Cluster Comput
DOI 10.1007/s10586-015-0452-x

Community structure mining in big data social media networks with MapReduce

Songchang Jin[1] · **Wangqun Lin**[2] · **Hong Yin**[1] · **Shuqiang Yang**[1] · **Aiping Li**[1] · **Bo Deng**[2]

Received: 14 March 2014 / Revised: 8 March 2015 / Accepted: 6 April 2015

Abstract Social media networks are playing increasingly prominent role in people's daily life. Community structure is one of the salient features of social media network and has been applied to practical applications, such as recommendation system and network marketing. With the rapid expansion of social media size and surge of tremendous amount of information, how to identify the communities in big data scenarios has become a challenge. Based on our previous work and the map equation (an equation from information theory for community mining), we develop a novel distributed community structure mining framework. In the framework, (1) we propose a new link information update method to try to avoid data writing related operations and try to speedup the process. (2) We use the local information from the nodes and their neighbors, instead of the pagerank, to calculate the probability distribution of the nodes. (3) We exclude the network partitioning process from our previous work and try to run the map equation directly on MapReduce. Empirical results on real-world social media networks and artificial networks show that the new framework outperforms our previous work and some well-known algorithms, such as Radetal, FastGN, in accuracy, velocity and scalability.

Keywords Social media · Community structure mining · MapReduce

✉ Songchang Jin
jsc04@126.com; jinsongchang87@gmail.com

Wangqun Lin
linwangqun2005@gmail.com

Hong Yin
yinhonggfkd@aliyun.com

Shuqiang Yang
sqyang9999@126.com

Aiping Li
apli1974@gmail.com

Bo Deng
bodeng@vip.tom.com

[1] College of Computer, National University of Defense Technology, Changsha 410073, China

[2] Beijing Institute of System and Engineering, Beijing 10010, China

1 Introduction

Social media network (SMN) has been booming now. BI Intelligence [6] recently reported that, monthly active users (MAU) of FaceBook have reached 1.16 billion by the end of 2013, YouTube has about 1 billion MAUs, and Qzone, a China's giant social media network, is running in third place at 712 million users.

In the study of complex networks, if the nodes of the network can be grouped into components such that nodes in each component are densely connected internally via links or edges and connections among different components are sparse, the network is said to have *community structure*. Nodes in each component along with the links among the nodes constitute a *community*. Studying the community structure has important theoretical and practical value [9,10], such as information propagation research, product recommendation, link prediction and so on. Given a SMN, the community structure is not manifest, and the communities are hidden in the complex structure. How to identify all the communities is the issue that mining community structure devotes oneself to solving, especially in big data scenarios.

We have made attempts to reveal the community structure of big data SMNs in [13,14]. In our previous work, we (1) divide the network into plenty of subnetworks and the size of each subnetwork should smaller than the block

Published online: 13 May 2015

size; (2) upload the subnetworks onto the Hadoop distributed file system (HDFS) [2]; (3) run MapReduce based infomap method to process the subnetworks independently. Limitation in this solution lies in the graph partitioning process. Because network partitioning is NP-complete [1], and current graph partitioning methods are not able to find the optimal solution in linear time. The links cut off by the graph partitioning methods may be far more than the optimal, and may assign the nodes of a community into several blocks which will destroy the community structure. In our previous work, we aim at accelerating the map equation [30] (shown in Expression (5)) using MapReduce programming model.

In this paper, we extend our previous work: (1) networks are represented and stored on HDFS in adjacency list format; (2) there is no network partitioning process and network integrity is preserved; (3) we only read data from HDFS and there is no data writing operations, such as append, modify and write, before outputting final results, which coincides the HDFS's WORM access model and significantly reduces the overhead of writing data to the disk; (4) it is easy-to-use and has high scalability, which makes the community mining in big data SMNs with MapReduce [7] cluster easier.

The rest of this paper is organized as follows. Section 2 presents terms, definitions and problem formulation, followed by the details of the distributed community structure mining framework. In Sect. 4, we conduct a couple of experiments to evaluate the performance of the framework. We discuss some related work in Sect. 5. Finally, Sect. 6 provides some concluding remarks and outlines future research directions.

2 Problem formulation

2.1 Terms, assumption and definitions

A *SMN* can be mathematically described as $G = (V, E)$, where V and $E \subseteq V \times V$ represent the node set and link set, respectively. $n = |V|$ and $m = |E|$ are the total number of nodes and links in G. