand and exploit such networks. So far, various methods and algorithms have been developed to perform the task of community mining, where it is often required that the networks are processed in a centralized manner, and their structures will not dynamically change. However, in the real world, many applications involve distributed and dynamically evolving networks, in which resources and controls are not only decentralized but also updated frequently. It would be difficult for the existing methods to deal with these types of networks since their global topological representations are either not available or too hard to obtain due to their huge size, decentralization, and/or dynamic updates. The aim of our work is to address the problem of mining communities from a distributed and dynamic network. It differs from the previous ones in that here we introduce the notion of self-organizing agent networks, and provide an autonomy-oriented computing (AOC) approach to distributed and incremental mining of network communities. The AOC-based method utilizes reactive agents that can collectively detect and update community structures in a distributed and dynamically evolving network, based only on their local views and interactions. While providing detailed formulations, we present the results of our systematic validations using real-world benchmark networks as

A preliminary, brief version of this work was presented as a regular paper at the 1st IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07), MIT, July 9–11, 2007.

B. Yang · D. Liu
College of Computer Science and Technology, Jilin University, Changchun 130012, China
e-mail: ybo@jlu.edu.cn

D. Liu
e-mail: dyliu@jlu.edu.cn

J. Liu (✉)
Department of Computer Science, Hong Kong Baptist University, Kowloon, Hong Kong
e-mail: jiming@comp.hkbu.edu.hk

Published online: 05 March 2009

well as synthetic networks that include a distributed intelligent Portable Digital Assistant (iPDA) network example.

Keywords Social networks · Distributed networks · Community mining · Autonomy-oriented computing · Self-organization · Agent networks · Multi-agent systems

1 Introduction

A *network community mining problem* (NCMP) refers to the problem of finding all communities from a given network, within which the links are dense but between which they are sparse [18], as illustrated in Fig. 1. A wide variety of applications can be formulated as NCMPs, such as agent scheduling in distributed systems (Fig. 1), social network analysis [7,19,20,23], biological network analysis [27], Web pages clustering [5,11,21], and image segmentation [25]. Social network communities can be social groups within which people share some common interests and have more contacts than those outside. Communities in a protein-protein interaction network can indicate groups of proteins with similar functions. On the other hand, communities in the World Wide Web (WWW) may consist of collections of Web pages related to certain common topics.

So far, various methods and algorithms for solving the NCMP have been developed, which can be generally divided into three main categories:

(1) *Bisection methods* spectral method [4,22,25], the Kernighan-Lin algorithm [10], the Wu-Huberman algorithm [29], and the FEC algorithm [30].

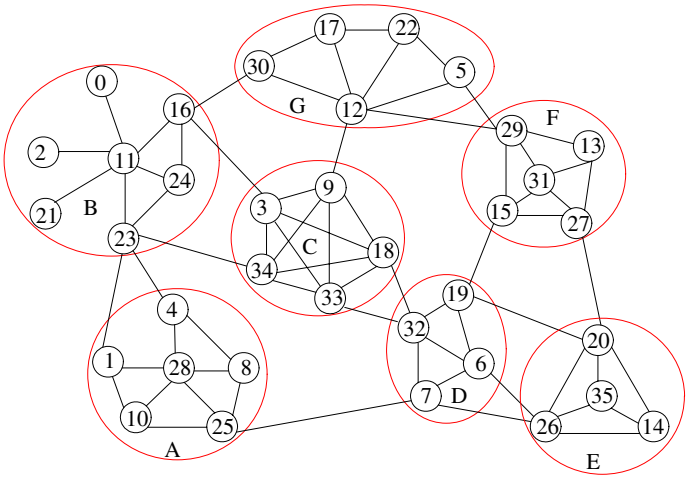

Fig. 1 An illustration of network communities, as inspired by [9]. This example network describes the communication relationships among 32 mobile agents running in an ad-hoc network containing seven portable computers. Seven communities, as denoted by A–G, may be detected based on their communication relationships. Within each community, the communications between agents are frequent, whereas between two communities, the communications are rare. In order to improve the efficiency of the entire network, those agents belonging to the same communities should autonomously move to and/or share the same computers

(2) *Hierarchical methods* agglomerative methods based on the similarity measure [24] and divisive methods, such as the GN algorithm [7], the Tyler algorithm [26], and the Radicchi algorithm [23];
(3) *Methods for detecting Web communities* the MFC algorithm [5], the HITS algorithm [11], the SAE algorithm [21] as well as others [3,12].

A more detailed discussion of these algorithms can be found in [5,18].

In the above-mentioned methods for solving the NCMP, the networks concerned are centralized (i.e., they are processed in a centralized manner and with a global control) and static (i.e., their structures are fixed) instead of being distributed and dynamic. However, in the real world, many applications involve distributed and dynamically evolving networks, in which resources and controls are not only decentralized but also frequently updated. These applications call for new approaches to a new type of NCMP, i.e., *distributed NCMP* (or d-NCMP) that is concerned with finding communities from a distributed and dynamically evolving network. The aim of this paper is to address this problem by presenting a new autonomy-oriented computing (AOC) method [13–15]. This method has the following distinct features.

First, the AOC-based method is applicable in situations where (1) the network to be processed is distributed and/or decentralized by nature, and its complete topological representation is either not available or too hard to obtain, such as WWW, and (2) computation is distributed and/or decentralized because a centralized control is not suitable for some distributed networks, such as P2P networks, in which nodes are computational entities carrying out their tasks autonomously and asynchronously. In this paper, we will present a distributed method, in which a group of autonomous computational agents (called entities) are dispatched into a distributed network and they collectively cluster the entire network into communities based only on their respective local views.

Second, the AOC-based method is suitable for mining dynamically evolving networks. Most of the existing methods and algorithms are off-line or non-incremental, that is, they are designed to deal with static networks rather than dynamic ones. When a clustered network is locally changed (e.g., due to local interactions), the entire network has to be re-processed once again by those algorithms. On the other hand, in our work, we will present an incremental algorithm that works in both distributed and incremental way, locally modifying community structures based on the dynamic updates of a network.

In the literature, there are some works related to ours in certain aspects, such as the methods of "distributed finding maximal cliques" [2,8] and "autonomously clustering sensor networks" [1,17,28]. In what follows, we will take a look at those related methods and compare them with ours.

The maximal clique finding problem is the problem of finding the largest clique (i.e., fully connected subgraph) in a given graph, which is a well-known NP hard problem. It is not expected to be a polynomial-time algorithm, and thus efforts have been made to come up with efficient approximations including parallel and/or distributed algorithms. For examples, Jennings and Motyskovd [8] have presented a distributed algorithm for finding the maximal clique in a graph, in which they adopted a divide-and-conquer strategy to recursively discover all "bipartite cliques" through message passing among distributed nodes. Bui and Rizzo [2] have provided a distributed ant colony optimization (ACO) algorithm to address this problem, where a group of ants work together to find an approximately maximal clique in a given network based on the local knowledge of each and a predefined pheromone distributing and sharing mechanism.

Recent research on sensor networks has addressed the problem of autonomous clustering, as an attempt to reduce the energy consumption of sensors or to easily control a group

of homogenous sensors. For instance, Bae and Yoon have proposed an algorithm that can autonomously cluster geographically adjacent sensors into groups and specify a head for each group at the same time [1]. Other related studies can be found in [17,28], which are concerned with clustering a sensor network into groups, where the variables monitored by sensors, such as the temperature and pressure of different sites on ocean surface, show similar changing patterns. Based on these clusters, remote users can discriminatively distribute different commands to sensors in different clusters.

The basic ideas behind the above-mentioned methods are similar to ours in the sense that all of them aim to find some global patterns of networks, such as cliques, clusters or communities, based on the local knowledge of distributed computational units. However, the AOC-based method to be introduced in this paper differs from them in the following key aspects:

(1) The motivation is completely different. The objective of our method is to detect all communities of a given network defined by the linkage relationships among nodes, rather than the cliques defined as fully connected subgraphs or the clusters of sensor networks defined in terms of the spatial locations of nodes.
(2) In addition, the problem solving mechanism adopted by our method, an AOC-based self-organization and self-aggregation paradigm [13–15], is completely distinct from those used by the above-mentioned methods, such as divide-and-conquer strategy [8], pheromone-based ACO system [2], self-declaration and self-pruning [1], asynchronous signaling [17], or quorum sensing [28].

The remainder of the paper is organized as follows: Sect. 2 gives a formal definition of a distributed NCMP (or d-NCMP) and the basic ideas behind our method. Section 3 presents an AOC-based method for solving the d-NCMP. In Sect. 4, we validate the method using some benchmark and synthetic networks, and examine its performances in detail. Section 5 presents an incremental AOC method for dynamically evolving networks. Finally, Sect. 6 concludes the paper by highlighting the important results of our work as well as some future research work.

2 Formal definition of a distributed NCMP

A distributed NCMP (or d-NCMP) is an NCMP where the nodes of a network and the links associated with them are distributed among a group of autonomous agents at different locations. The objective of solving the d-NCMP is to find all communities hidden in a distributed network based on the local views and interactions (computations) of agents. Formally, the problem can be stated as follows:

Definition 2.1 Let $N = (V, A)$ be a distributed network, where $V = \{v_1, \ldots, v_n\}$ is the set of nodes (also called vertices) and $A = \{<v_i, v_j, w_{ij}>|1 \leq i, j \leq n\}$ is the set of directed links (also called arcs) that connect a pair of nodes with weight w_{ij}. N is distributed among m autonomous agents, and the *view* of each agent p, *where* $1 \leq p \leq m$, is made up of (V_p, A_p), satisfying with following conditions:

(1) $V_p \subseteq V$;
(2) $\bigcup_{1 \leq p \leq m} V_p = V$;
(3) $A_p = \{<v_i, v_j, w_{ij}>|<v_i, v_j, w_{ij}> \in A \wedge v_i \in V_p\}$.

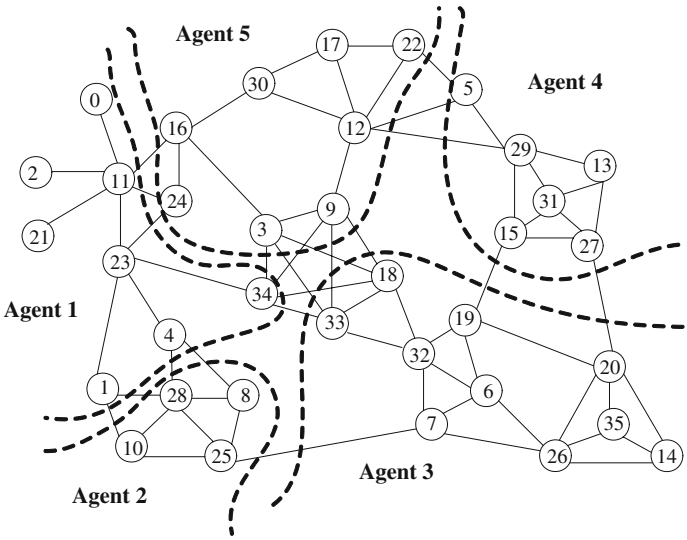

Fig. 2 A schematic representation of the d-NCMP. The network shown in Fig. 1 is distributed among five agents at different locations. Each of them only has a local view of the entire network including the nodes they control and the links going out from these nodes. The task of the d-NCMP is to find all seven network communities denoted by A to G as shown in Fig. 1 through the collaborations among the five agents based on their respective local views

Let $NC(v_i) \subseteq V$ be the *community* containing node v_i. Thus, a d-NCMP can be defined as follows: for each agent p, *where* $1 \leq p \leq m$, how to compute $NC(v_i)$ for all $v_i \in V_p$ based on its view (V_p, A_p).

From the above definition, we can see the essential distinction between a typical NCMP and a d-NCMP. In former case, we wish to find all communities of a given network correctly and efficiently based on its given global topological structure. While in later case, we are especially interested in how to find such communities of a distributed network based only on a set of separated local views of the network.

Example 2.1 As an example, let us observe the network shown in Fig. 1, which is supposed to be distributed among five agents, as illustrated in Fig. 2.

The local view of agent p is denoted by (V_p, A_p). For example, the local view of agent 2 is represented as (V_2, A_2), where:

$$
\begin{aligned}
V_2 &= \{8, 10, 25, 28\};\\
A_2 &= \{<28, 4>, <28, 1>, <28, 8>, <28, 10>, <28, 25>, <8, 4>, <8, 28>,\\
&\quad <8, 25>, <10, 1>, <10, 28>, <10, 25>, <25, 10>,\\
&\quad <25, 28>, <25, 8>, <25, 7>\}.
\end{aligned}
$$

Note that each element $<v_i, v_j, w_{ij}>$ in A_2 is abbreviated as $<v_i, v_j>$ since all weights of the network are equal to 1.

The objectives of the five agents are to discover all the community members for their respectively governed nodes. For example, agent 1 manages to compute all $NC(v_i)$ for each $v_i \in V_1 = \{0, 2, 11, 21, 23, 34, 4, 1\}$ based on its local view (V_1, A_1). As shown in Fig. 1, the objective of agent 1 is to find:

$NC(0) = NC(2) = NC(21) = NC(11) = NC(23) = \{0, 2, 11, 21, 23, 24, 16\}$;
$NC(1) = NC(4) = \{1, 4, 8, 28, 10, 25\}$;
$NC(34) = \{3, 9, 18, 34, 33\}$.

One immediate approach to solving such a d-NCMP is to introduce an administrative agent, who will reconstruct a *global* view of the entire distributed network based on the patched local views provided by all other agents. After that, the administrative agent will apply one of the algorithms designed for the NCMP to discover all communities hidden in the network. The final results will then be sent to each agent from this administrative agent. Obviously, this strategy is not distributed, and will have several inherent limitations in dealing with the huge sizes, decentralized distributions, dynamic features or privacy protecting of networks, as discussed in the introduction. Additionally, this strategy will inevitably be inefficiency in that a whole network will be heavily flooded by too many messages as required to get an integrated structural view of the network.

Due to the above considerations, we will, in this paper, present a fully distributed approach to addressing those raised issues based on the methodology of AOC [13–15].

Generally speaking, an AOC system can be viewed a multi-agent system (MAS), however, with an explicit account of the characteristics of *self-organization*, self-organized computability, interactivity, and computational *scalability* in solving large-scale computationally-hard problems or modeling complex systems. As presented by Liu [13], AOC tackles a computing problem by defining and deploying a system of local autonomy-oriented agents or entities; the entities can be "light-weight" and they may not necessarily be cognitive decision-making entities. The entities spontaneously interact with their local environments and self-organize their structural relationships as well as behavioral dynamics, with respect to some specific control settings. Such a capability is referred to as the *self-organized computability* of autonomous entities. Several general *behavioral principles* underlying an AOC system have been identified. First, it is necessary for the system to manifest *diversification* and *aggregation*. The short- and long-range exploratory actions of entities are useful for achieving computable diversity, whereas positive feedback-based accelerated aggregations are necessary for emerging macroscopically dominant patterns. Second, the entities should embody *collective regulation*. In order to achieve desired macroscopic patterns or a desired global solution, the autonomy of entities needs to be collectively regulated in their deliberative or reactive interactions.

For the clarity of description, in the following presentation of our AOC-based method, we will assume that each agent controls only one node. Note that this assumption will not lower the degree of difficulty of a d-NCMP, and our method described based on the assumption can readily be modified to suit for more general cases, where each agent controls multiple nodes, as stated in Definition 2.1.

3 AOC-based method for solving a distributed NCMP

A typical AOC system is made up of three main elements: a group of autonomous agents, an environment where agents reside, and the system objective [13–15]. A d-NCMP can be instantiated as an AOC system, named AOC-0, as follows:

(1) The environment of the AOC-0 system is the distributed network to be processed;
(2) For each node of the distributed network, there is one autonomous agent residing on it;

(3) The system objective of the AOC-0 system is to find the solution of the d-NCMP, that is, to discover all communities hidden in its environment through the self-organization and self-aggregation of autonomous agents.

In this section, we will describe the basic formulations and algorithms of the AOC-0 system for solving the d-NCMP.

3.1 Environment of the AOC-0 system

The environment of the AOC-0 system, E, is defined as a distributed network in which a group of nodes are distributed. The information about the network is distributed among its nodes, and each node only stores the relations associated to its direct neighbors. Formally, each node of E is defined as a tuple $<ID, NL, MP, DP>$, where the components denote the identifier of a node (ID), the identifiers of its adjacent neighbors (NL), the message pool on the node (MP), and the data pool on the node (DP), respectively.

Given a distributed network $N = (V, A)$, where $V = \{v_1, \ldots, v_n\}$ and $A = \{<v_i, v_j, w_{ij}> | 1 \leq i, j \leq n\}$, node v_i of E is initialized as follows:

$$\begin{aligned} v_i.ID &= i; \\ v_i.NL &= \{<i, j, w_{ij}> | <v_i, v_j, w_{ij}> \in A\}; \\ v_i.MP &= v_i.DP = \{\}. \end{aligned}$$

The identifier of a node is also used to denote the address of the location, in which the node situates. After knowing the identifier of a node, agents in the environment E can send messages to it. All messages received by a node will be stored in its messages pool MP. The data produced by agents will be stored in the data pools of the nodes that they respectively control. We assume that in the environment E, there exists a message passing mechanism which guarantees that each message be delivered to its destination; this can be implemented by one of the reliable transfer protocols.

3.2 System objective of the AOC-0

The system objective of the AOC-0, Ψ, is defined as follows:

$$\Psi = \psi_1 \cup \cdots \cup \psi_n$$

where n is the total number of agents in the AOC-0, and ψ_p is the objective of agent p that is defined as:

$$\psi_p = NC(v_p)$$

where v_p is the node controlled by agent p. The system objective of the AOC-0 will be achieved if the distributed agents autonomously and asynchronously achieve their respective objectives.

3.3 Agent model of the AOC-0

The agent of the AOC-0 is characterized by five attributes, including: identifier, clock, objective, view, and actions, denoted as $<ID, t, \psi, view(t), actions>$. There is only one agent residing on each node of the environment E. The agent on node v_p is initialized as follows:

$agent.ID = v_p.ID = p;$
$agent.t = 0;$
$agent.\psi = NC(v_p);$
$agent.view(0) = v_p.NL;$
$agent.actions = <get\_view, evaluate\_view, shrink\_view,$
$enlarge\_view, balance\_view, sleep>.$

The variable t denotes the clock maintained by agent p, which is used to implement the synchronization between agent p and other agents. After taking each action, agent p will adjust its clock by $t \leftarrow t + 1$. Agent p will become inactive, when its clock reaches the final time T, which is preset by the AOC-0.

As stated in Definition 2.1, the view of agent p (also denoted as $agent_p$) on an entire network is denoted as (V_p, A_p), which represents the set of nodes controlled by the agent and the set of links going out from these nodes. In the AOC-0, the view of $agent_p$ is abbreviated as A_p, because one agent controls only one node, and hence we have $V_p = \{v_p\}$.

Agent view is a critical concept in our purposed AOC-based method, in which agents update their respective views autonomously and asynchronously by taking some predefined actions. $agent.view(t)$ denotes the view of an agent at time t. Initially, the view of each agent, $agent.view(0)$, is set to the NL value of the node it controls; that is to say, at the beginning, each agent can only see the adjacent area of the node. Then, each agent starts to update (e.g., shrink or enlarge) its own view based on the information it gathers from its environment, until it finds that its view cannot be changed any more. During the course, for $agent_p$, the nodes belonging to the network community $NC(v_p)$ will be added into its view one by one. In contrast, those nodes not belonging to $NC(v_p)$ will be gradually excluded out of its view. At the time T, or when the agent view has converged, all the nodes that remain in the view of $agent_p$, $agent_p.view(T)$, are regarded as the members of the network community $NC(v_p)$. In other words, we have:

$$agent_p.\psi = NC(v_p) = \{v_i | <p, i, w_{pi}> \in agent_p.view(T)\} \cup \{v_p\}.$$

3.3.1 Getting views from other agents

Agent p takes the action $agent_p.get\_view(q, t)$ to get the view of agent q at time t, i.e., $agent_q.view(t)$. From the viewpoint of implementation, this action will be accomplished through the cooperation between both sides as follows.

On the side of agent p: First, it sends a request message $<p, q, t>$ to node v_q. Then, it suspends its current execution and waits for the reply from node v_q. After receiving the reply, it will resume its execution from the last interruption.

On the side of agent q: It will periodically process its message pool. First, it picks up a message $<p, q, t>$ from its MP. Then, it matches the message with the data stored in its data pool. If it finds the right datum, $agent_q.view(t)$, in its DP, it will send it to agent p and delete the message $<p, q, t>$ from its MP.

3.3.2 Evaluating agent view

Agent p on node v_p takes the action $agent_p.evaluate\_view(t)$ to evaluate its current view at time t. In other words, to compute the similarities between node v_p and other nodes in its current view. The nodes with larger similarity values are more likely to belong to the

community $NC(v_p)$. In the following actions, based on such similarity values worked out in this action, agent p will determine which nodes will be kept in, and which ones will be excluded from, its current view.

In an unweighted and undirected network, the similarity of node v_i and node v_j can be defined as:

$$s_{ij} = \frac{|K(i) \cap K(j)|}{|K(i)|} \cdot \frac{|K(i) \cap K(j)|}{|K(j)|} \tag{3.1}$$

where $K(i)$ is the set of neighbors of node v_i.

The main idea behind this definition is inspired from the human society where two individuals sharing more acquaintances are more likely in the same social circle. s_{ij} actually computes the probability of two individuals belonging to the same community. This can readily be understood based on the following *random talking* model.

Considering two persons, i and j, who are freely talking about their respective acquaintances to each other. The probability that the one randomly talked about by person i is also known by person j is:

$$p(i, j) = \frac{|K(i) \cap K(j)|}{|K(i)|}. \tag{3.2}$$

Similarly, the probability that the one randomly talked about by person j is also known by person i is:

$$p(j, i) = \frac{|K(i) \cap K(j)|}{|K(j)|}. \tag{3.3}$$

In both cases, the function $K(\cdot)$ denotes the acquaintance set of a given person. The probabilities of $p(i, j)$ and $p(j, i)$ measure the degree of one person knowing the other, thus the probability of $p(i, j)p(j, i)$ measures the degree of two persons knowing each other. As illustrated in Fig. 3a, b, the higher the value of $p(i, j)p(j, i)$, the denser the links appearing among node i, node j, and their respective neighbors, and thus the more likely these nodes belong to the same community due to their highly clustered relationships.

Based on Eq. 3.1, we can define the similarity between two nodes in a weighted, directed, and distributed network.

Definition 3.1 Let v_p and v_q be two nodes in a distributed network. Agents p and q are on nodes v_p and v_q, respectively. The *similarity* between nodes v_p and v_q at time t is defined as follows:

$$\begin{aligned} s_{pq}^t &= S(agent_p.view(t), agent_q.view(t)) \\ &= \frac{\sum_{k \in K_p(t) \cap K_q(t)} w_{pk}^*}{\sum_{k \in K_p(t)} w_{pk}^*} \cdot \frac{\sum_{k \in K_p(t) \cap K_q(t)} w_{qk}^*}{\sum_{k \in K_q(t)} w_{qk}^*} \end{aligned} \tag{3.4}$$

where $K_i(t) = \{i\} \cup \{k | <i, k, w_{ik}> \in agent_i.view(t)\}$ and $w_{ik}^* = agent_i.view(t).<i, k, w_{ik}>.w_{ik}$.

The action $agent_p.evaluate\_view(t)$ is defined in Table 1, in which symbol "$\leftarrow$" denotes "assigned to". In Step 7, agent p submits its current view $agent_p.view(t + 1)$ to the data pool on node v_p.

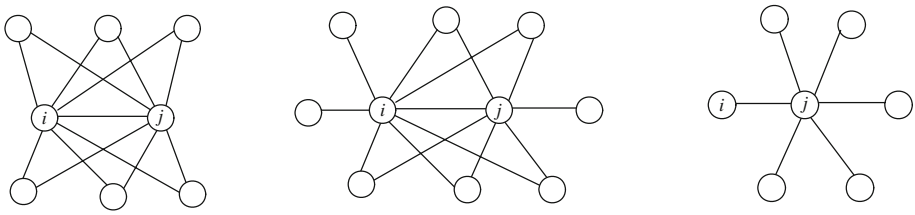

Fig. 3 The mutually linked relationship between two talkers in the *random talking* model. In **a** and **b**, talkers i and j are more likely in the same community with a high probability of $s_{ij} = 0.73$ and $s_{ij} = 0.45$, respectively, In **c**, the probability s_{ij} obtained by Eq. 3.1 is 0, which will isolate talker i from others, while this probability obtained by Eq. 3.4 is 0.17, which is more feasible

Table 1 The definition of the action $agent_p.evaluate\_view(t)$

1. $agent_p.view(t+1) \leftarrow \{\}$;
2. for $\forall < p, q, w_{pq} > \in agent_p.view(t)$
3. $\quad agent\_q\_view\_t \leftarrow agent_p.get\_view(q, t)$;
4. $\quad s^t_{pq} \leftarrow S(agent_p.view(t), agent\_q\_view\_t)$;
5. $\quad agent_p.view(t+1) \leftarrow agent_p.view(t+1) \cup \{<p, q, s^t_{pq}>\}$;
6. end
7. $v_p.DP \leftarrow v_p.DP \cup agent_p.view(t+1)$;

Example 3.1 Considering the distributed network shown in Fig. 1, in which there are total 36 agents on 36 nodes. At the beginning, i.e., $t = 0$, the views of all agents are listed as follows:

$agent_0.view(0) = \{<0, 11, 1>\}$;
$agent_1.view(0) = \{<1, 23, 1>, <1, 28, 1>, <1, 10, 1>\}$;
… …
$agent_{35}.view(0) = \{<35, 20, 1>, <35, 26, 1>, <35, 14, 1>\}$.

After each agent p takes the action $agent_p.evaluate\_view(0)$, the views of all agents at time 1, i.e., $t = 1$, are obtained, as listed as follows:

$agent_0.view(1) = \{<0, 11, 0.17>\}$;
$agent_1.view(1) = \{<1, 23, 0.07>, <1, 28, 0.27>, <1, 10, 0.44>\}$;
… …
$agent_{35}.view(1) = \{<35, 20, 0.6>, <35, 26, 0.6>, <35, 14, 1>\}$.

For the sake of description, let us represent the *views of all agents* using a *matrix*, called *agent view matrix* at time t or $AVM^{(t)}$, which is defined as follows:

$$AVM^{(t)}(i, j) = \begin{cases} w^*_{ij}, & \text{if } <i, j, w_{ij}> \in agent_i.view(t) \\ 0, & \text{else} \end{cases} \tag{3.5}$$

where $w^*_{ik} = agent_i.view(t).<i, k, w_{ik}>.w_{ik}$.

We note that the $AVM^{(0)}$ is, in fact, the adjacency matrix of a network. After each agent p completes the action $agent_p.evaluate\_view(0)$, we have the $AVM^{(1)}$ of the network updated

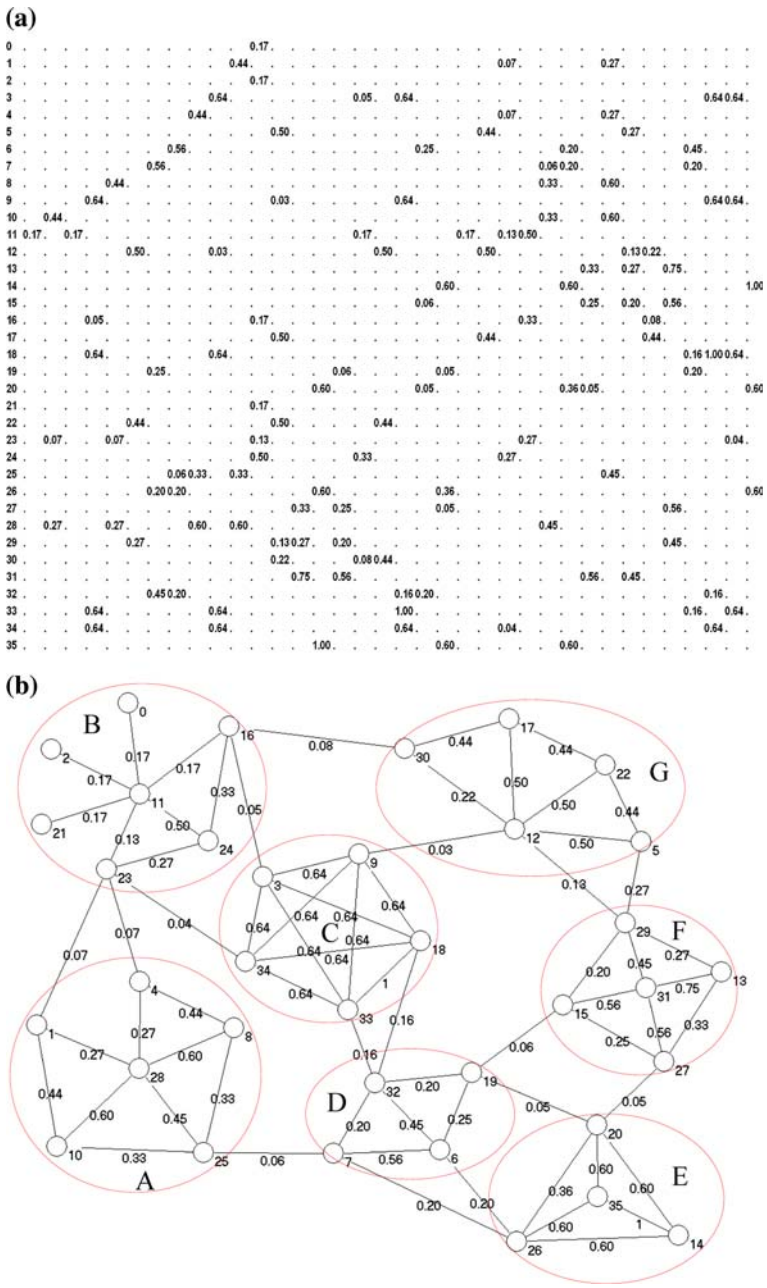

Fig. 4 The *agent view matrices* and corresponding *agent networks* in the AOC-0. **a** Agent view matrix at time 1, $AVM^{(1)}$, with respect to the network shown in Fig. 1. **b** Agent network at time 1, $AN^{(1)}$, with respect to the network shown in Fig. 1

Table 2 The definition of the action $agent_p.shrink\_view(t)$

1. $agent_p.view(t+1) \leftarrow \{\}$;
2. $\mu_p^t = \frac{\sum_q agent_p.view(t).<p,q,w_{pq}>.w_{pq}}{|agent_p.view(t)|}$;
3. $\sigma_p^t = \left(\frac{\sum_q (agent_p.view(t).<p,q,w_{pq}>.w_{pq}-\mu_p^t)^2}{|agent.view(t)|}\right)^{1/2}$;
4. for $\forall <p, q, w_{pq}> \in agent_p.view(t)$
5. if $w_{pq} \geq \omega_1 \cdot (\mu_p^t + \omega_2 \cdot \sigma_p^t)$ then
6. $agent_p.view(t+1) \leftarrow agent_p.view(t+1) \cup \{<p, q, w_{pq}>\}$;
7. end
8. end
9. $v_p.DP \leftarrow v_p.DP \cup agent_p.view(t+1)$;

with similarity-valued relationships. As an example, the $AVM^{(1)}$ of such an updated network, with respect to the original network of Fig. 1, is given in Fig. 4a.

In what follows, we will refer to the network whose adjacency matrix is given by $AVM^{(t)}$ as an *agent network* at time t, denoted as $AN^{(t)}$, where each node denotes an agent and the links describe the relationships among these agents in terms of the similarity measure as mentioned earlier. Corresponding to the $AVM^{(1)}$ of Fig. 4a, the *agent network* $AN^{(1)}$ for the network of Fig. 1 is shown in Fig. 4b, from which we observe that the similarities between the nodes belonging to different communities are much less than those belonging to the same communities.

The time complexity of the action $agent_p.evaluate\_view(t)$ is $O(d_t^2)$, where d_t is the average degree of the *agent network* at time t. Let $d_t^p = |agent_p.view(t)|$ be the degree of agent p at time t, then we have $d_t = \frac{1}{n}\sum_{p=1}^{n} d_p^t$, where n is the total number of agents.

3.3.3 Shrinking agent view

Agent p on node v_p takes the action $agent_p.shrink\_view(t)$ to exclude those nodes that may not belong to the network community $NC(v_p)$ from its current view at time t. The action $agent_p.shrink\_view(t)$ is defined in Table 2, in which ω_1 and ω_2 are two constants, μ_p^t and σ_p^t are the mean and the standard deviation of the view of agent p, respectively.

The time complexity of the action is $O(d_t)$, where d_t is the average degree of the *agent network* at time t, i.e., $AN^{(t)}$.

3.3.4 Enlarging agent view

Agent p on node v_p takes the action $agent_p.enlarge\_view(t)$ to incorporate those nodes that may belong to the network community $NC(v_p)$ into its current view at time t. This action is inspired by the human society in which an individual is likely to make new friends out of his/her old friends' friends. The action $agent_p.enlarge\_view(t)$ is defined in Table 3, where the denotations, such as ω_1, ω_2, μ_p^{t+1}, and σ_p^{t+1}, are the same as those introduced in Table 2. The function $rand(S)$ will randomly select one element from the set S and return it.

The time complexity of the action is $O(d_t)$, where d_t is the average degree of the *agent network* at time t, $AN^{(t)}$.

Example 3.2 Considering the network shown in Fig. 5a, in which there are in total seven agents on seven nodes. At time t, the view of agent i is:

$$agent_i.view(t) = \{<i, 1, 0.3>, <i, j, 0.5>, <i, 5, 0.15>\}.$$

Table 3 The definition of the action $agent_p.enlarge\_view(t)$

1. $agent_p.view(t+1) \leftarrow agent_p.view(t)$;
2. $q \leftarrow rand(\{i | <p, i, w_{pi}> \in agent_p.view(t+1)\})$;
3. $agent\_q\_view\_t \leftarrow agent_p.get\_view(q, t)$;
4. $w_{pq} \leftarrow agent_p.view(t+1).<p, q, w_{pq}>.w_{pq}$;
5. for $\forall <q, i, w_{qi}> \in agent\_q\_view\_t$
6. if $<p, i, w_{pi}> \in agent_p.view(t+1)$ then
7. $w_{pi} \leftarrow agent_p.view(t+1).<p, i, w_{pi}>.w_{pi}$;
8. else
9. $w_{pi} \leftarrow 0$;
10. end
11. $s_{pi} \leftarrow \max(w_{pi}, w_{pq} \cdot w_{qi})$;
12. $agent_p.view(t+1) \leftarrow agent_p.view(t+1) - \{<p, i, w_{pi}>\} \cup \{<p, i, s_{pi}>\}$;
13. end
14. for $\forall <p, q, w_{pq}> \in agent_p.view(t+1)$
15. if $w_{pq} < \omega_1 \cdot (\mu_p^{t+1} + \omega_2 \cdot \sigma_p^{t+1})$ then
16. $agent_p.view(t+1) \leftarrow agent_p.view(t+1) - \{<p, q, w_{pq}>\}$;
17. end
18. end
19. $v_p.DP \leftarrow v_p.DP \cup agent_p.view(t+1)$;

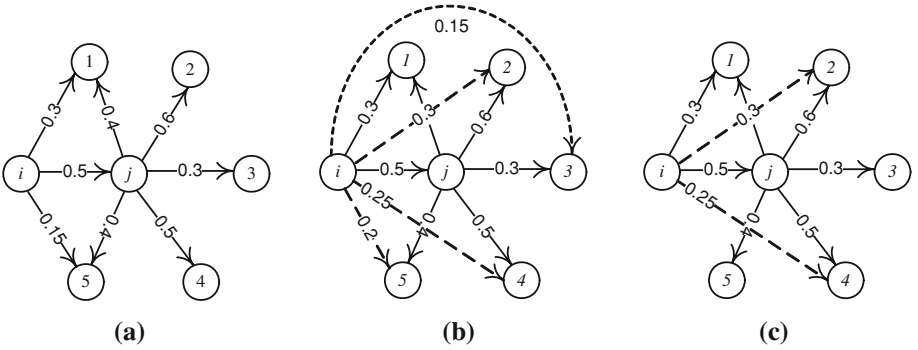

Fig. 5 The schematic illustration of the action *enlarge_view*. **a** presents the original network in which agent *i* will apply the action. First, agent *i* randomly selects one neighbor out of its three neighbors, denoted as agent *j*. Then, agent *i* chooses all the neighbors of agent *j* as its new neighbors, as illustrated in (**b**), where the *dashed lines* denote the newly created links. Finally, agent *i* evaluates its current neighbors and abandons those with low similarity values, as illustrated in (**c**)

After agent *i* completes the action, the view of agent *i* at time $t+1$ is listed as follows, where two new nodes, nodes 2 and 4, are added to the agent view, and at the same time, node 5 is deleted:

$$agent_i.view(t+1) = \{<i, 1, 0.3>, <i, 2, 0.3>, <i, j, 0.5>, <i, 4, 0.25>\}.$$

Table 4 The definition of the action $agent_p.balance\_view(t)$

1. $agent_p.view(t+1) \leftarrow \emptyset$; (i.e., *null*)
2. for $\forall <p, q, w_{pq}> \in agent_p.view(t)$
3. $\quad agent\_q\_view\_t \leftarrow agent_p.get\_view(q, t)$;
4. $\quad$ if $<q, p, w_{qp}> \in agent\_q\_view\_t$ then
5. $\quad\quad s_{pq} \leftarrow (w_{pq} + w_{qp})/2$;
6. $\quad$ else
7. $\quad\quad s_{pq} \leftarrow w_{pq}/2$;
8. $\quad$ end
9. $\quad agent_p.view(t+1) \leftarrow agent_p.view(t+1) \cup \{<p, q, s_{pq}>\}$;
10. end
11. $v_p.DP \leftarrow v_p.DP \cup agent_p.view(t+1)$;

3.3.5 Balancing agent view

Finally, let us take a look at the action $agent_p.balance\_view(t)$ as performed by agent p on node v_p. This action will enable the agent to average the similarity values between v_p and the nodes in its view at time t (Table 4).

The time complexity of the action is $O(d_t^2)$, where d_t is the average degree of the *agent network* at time t, $AN^{(t)}$.

3.3.6 The life-cycle of an agent

Based on the actions discussed above, the life-cycle of agent p on node v_p is given in Table 5.

The life-cycle of the agent in the AOC-0 is divided into three main phases: initialization phase (Steps 1–3), active phase (Steps 4–13), and inactive phase (Steps 14 and 15).

In the initialization phase, agent p initializes itself by setting its starting clock and its initial view, and putting the view into the data pool of the node it resides on.

In the active phase, agent p takes its actions, as a sequence of *evaluate_view*, *shrink_view*, *enlarge_view,* and *balance_view*, to repeatedly update its view until it becomes stabilized or its clock reaches a predefined threshold. After the view of agent p become stabilized, i.e., its current view is identical to its previous view, we say that a convergent view has been reached. At this time, the current value of t is denoted as t_c, and the current view is called the *convergent view* of agent p. For agent p, its convergent view becomes:

$$agent_p.view(t_c) = \{<p, q_1, 1>, \ldots <p, q_l, 1>\}$$

where l denotes the number of elements in its convergent view. Note that all weights in the convergent view are equal to 1.

When the clock is equal to t_c or T, agent p will stop updating. At this time, agent p finds all members belonging to the network community $NC(v_p)$ based on its convergent view, i.e.,

$$agent_p.\psi = NC(v_p) = \{v_i | (p, i, w_{pi}) \in agent_p.view(t)\} \cup \{v_p\}$$

where $t = \min\{t_c, T\}$. Now, the agent has accomplished its objective, and becomes inactive and sleeps.

Table 5 The life-cycleof $agent_p$

/* initialization phase */
1. $agent_p.t \leftarrow 0$;
2. $agent_p.view(0) \leftarrow v_p.NL$;
3. $v_p.DP \leftarrow v_p.DP \cup agent_p.view(0)$;
/* active phase */
4. while $agent_p.t < T$ do
5. $agent_p.evaluate\_view(agent_p.t)$;
6. $agent_p.shrink\_view(agent_p.t + 1)$;
7. $agent_p.enlarge\_view(agent_p.t + 2)$;
8. $agent_p.balance\_view(agent_p.t + 3)$;
9. $agent_p.t \leftarrow agent_p.t + 4$;
10. if $agent_p.view(agent_p.t) = agent_p.view(agent_p.t - 4)$ then
11. goto 14;
12. end
13. end
/* inactive phrase */
14. $agent_p.\psi = \{v_i | <p, i, w_{pi}> \in agent_p.view(agent_p.t)\} \cup \{v_p\}$;
15. $agent_p.sleep$;

3.4 AOC-based method for solving a distributed NCMP

In this section we summarize the AOC-based method for solving a d-NCMP, and discuss its characteristics. For the sake of notation, the method is named the AOC-d algorithm.

3.4.1 A summary of the AOC-based method

A given d-NCMP can be modeled as an AOC system, in which a distributed network in NCMP is considered as the environment of the AOC system, and the solution of the NCMP is the objective of the AOC system. The nodes in the distributed network are controlled by a group of autonomous agents running autonomously and asynchronously. The collaborations and asynchronizations between them are implemented with the aid of a distributed clock mechanism, in which each agent maintains its own clock. Each agent only has a local view of the entire environment where it resides. During its life-cycle, each agent repeatedly updates its view by taking some predefined actions as shown in Table 5. After the updating process terminates, the final view of each agent contains exactly the members belonging to the network community that the agent aims to find. The agents that have achieved their objectives suspend their actions and become inactive. The objective of the AOC system is thus achieved when all its agents become inactive.

Example 3.3 In this example, we show how to mine network communities hidden in the distributed network shown in Fig. 1 using the AOC-d algorithm. We will focus on a single agent and observe its view at different time steps during its entire life-cycle. In this case, we choose agent 0, and three parameters involved are set as follows: $T = 100$, $\omega_1 = 0.4$, and $\omega_2 = 0.3$.

First, $agent_0$ initializes itself. The time of its clock is set to 0, and its view at time 0 is set to:

$$agent_0.view(0) = \{<0, 11, 1>\}.$$

Then, $agent_0$ starts its active phase. After taking the action *evaluate_view* in Step 5, its view at time 1 becomes:

$$agent_0.view(1) = \{<0, 11, 0.17>\}.$$

After the action *shrink_view* in Step 6, its view at time 2 becomes:

$$agent_0.view(2) = \{<0, 11, 0.17>\}.$$

After the action *enlarge_view* in Step 7, its view at time 3 becomes:

$$agent_0.view(3) = \{<0, 11, 0.17>, <0, 24, 0.08>\}.$$

After the action *balance_view* in Step 8, its view at time 4 becomes:

$$agent_0.view(4) = \{<0, 11, 0.17>, <0, 24, 0.04>\}.$$

Next, in Step 9, $agent_0$ updates its clock; the time of its clock is 4.

In Step 10, $agent_0$ checks whether it has reached a convergent view by comparing its current view, $agent_0.view(4)$, with its previous view, $agent_0.view(0)$. Because $agent_0.view(4)$ is not equal to $agent_0.view(0)$ and the current time is less than T, $agent_0$ goes to Step 4 and starts a new iteration. After finishing the second iteration, $agent_0.t = 8$, and the view of $agent_0$ at time 8 becomes:

$$agent_0.view(8) = \{<0, 2, 0.26>, <0, 21, 0.26>, <0, 23, 0.25>, <0, 11, 0.51>, <0, 24, 0.51>, <0, 16, 0.26>\}.$$

As $agent_0.view(8)$ is not equal to $agent_0.view(4)$ and the current time is less than T, $agent_0$ goes to Step 4 and starts the third iteration. After finishing the third iteration, $agent_0.t = 12$, and the view of $agent_0$ at time 12 becomes:

$$agent_0.view(12) = \{<0, 2, 1>, <0, 21, 1>, <0, 23, 1>, <0, 11, 1>, <0, 24, 1>, <0, 16, 1>\}.$$

Once again, $agent_0.view(12)$ is not equal to $agent_0.view(8)$ and the current time is still less than T. $agent_0$ goes to Step 4 and starts the fourth iteration, after which the view of $agent_0$ at time 16 becomes:

$$agent_0.view(16) = \{<0, 2, 1>, <0, 21, 1>, <0, 23, 1>, <0, 11, 1>, <0, 24, 1>, <0, 16, 1>\}.$$

In this case, $agent_0$ has reached its convergent view since $agent_0.view(16)$ is exactly equal to $agent_0.view(12)$. So, it quits the cycle of view updating.

In the inactive phase, $agent_0$ obtains all the members belonging to the network community $NC(0)$ by taking the action in Step 14:

$$\begin{aligned} agent_0.\psi &= \{i | (p, i, w_{pi}) \in agent_0.view(16)\} \cup \{0\} \\ &= \{2, 21, 23, 11, 24, 16, 0\} \\ &= NC(0). \end{aligned}$$

Finally, $agent_0$ becomes inactive by taking *sleep* in Step 15.

Similarly, we can observe the activities of other agents. Using the concept of *agent network*, as introduced in Sect. 3.3.2, we can easily visualize the dynamic process of view updating

of all agents in the AOC system, as shown in Fig. 6. Figure 6a–h, respectively, present the snapshots of the dynamically updated agent networks at different time steps, in which circular nodes denote active agents and triangular nodes denote inactive agents. At time 24, i.e., after six iterations, all agents in the AOC system find their respective network communities and become inactive.

3.4.2 Self-organizing agent network and positive feedback mechanism

It can be noted that in the AOC-0 system, there are two types of networks, which form a two-layer structure. The bottom layer is the distributed network (i.e., physical network) to be mined, and the top layer is the agent network (i.e., conceptual network) consisting of the views of all agents upon the bottom layer. The agent network is dynamically updated as the views of agents keep on evolving during their life-cycles. When agent p appears in the view of agent q, a new link from q to p will be added to the agent network. In contrast, when agent p is excluded from the view of agent q, the existing link from q to p will be deleted from the agent network. As directed by the predefined actions, the agent network evolves continuously, from an old structure to a new one, similar to such dynamic systems as human societies and ecosystems.

The evolving process of the agent network is, in fact, a positive feedback process, in which the links between agents corresponding to the nodes in the same network communities will gradually emerge, while the links between agents corresponding to the nodes in different network communities will gradually disappear. Based on the positive feedback mechanism, agents self-organize themselves into a number of separated groups; each group corresponds to one network community of the distributed network in question. The finally evolved agent network comes into being when all its agents become inactive, which is made up of a set of separated cliques, and each of them corresponds to one network community. For example, in Fig. 6i, there are seven cliques in the finally evolved agent network, each of which corresponds to a network community of the distributed network as shown in Fig. 1.

Example 3.4 In this example, we will examine the positive feedback mechanism of the AOC-d algorithm with the aid of the *agent view matrix*, *AVM*, as introduced in Sect. 3.3.2.

The network to be discussed here is a synthetic random network, which is commonly used to test the performance of algorithms for mining a community structure [7,19,20,23]. The community structure of the random network is known beforehand. But, it is stochastic in other respects as regulated by parameters. The network is specified as $G(c, n, k, p_{in})$, where c is the number of communities in the network; n is the number of nodes in each community; k is the average degree of each node; p_{in} is the probability of each node connecting other nodes in the same community. Obviously, as $p_{in} \to 0$, the community structure of a random network becomes increasingly ambiguous.

In this example, the random network to be processed is $G(4, 64, 30, 0.5)$, and we set $T = 50$, $\omega_1 = 0.3$ and $\omega_2 = 0.2$. Figure 7a presents the *agent view matrices* for the random network at different time steps, in which each dot "." denotes a non-zero element, i.e., a non-zero weight associated with a link between two agents. The nodes belonging to the same community are arranged together. We can see four dense areas along the diagonal line in these matrices. Each of them corresponds to a community. The evolving process of the agent network converges after 28 time steps, as shown in Fig. 7a–h. From them we can see that as the agent network evolves, the links between communities (the dots out of the four dense

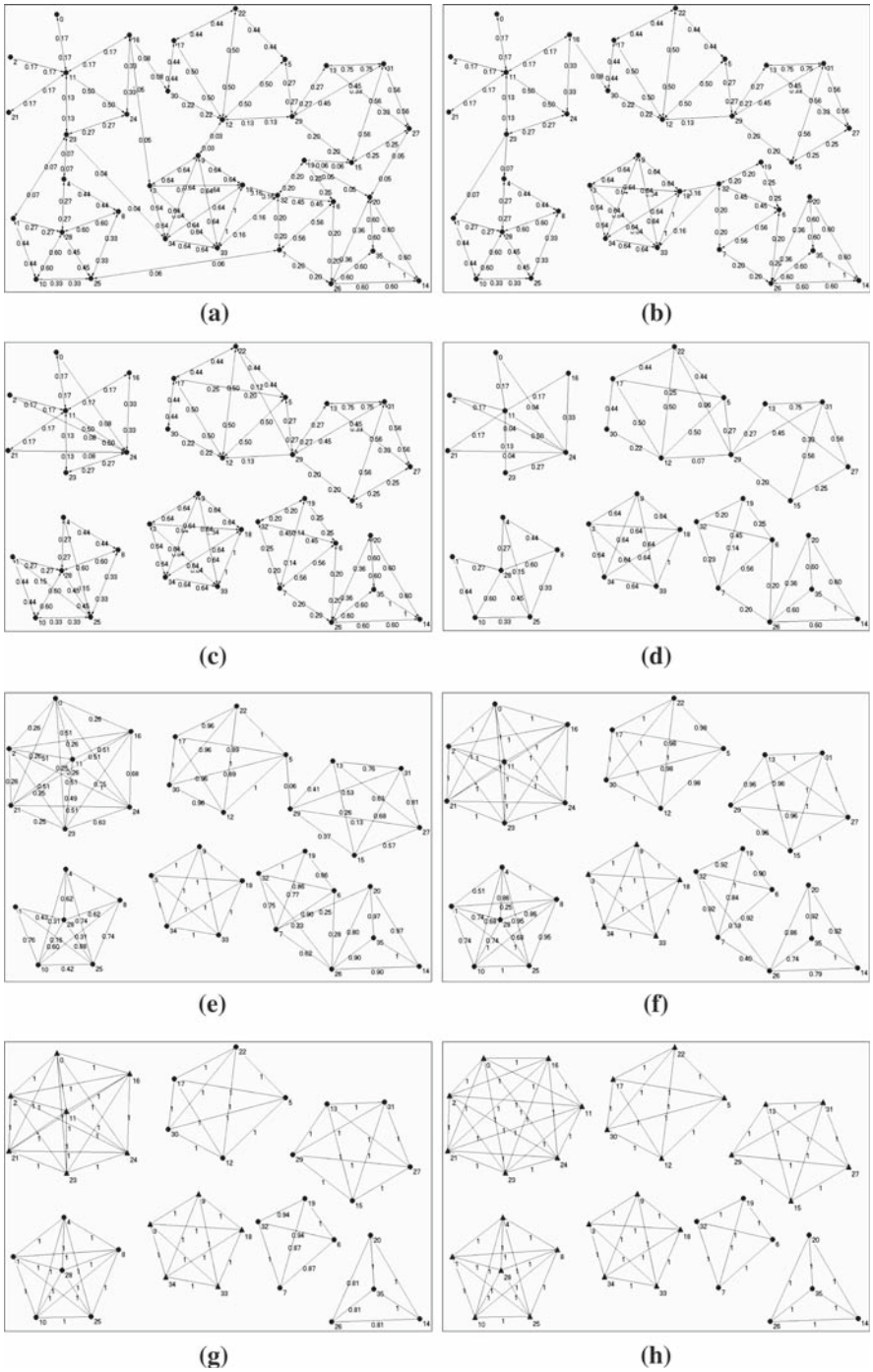

Fig. 6 The updating of an *agent network* with respect to the distributed network as shown in Fig. 1. **a** Agent network at time $t = 1$, **b** agent network at time $t = 2$, **c** agent network at time $t = 3$, **d** agent network at time $t = 4$, **e** agent network at time $t = 8$, **f** agent network at time $t = 12$, **g** agent network at time $t = 16$, **h** agent network at time $t = 20$ and $t = 24$

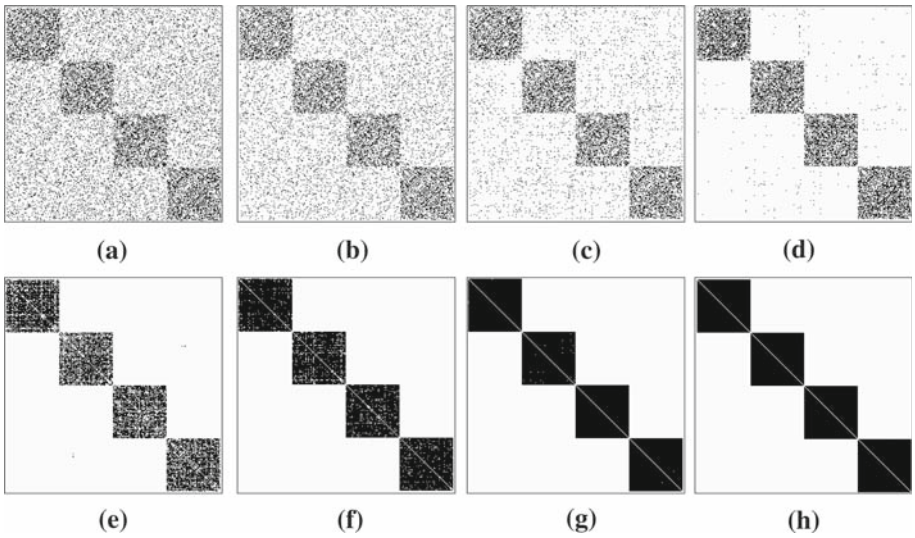

Fig. 7 Examining the positive feedback mechanism of the AOC-d algorithm, in terms of a series of evolving agent view matrices, with respect to the random network G(4, 64, 30, 0.5). **a** $AVM^{(0)}$; **b** $AVM^{(4)}$; **c** $AVM^{(8)}$; **d** $AVM^{(12)}$; **e** $AVM^{(16)}$; **f** $AVM^{(20)}$; **g** $AVM^{(24)}$; **h** $AVM^{(28)}$

areas) gradually disappear, while those within communities (the dots within the four dense areas) gradually increase. Finally, the agent network evolves into a new network containing four separated cliques, each of which corresponds to a community of the random network, as shown in Fig. 7h.

3.4.3 Time complexity of the AOC-d algorithm

Let n be the number of nodes in a distributed network in question. So, there are in total n agents running asynchronously in the AOC-0 system. The time complexity of the life-cycle of each agent, as listed in Table 5, is:

$$\sum_{t=0}^{\min\{(t_c-3)/4,(T-3)/4\}} \left(\mathrm{O}(d_{4_t}^2) + \mathrm{O}(d_{4_{t+1}}) + \mathrm{O}(d_{4_{t+2}}) + \mathrm{O}(d_{4_{t+3}}^2)\right) \leq \mathrm{O}(T \cdot d_T^2)$$

where t_c is the time when an agent finds its convergent view, and d_t is the average degree of the *agent network* at time t.

When T is considered as a constant, the time in the worst case required by each agent to run through its life-cycle is $O(d_T^2)$. Let k be the number of communities in the distributed network in question. So, in the finally evolved agent network, there will be k separated cliques, and we have:

$$d_T = \frac{2 \cdot m_T}{n} = 2 \cdot k \cdot \frac{n}{k} \cdot \left(\frac{n}{k} - 1\right) \cdot \frac{1}{2n} = \mathrm{O}(n/k)$$

where m_T denotes the number of links in the agent network at time T.

So, for each agent in the AOC-0, the time required to complete its task in the worst case is:

$$WT = \mathrm{O}(d_T^2) = \mathrm{O}(n^2/k^2).$$

In the case of $k \ll n$, i.e., the number of communities in the distributed network is very small and k can be considered as a constant, we have:

$$WT = \mathrm{O}(d_T^2) = \mathrm{O}(n^2/k^2) = \mathrm{O}(n^2).$$

While in the case of $k \geq \sqrt{n}$, i.e., the number of communities in the distributed network is quite large, we have:

$$WT = \mathrm{O}(d_T^2) = \mathrm{O}(n^2/k^2) = \mathrm{O}(n).$$

That is to say, each agent will quickly complete its task within a constant time.

From the above analysis, we can see that given a network, the efficiency of the AOC-d algorithm is related to the community structure of the network. Generally, the more communities the network contains, the more efficient the AOC-d algorithm is.

4 Evaluation of the AOC-d algorithm

4.1 Validation of the AOC-d algorithm

In this section, we will validate the AOC-d algorithm using some benchmark networks that are commonly used in related work. Although these networks are given as centralized representations, for the purpose of testing our distributed method, here we treat them as distributed networks, i.e., we consider that their nodes and links are distributed (e.g., over different sources, or geographical locations).

4.1.1 Karate network

Figure 8a shows the karate club network. This network corresponds to the observation made by Wayne Zachary [31] on the social interactions between members of a karate club at an American university. During his study, it was noticed that a dispute arose between the club's administrator and its principal karate teacher. As a result, the club split into two roughly equal-sized clubs: One was led by the administrator, represented by squares, and the other by its teacher represented by circles, as shown in Fig. 8a. In this experiment, we set $\omega_1 = 0.2$ and $\omega_2 = 0.2$. The process of agent network updating converges after 24 time steps. Figure 8b presents the result as obtained by the AOC-d algorithm, which is presented by the finally evolved agent network. As compared with the actual division shown in Fig. 8a, only node 10 is misclassified.

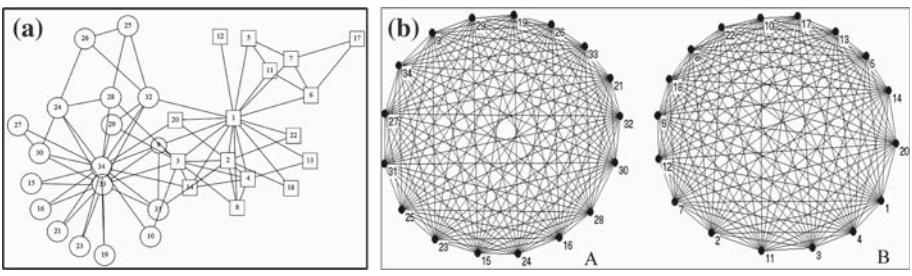

Fig. 8 Mining communities from the karate club network using the AOC-d algorithm

Now, let us observe agent 17, which is on node 17 belonging to the community B. Its initial view, convergent view and objective are listed as follows:

$agent_{17}.view(0) = \{<17, 6, 1>, <17, 7, 1>\}$;
$agent_{17}.view(24) = \{<17, 4, 1>, <17, 3, 1>, <17, 11, 1>, <17, 2, 1>, <17, 7, 1>, <17, 12, 1>, <17, 6, 1>, <17, 18, 1>, <17, 8, 1>, <17, 22, 1>, <17, 10, 1>, <17, 1, 1>, <17, 13, 1>, <17, 5, 1>, <17, 14, 1>, <17, 20, 1>\}$;
$agent_{17}.\psi = \{4, 3, 11, 2, 7, 12, 6, 18, 8, 22, 10, 17, 13, 5, 14, 20, 1\}$.

4.1.2 Football network

As another test, we turn to the network of US college football association in the 2000 season, which was previously used by Girvan and Newman [7]. This network contains 115 nodes and 616 links, which correspond to football teams and games among those teams, respectively. The 115 teams were grouped into 12 conferences; games between members of the same conference would be more frequent than between those of different conferences. In this case, each conference can be naturally considered as one community of the network. Figure 9a is the adjacency matrix of the initial foot association network, in which Name (ID) on the

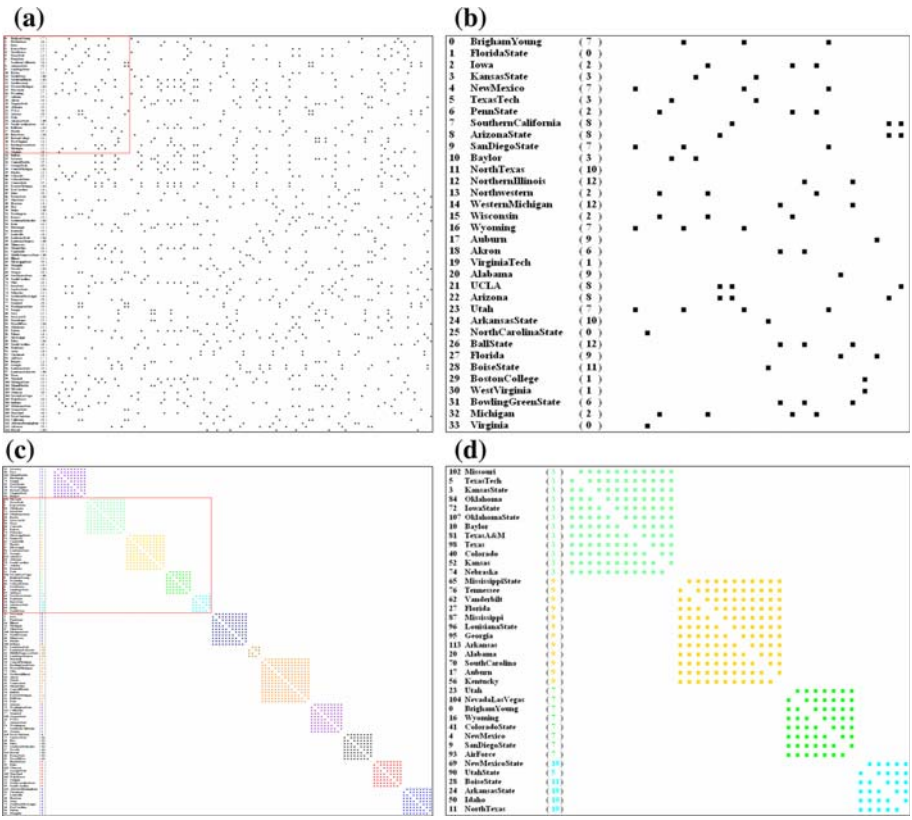

Fig. 9 Mining communities from the football association network. **a** The adjacency matrix of the football network, **b** the enlarged view of the rectangular area in (**a**), **c** the agent view matrix of the finally evolved agent network, **d** the enlarged view of the rectangular area in Fig. 9c

left of each row denotes the name of a football team and the ID of the conference it belongs to. In this experiment, we set $\omega_1 = 0.3$ and $\omega_2 = 0.2$. The updating of the agent network converges after 28 time steps.

Figure 9d shows the final agent view matrix for the football network, in which 12 cliques are formed, and each of them corresponds to one conference, respectively. As compared with real communities, most associations are detected correctly except for a few teams, such as five teams of IA Independents (No. 5), teams 28 and 58 of Western Athletic (No. 11), and team 110 of Texas Christian (No. 4). The reason for misclassification is that these teams play more matches with teams in other associations than with those in their own associations.

4.1.3 Dolphin network

Figure 10a presents the dolphin network that represents the social relationships of 62 bottlenose dolphins living in Doubtful Sound of New Zealand. This network was introduced by Lusseau [16] based on his observations of these dolphins for seven years. As he observed, the dolphins were, for some reasons, divided into two groups, as shown in Fig. 10a, b presents the final agent view matrix as obtained by the AOC-d algorithm, in which two groups, A and B, are correctly divided. Furthermore, four latent subdivisions of group B are also predicted by

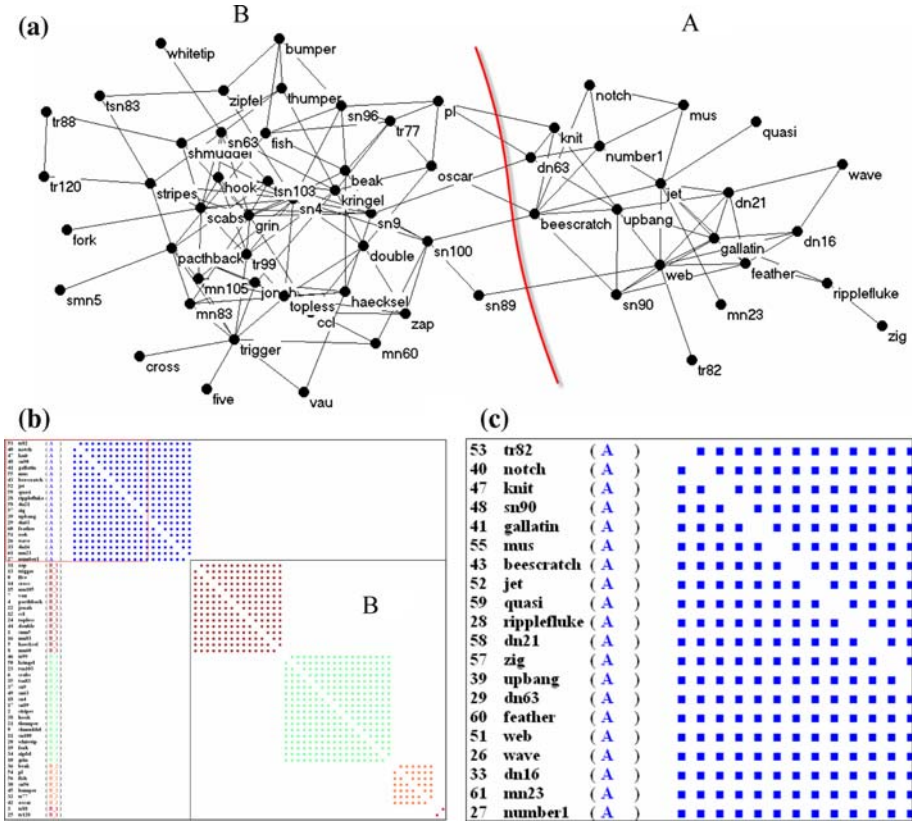

Fig. 10 Mining communities from the dolphin network. **a** The dolphin network, **b** the final agent view matrix, **c** the enlarged view of the left-top rectangular area in Fig. 10b

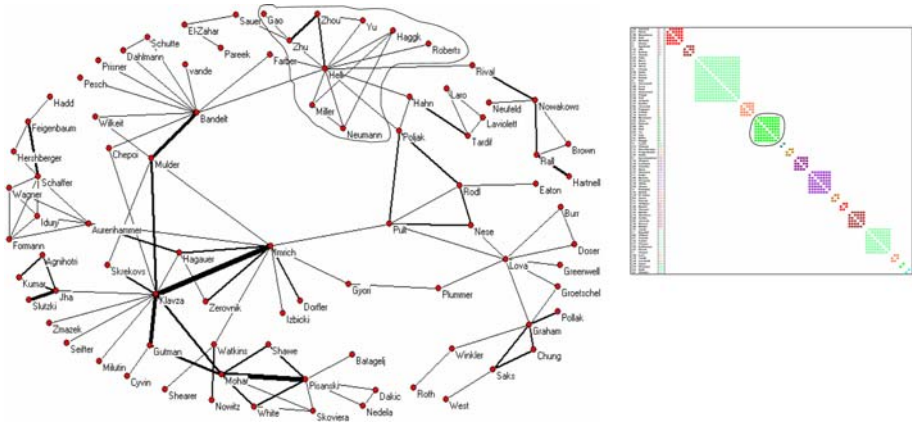

Fig. 11 An illustrative iPDA network distributed at a certain time in terms of communications between individuals, where the weight of an edge denotes the frequency of communications. The right-hand-side panel is the agent view matrix of the finally-evolved agent network

the algorithm. In this experiment, we set $\omega_1 = 0.25$ and $\omega_2 = 0.2$, and the evolving process converges after 36 time steps.

4.1.4 An application of the AOC-d algorithm in iPDA

iPDAs (intelligent Portable Digital Assistants) carried by people form a distributed network in which people communicate with each other through calls or messages, as shown in Fig. 11. One useful function of such an iPDA would be to find and recommend new friends with similar interests or promising partners in research or business to its owners, in terms of the accumulated communication records during a certain period. To implement that, each iPDA does three things: (1) selecting people who have communicated at least a certain number of times (i.e., above a threshold) as its initial local view; (2) finding the community containing its owner based on the AOC-d algorithm; (3) reporting new persons to its owner. For example, for the network shown in Fig. 11, 16 communities have been detected, with $\omega_1 = 0.3$ and $\omega_2 = 0.4$. For the user named Neumann in the circled community, its iPDA will recommend five new people to him/her as new partners, including Gao, Zhou, Zhu, Yu, and Roberts.

4.2 Performance analysis of the AOC-d algorithm

4.2.1 Emergent small-world property

Like the small-world model, the *clustering coefficient* of the agent network produced by the AOC-d algorithm increases during the updating process, and the finally evolved agent network can be highly clustered. From Fig. 12, we note that the *clustering coefficient* of different agent networks increases during their updating, and finally arrives at 1, corresponding to a set of separated cliques. In this experiment, we set $\omega_1 = 0.3$ and $\omega_2 = 0.2$ for all networks.

However, unlike the small-world model, in which an entire network is considered as one small world with high clustering coefficients, the agent network from the AOC-d algorithm is made up of a set of small worlds that are formed and separated gradually during the course

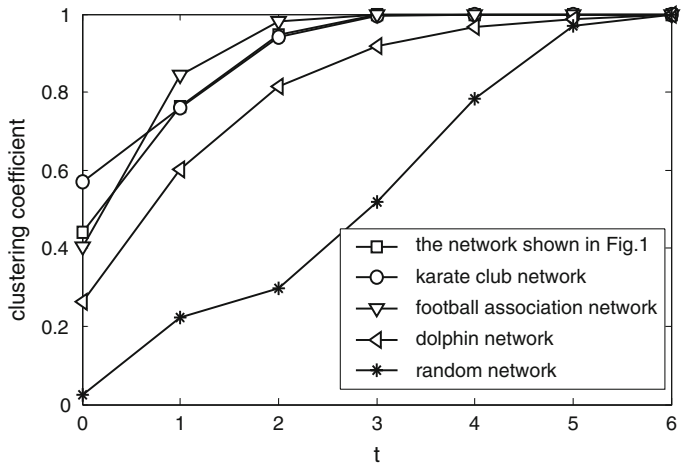

Fig. 12 Clustering coefficients of different evolving *AN*s

of updating. In the finally evolved agent network, multiple isolated small-world networks emerge, each of which is a clique with the highest clustering coefficient equal to 1. Such small-world networks are the network communities that we try to find.

4.2.2 Testing the clustering accuracy

Figure 13 presents a comparison of clustering accuracy among three algorithms, including: GN algorithm [7,20], Newman algorithm [19], and the AOC-d algorithm. The experimental network is $G(4, 32, 16, p_{in})$, a benchmark network that has been commonly used by many papers [7,19,20,23]. In this network, each node has *zin* edges connecting it to the members of the same group, and *zout* edges to the members of other groups, with the sum $z_{in} + z_{out} = 16$. The graphs with $z_{out} = k + 0.5$ are those in which half of nodes have k between-community links and half have $k + 1$. Algorithm is considered to be successful if each node is classified in the right community, and the four communities are not further subdivided. In Fig. 13, y-axis denotes the fraction of nodes correctly identified by these three algorithms, and each point in the curves is obtained by running corresponding algorithm over 200 graphs. In this experiment, we set $\omega_1 = 0.25$ and $\omega_2 = 0.2$.

From Fig. 13, we can see that all algorithms work very well when $z_{out} \leq 5.5$, correctly identifying more than 95% of nodes. In the case of $6 \leq z_{out} \leq 8$, the classification accuracy of the AOC-d algorithm is much better than the GN algorithm and the Newman algorithm.

4.2.3 Discussion on the parameters of the AOC-d algorithm

Two main parameters, ω_1 and ω_2, are involved in the AOC-d algorithm, which are used to control the granularity of communities. More communities with smaller sizes will be obtained under larger values of $\omega_1$and ω_2. Otherwise, fewer communities with larger sizes will be obtained. Figure 14 illustrates this fact using two networks, in which the z-axis denotes the number of communities obtained under different values of the two parameters. From this figure, we can see that the number of communities obtained by the AOC-d algorithm is related to but not sensitive to parameters when $0.2 \leq \omega_1, \omega_2 \leq 0.6$. In Fig. 14a, the number of communities obtained in the football association network falls into the interval of [25,30], and its

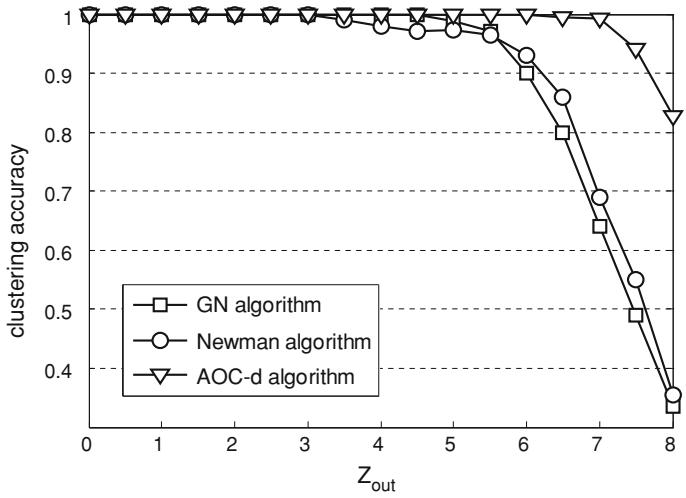

Fig. 13 Clustering accuracy of three algorithms

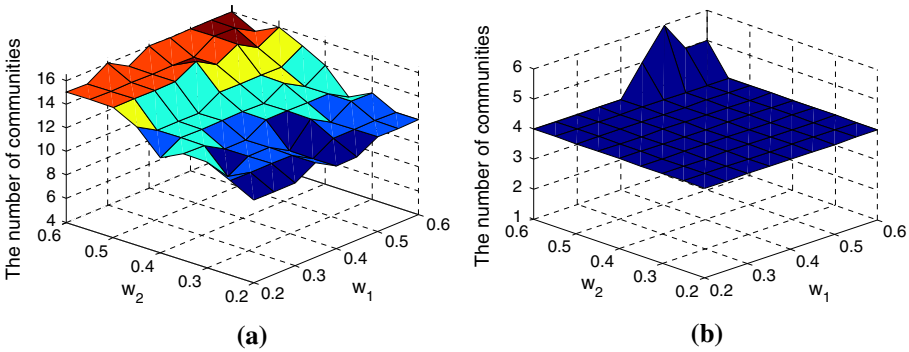

Fig. 14 The number of communities obtained by the AOC-d algorithm with different parameters. **a** The football association network, **b** G(4, 16, 16, 0.6)

average is 13, very close to the actual number of football teams. In Fig. 14b, the number of communities obtained in a random network with four communities varies within the interval of [23,27], and its average is 4.

4.2.4 Modularity evaluation

As presented by Newman [19,20], the concept of *modularity* has been used to evaluate the quality of a particular division of a network. Let us consider a particular division of a network into k communities. The modularity Q of this division is defined as follows:

$$Q = \sum_i (e_{ii} - a_i^2) \tag{4.1}$$

where e is a $k \times k$ symmetric matrix whose element e_{ij} is the fraction of all edges in the network that link the nodes in community i to the nodes in community j, and $a_i = \sum_j e_{ij}$. As pointed out by Newman, a higher modularity value indicates a better division for a network.

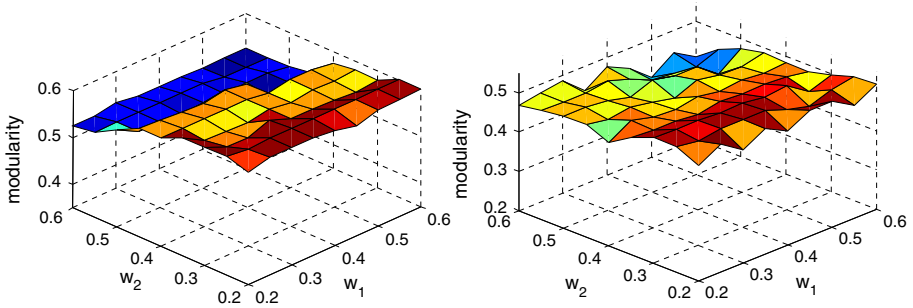

Fig. 15 The modularity obtained by the AOC-d algorithm with different parameters. **a** The football association network, **b** the dolphin network

From Fig. 15 in which the z-axis denotes the Q-value of the divisions obtained under different parameters, we can see that the modularity obtained by the AOC-d algorithm is related to but not sensitive to parameters when $0.2 \leq \omega_1, \omega_2 \leq 0.6$. In Fig. 15a, the obtained modularity of the football association network falls into the interval of [0.52, 0.603], and gets 0.601 at $\omega_1 = 0.3$ and $\omega_2 = 0.2$, as used in the experiment shown in Fig. 9. In Fig. 15b, the obtained modularity of the dolphin network falls into the interval of [0.43, 0.525], and gets 0.521 at $\omega_1 = 0.25$ and $\omega_2 = 0.2$, as used in the experiment shown in Fig. 10.

5 Incremental AOC-based method for mining dynamic networks

5.1 Problem definition

Usually, the network in a d-NCMP is dynamic, i.e., its structure will be updated periodically depending on new local updates (e.g., local interactions). Suppose N is a dynamic distributed network, and its updating process is represented by a discrete time series as follows:

$$N_0, N_1, \ldots, N_t, \cdots$$

where N_t denotes the updated network at time t. Let $CS(N_t) = \{NC_1, \ldots NC_k\}$ be the *community structure* of N_t, where $NC_i (1 \leq i \leq k)$ denotes the i-th network community and k is the total number of communities. Now, the problem is how to efficiently compute $CS(N_{t+1})$ after the network N_t is transformed into N_{t+1}.

The problem can be directly solved using the AOC-d algorithm as discussed in Sect. 3.4. Essentially, the method can be formalized as a mathematical function F, defined as:

$$CS(N) = F(N). \tag{5.1}$$

For a given network N, F can figure out its community structure $CS(N)$. So, after each update of the dynamic network, its new community structure can be computed as follows:

$$CS(N_{t+1}) = F(N_{t+1}). \tag{5.2}$$

Obviously, the strategy of re-calculating the entire network after each local update (e.g., interaction) is not efficient, especially when we process large-scale, locally-interacting, or highly-dynamic networks.

In this section, we hope to find an incremental function F^*, which can figure out the new community structure of a dynamic network based on its incremental update or change each time and its previous community structure that has been discovered. Formally, F^* can be defined as:

$$CS(N_{t+1}) = F^*(\Delta N_{t+1}, CS(N_t)) \tag{5.3}$$

where $\Delta N_{t+1} = N_{t+1} - N_t$ denotes the incremental change of the dynamic network N at time t+1.

5.2 Incremental AOC-based method

In the AOC-d algorithm, to find the community structure of a distributed network is to obtain a finally evolved agent network. In the incremental AOC-based method, named AOC-i algorithm, in which the network to be mined is dynamically changing (e.g., with local interactions), the evolution of the agent network will be evoked periodically. After each new update of the dynamic network, a new evolution of the agent network will be triggered to detect the new community structure of the updated network. Thus, the process of mining a dynamic network using the AOC-i algorithm is actually an iterative process consisting of a series of discrete evolutionary cycles, formalized as follows:

evolutionary cycle 0: $CS(N_0) = F(N_0)$;
evolutionary cycle 1: $CS(N_1) = F^*(\Delta N_1, CS(N_0))$;
… …
evolutionary cycle l: $CS(N_l) = F^*(\Delta N_l, CS(N_{l-1}))$;
… …

In each evolutionary cycle, the new community structure can be quickly derived based on the incremental change and the old community structure as obtained in the previous cycle. The life-cycle of agent p in the AOC-i algorithm is given in Table 6.

$v_p.NL(l)$ denotes the neighbor configuration of node v_p in the evolutionary cycle l. If the incremental change, ΔN_l, happens to node v_p, we have $v_p.NL(l) \neq v_p.NL(l-1)$. Otherwise we have $v_p.NL(l) = v_p.NL(l-1)$. In the former case, the view of agent p will be initialized as $v_p.NL(l)$, the new neighbor configuration of node v_p. While in the latter case, the view of agent p will be initialized as its final view in the previous evolutionary cycle. $agent_p.\psi(l)$ denotes the objective obtained in the evolutionary cycle l. Except for the initialization phase, the life-cycle of an agent in the AOC-i algorithm is almost the same as that in the AOC-d algorithm, as shown in Table 5.

Example 5.1 In this example, we will show how to mine a simple dynamic network using the AOC-i algorithm. The network discussed here is the network shown in Fig. 1. What is different in this experiment is that the network is considered as a dynamic network that grows gradually. In each update of the network, five new nodes and the links from them are added. The entire process contains eight evolutionary circles. Figure 16a–h present the snapshots of the agent network after each evolutionary cycle, respectively. In the figures, triangular nodes denote inactive agents. The final agent network from the entire process is shown in Fig. 16h, from which we can see that the community structure found by the AOC-i algorithm is exactly the same as that found by the AOC-d algorithm, as shown in Fig. 6h.

Table 6 The life-cycle of $agent_p$ in the evolutionary cycle l

/* initialization phase */
1. $v_p.DP \leftarrow \{\}$;
2. $agent_p.t \leftarrow 0$;
3. if $v_p.NL(l) = v_p.NL(l-1)$ then
4. $\quad agent_p.view(0) \leftarrow \{<p, q, 1> | q \in agent_p.\psi(l-1) \wedge q \neq p\}$;
5. else
6. $\quad agent_p.view(0) \leftarrow v_p.NL(l)$;
7. end
8. $v_p.DP \leftarrow v_p.DP \cup agent_p.view(0)$;
/* active phase */
9. while $agent_p.t < T$ do
10. $\quad agent_p.evaluate\_view(agent_p.t)$;
11. $\quad agent_p.shrink\_view(agent_p.t + 1)$;
12. $\quad agent_p.enlarge\_view(agent_p.t + 2)$;
13. $\quad agent_p.balance\_view(agent_p.t + 3)$;
14. $\quad agent_p.t \leftarrow agent_p.t + 4$;
15. $\quad$ if $agent_p.view(agent_p.t) = agent_p.view(agent_p.t - 4)$ then
16. $\quad\quad$ goto 19;
17. $\quad$ end
18. end
/* inactive phase */
19. $agent_p.\psi(l) = \{v_i | (p, i, w_{pi}) \in agent_p.view(agent_p.t)\} \cup \{v_p\}$;
20. $agent_p.sleep$;

In the same way, we have validated the AOC-i algorithm using all benchmark networks as discussed in Sect. 4, and are able to find the correct community structures in all cases.

5.3 Performance analysis of the AOC-i algorithm

Let t_p be the time steps required by agent p to go through its life-cycle in the evolutionary cycle l. The total time steps required by all agents in this cycle can be calculated as follows:

$$G_l = \sum_{p=1}^{n} t_p \tag{5.4}$$

where n is the number of agents.

As discussed in Sect. 3.4.3, each agent will take $O(t_p \cdot d^2)$ time to run through its life-cycle, where d is the average degree of the agent network at certain time. So the total time required by one evolutionary cycle is $O(G_l \cdot d^2)$. That is, we can approximately evaluate the efficiency of the AOC-i algorithm using G_l. Smaller value of G_l indicates that less time is required in the evolutionary cycle l.

Based on this idea, we validate the performance of the AOC-i algorithm using several benchmark networks. In the experiments, for each network, we use two different strategies to find out its community structures. The first one is based on Eq. 5.2, i.e., after a network is updated, the new network, including the original part and the incremental change, will be

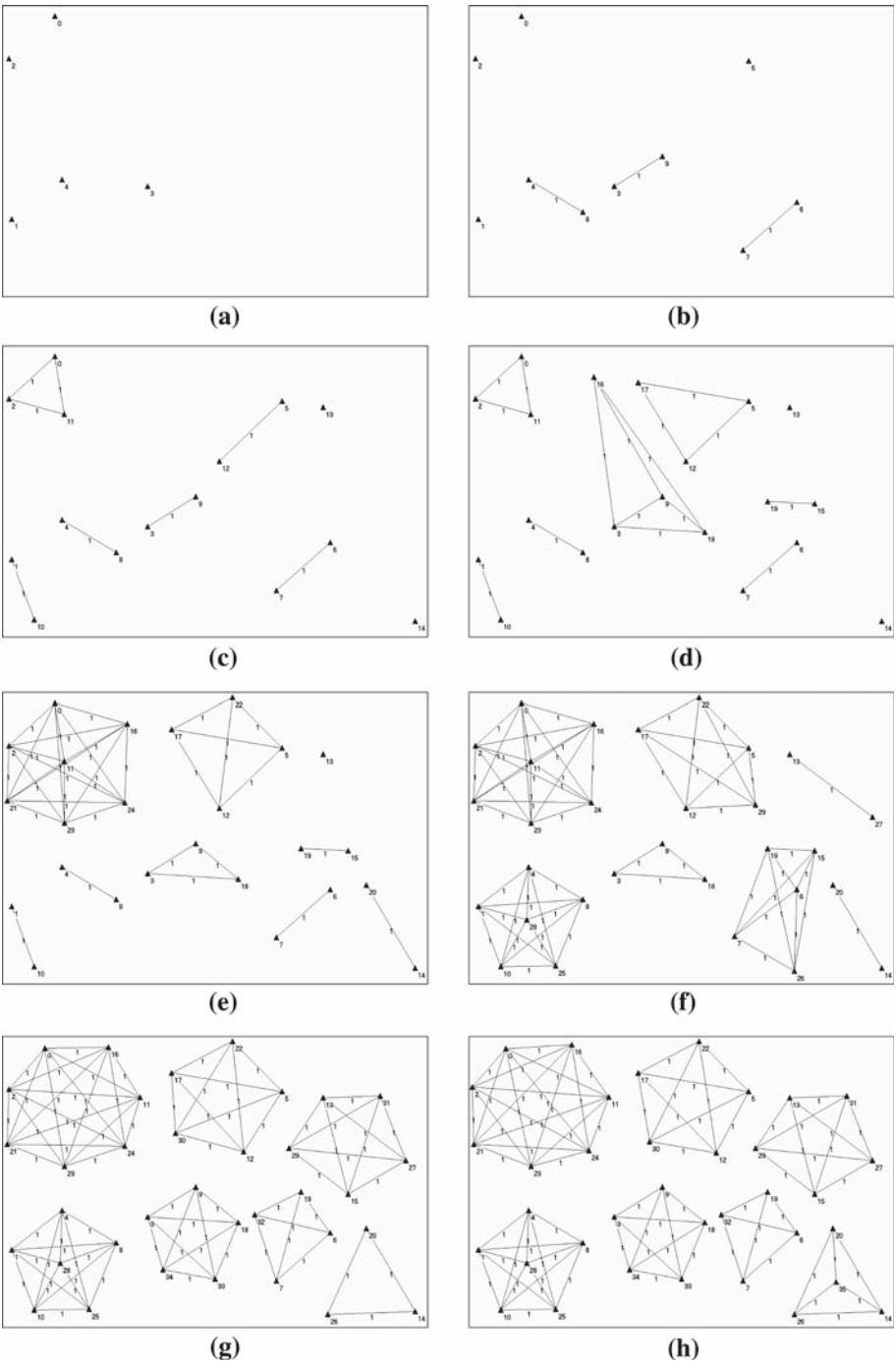

Fig. 16 The process of mining a dynamic network using the AOC-i algorithm. **a** Evolutionary cycle 0: $n = 5$, **b** evolutionary cycle 1: $n = 10$, **c** evolutionary cycle 2: $n = 15$, **d** evolutionary cycle 3: $n = 20$, **e** evolutionary cycle 4: $n = 25$, **f** Evolutionary cycle 5: $n = 30$, **g** evolutionary cycle 6: $n = 35$, **h** evolutionary cycle 7: $n = 36$

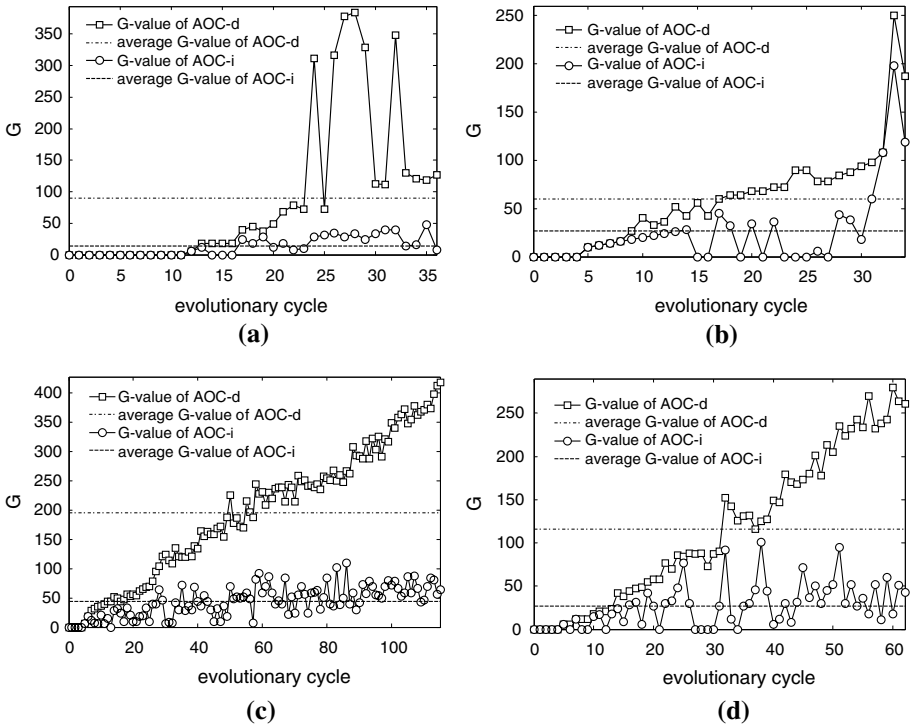

Fig. 17 Validating the performance of the AOC-i algorithm. **a** The network shown in Fig. 1, **b** the karate club network, **c** the football association network, **d** the dolphin network

input into the AOC-d algorithm to re-calculate it once again. The second one is based on Eq. 5.3, i.e., after a network is updated, its new community structure will be detected by the AOC-i algorithm based on the incremental change and the previously discovered community structure.

Again, all networks used here are considered as dynamic ones, which grow gradually by adding one new node and its associated links in each update. The experimental results are given in Fig. 17. From the results, we can see that as compared with the AOC-d algorithm without the incremental feature, the AOC-i algorithm works efficiently and requires much less time to deal with the new network after each update.

It should be mentioned that the process of mining a dynamic network using the AOC-i algorithm is essentially an iterative process consisting of a series of evolutionary cycles. If the updating speed of a dynamic network is matched with the average convergent speed of each evolutionary cycle, the performance of the AOC-i algorithm will be good without any delay. Otherwise, updates will be aggregated and be processed as a batch in the next cycle, which can result in some delay. We have tested the average convergent speed of each evolutionary cycle with the benchmark networks, as shown in Fig. 17. From these figures, we can see that in all cases, the average convergent speed of each cycle is less than $50\,d^2$. With these figures, we can quantitatively analyze how fast a network can update without strong negative effects (delays) on the performance of the AOC-i algorithm.

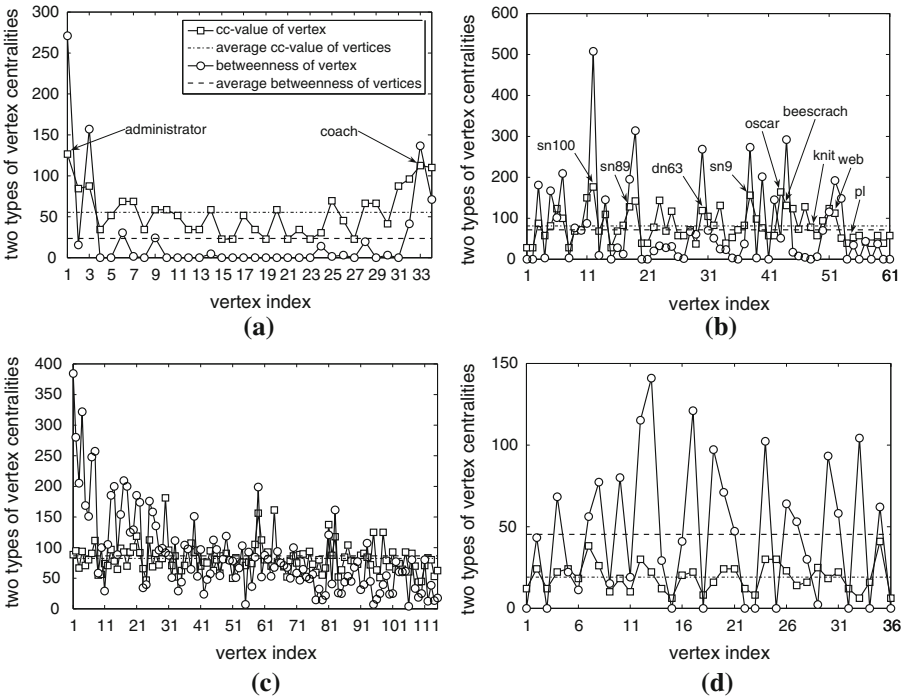

Fig. 18 Clustering centrality and betweenness centrality of nodes in different networks. **a** The karate club network, **b** the dolphin network, **c** the football association network, **d** the network shown in Fig. 1

5.4 The centrality-of-node problem

Now, let us take a look at a centrality-of-node problem. Given a network, we first find its community structure using the AOC-d algorithm. Then, we select one node, and delete it and its associated links from the network. Finally, we re-classify the remained network using the AOC-i algorithm. Through experiments, we note that some nodes that play special roles in the original network will bring greater disturbance to the network, once they are taken away from the network, and thus it will take more efforts to find the new community structure of the remained network.

As inspired by this observation, we now use the AOC-i algorithm to approximately evaluate the influence of a particular individual on the structural stability of the entire network.

5.4.1 The clustering centrality

The *clustering centrality* of a node v, denoted as $cc(v)$, is defined as the total time steps required by all agents in a new evolutionary cycle evoked by deleting it and its associated links from a network that has been clustered using the AOC-d algorithm.

Figure 18 presents the clustering centrality of each node in four networks. In the experiments, we set $\omega_1 = 0.25$ and $\omega_2 = 0.3$ for all networks. From these figures, we can see that nodes with higher *cc*-values above the average are likely the centers or the brokers of the communities, while those with lower *cc*-values below the average are likely the peripheral ones. Brokers are those nodes that are situated at the common boundaries between different communities.

For example, in Fig. 18a, the two individuals with the highest *cc*-values are, respectively, the centers of two communities of the karate club network as shown in Fig. 8a. Other 14 individuals with higher *cc*-values above the average are all brokers of the network except for 6, 7, and 25. All of the 18 individuals with lower *cc*-values below the average are the peripheral nodes of the network.

For another example, the brokers of the dolphin network as shown in Fig. 10a are sn100, oscar, sn89, sn9, pl, knit, dn63, beescratch, and web. As shown in Fig. 18b, most of them get higher *cc*-values above the average except for knit and pl. The nodes with lower cc-values below the average, such as 0_five, 1_smn5, 7_vau, 14_cross, 19_fork, 20_whitetip, 28_ripplefluke, 53_tr82, 57_zig, and 59_quasi, are the peripheral ones of the network. From Fig. 18c and d, we can also make similar observations in the cases of the football association network and the network of Fig. 1.

Brokers usually play bonding roles in maintaining the stability of the community structure of a network. If main brokers with the highest clustering centrality are destroyed, the entire network will be broken down with a high probability. For example, as argued by Lusseau [16], the dolphin network was split into two parts due to the disappearance of the dolphin SN100, the individual with the highest *cc*-values, as shown in Fig. 18b.

5.4.2 The betweenness centrality

Betweenness is another centrality measure widely used, which was introduced by Freeman [6] to measure the influence of individuals over the information flow in a network. By measuring the betweenness centrality, we know that some nodes with higher values will be passed through by more information than the others.

Figure 18 presents the measures of the clustering and betweenness centralities for each node in four networks, from which we observe that the two measures are related, to some extent, but different in essence. Their interrelation can be quantitatively analyzed by calculating their *correlation coefficient*. By standard definition, the correlation coefficient of two measures, X and Y, is given as follows:

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sqrt{DX}\cdot\sqrt{DY}} = \frac{E((X-EX)(Y-EY))}{\sqrt{DX}\cdot\sqrt{DY}} \tag{5.5}$$

where $-1 \leq \rho \leq 1$. If the two measures have a higher correlation coefficient, then they are more likely to be correlated to each other. Table 7 gives the correlation coefficients between the two centrality measures in different networks.

From Table 7, we can see that the two measures are highly correlated in the karate club and dolphin networks, i.e., two networks that represent the *interactions between social individuals*. In other words, for an individual in such networks, if it has a higher/lower value of one centrality, it will also have a higher/lower value of another centrality, accordingly.

Table 7 The correlation coefficients between two centrality measures

Networks	$\rho(cc, betweenness)$
Karate club network	0.71
Dolphin network	0.61
Football association network	0.32
Network shown in Fig. 1	0.42

Based on this observation, a reasonable proposition can be made, that is, the distribution of information flow in such societies is not even but highly related to the underlying social structures. Usually the density of information flow in the centers or the common boundaries of communities is much higher than that in other areas.

6 Conclusions

In this paper, we have presented a distributed network community mining problem (called d-NCMP), and have demonstrated how an AOC-based method can be effectively developed and applied in solving d-NCMPs. In this method, the nodes and links of distributed networks are distributed among a group of autonomous agents, who are responsible for finding all communities hidden in distributed networks based on their respective local views. Agents in the AOC-based method act asynchronously without any global control, while being able to collectively achieve their objectives based on some autonomy-oriented, self-organized computing principles, such as positive feedback. For further discussions on the basic constructs, desirable characteristics, and necessary and sufficient conditions of scalable, self-organized computing with an AOC system, readers are referred to [13].

Furthermore, we have described how the AOC-based method can be extended to a more efficient incremental AOC-based method for mining communities from dynamic and distributed networks. We have validated these methods using several commonly used benchmark networks as well as some synthetic, distributed and dynamic networks as generated by simulations. Our experimental results have shown their effectiveness and good performances. In addition to autonomous mining of network communities, we have also used the AOC-based method to evaluate the centrality of network nodes, and have found a very interesting correlation between the distribution of information flow in some networks and their community structures.

Several extensions can readily be pursued based on our present work. For instance, considering the applications of our AOC-based method to solving large-scale, real-world problems, it would be desirable to study how the local communication cost involved can be further minimized, and how stable and robust community structures can be collectively found and maintained in a distributed and dynamic environment (e.g., where communications may not always be reliable). On the theoretical side, it would be interesting to further study the general models and principles of autonomy-oriented, self-organized computing (e.g., [13–15]), answering such questions as: how to asynchronously and autonomously self-organized *globally optimal* solutions, i.e., most utility/cost-aware, stable, and/or robust structural patterns or emergent behaviors in *open* or dynamic environments, how to efficiently do so with *lightweight* reactive agents, whose behaviors may be *dynamically evolved* or organized, and what are the roles of short- and long-range exploration and collective regulation (without global information).

Acknowledgements This work was supported in part by the National Natural Science Foundation of China Grants (60873149, 60503016, and 60496321) and the National High-Tech Research and Development Plan of China (2006AA10Z245), and in part by a Hong Kong Baptist University FRG grant (06-07-II-66).

References

1. Bae, K., & Yoon, H. (2005). Autonomous clustering scheme for wireless sensor networks using coverage estimation-based self-pruning. *IEICE Transactions on Communications, E88-B*(3), 973–980.

2. Bui, T. N., & Rizzo, J. R. (2004). Finding maximum cliques with distributed ants. In *Proceedings of the conference on genetic and evolutionary computation* (pp. 24–35). Springer.
3. Chakrabarti, S., Van Der Berg, M., & Dom, B. (1999). Focused crawling: A new approach to topic-specific web resource discovery. In *Proceedings of the eighth international conference on world wide web* (pp. 1623–1640). ACM Press.
4. Fiedler, M. (1973). Algebraic connectivity of graphs. *Czechoslovakian Mathematical Journal, 23*(2), 298–305.
5. Flake, G. W., Lawrence, S., Giles, C. L., & Coetzee, F. M. (2002). Self-organization and identification of web communities. *IEEE Computer, 35*(3), 66–71.
6. Freeman, L. C. (1977). A set of measures of centrality based upon betweenness. *Sociometry, 40*(1), 35–41.
7. Girvan, M., & Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of National Academy of Science (PNAS), 99*(12), 7821–7826.
8. Jennings, E., & Motyckova, L. (1992). A distributed algorithm for finding all maximal cliques in a network graph. In *Proceedings of the first Latin American symposium on theoretical informatics* (pp. 281–293). Springer.
9. Jünger, M., & Mutzel, P. (2003). *Graph drawing software (mathematics and visualization)*. Springer.
10. Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Technical, 49*(1), 291–307.
11. Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of ACM, 46*(5), 604–632.
12. Kumar, R., Raghavan, P., Rajagopalan, S., & Tomkins, A. (1999). Trawling the web for emerging cyber-communities. In *Proceedings of the eighth international conference on world wide web* (pp. 1481–1493). ACM Press.
13. Liu, J. (2008). Autonomy-Oriented Computing (AOC): The nature and implications of a paradigm for self-organized computing. In *Proceedings of the fourth international conference on natural computation* (Keynote talk) (pp. 3–11). IEEE Computer Society.
14. Liu, J., Jin, X. L., & Tsui. K. C. (2004). *Autonomy oriented computing*. Springer/Kluwer Academic Publishers.
15. Liu, J., Jin, X. L., & Tsui. K. C. (2005). Autonomy oriented computing (AOC): Formulating computational systems with autonomous components. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 35*(6), 879–902.
16. Lusseau, D. (2003). The emergent properties of a dolphin social network. *Proceedings of the Royal Society of London—Series B, 270*(Suppl. 2), S186–S188.
17. Meka, A., & Singh, A. K. (2006). Distributed spatial clustering in sensor networks. In *Proceedings of the tenth international conference on extending database technology* (pp. 980–1000). Springer.
18. Newman, M. E. J. (2004). Detecting community structure in networks. *European Physical Journal B, 38*(2), 321–330.
19. Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Physical Review E, 69*(6), 066133.
20. Newman, M. E. J., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E, 69*(2), 026113.
21. Pirolli, P., Pitkow, J., & Rao, R. (1996). Silk from a sow's ear: Extracting usable structures from the web. In *Proceedings of the ACM conference on human factors in computing systems* (pp. 118–125). ACM Press.
22. Pothen, A., Simon, H., & Liou, K. P. (1990). Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis and Application, 11*(3), 430–452.
23. Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., & Parisi, D. (2004). Defining and identifying communities in networks. *Proceedings of National Academy of Science (PNAS), 101*(9), 2658–2663.
24. Scott, J. (2000). *Social network analysis: A handbook* (2nd ed.). Sage Publications.
25. Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligent, 22*(8), 888–904.
26. Tyler, J. R., Wilkinson, D. M., & Huberman, B. A. (2003). Email as spectroscopy: Automated discovery of community structure within organizations. In *Proceedings of the first conference on communities and technologies* (pp. 81–96). Kluwer Academic Publishers.
27. Wilkinson, D. M., & Huberman, B. A. (2004). A method for finding communities of related genes. *Proceedings of National Academy of Science (PNAS), 101*(1), 5241–5248.
28. Wokoma, I., Shum, L. L., Sacks, L., & Marshall, I. (2005). A biologically-inspired clustering algorithm dependent on spatial data in sensor networks. In *Proceedings of the second European workshop on wireless sensor networks* (pp. 386–390). IEEE Computer Society.

29. Wu, F., & Huberman, B. A. (2004). Finding communities in linear time: A physics approach. *European Physical Journal B, 38*(2), 331–338.
30. Yang, B., Cheung, W. K., & Liu, J. (2007). Community mining from signed social networks. *IEEE Transactions on Knowledge and Data Engineering, 19*(10), 1333–1348.
31. Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of Anthropological Research, 33*(4), 452–473.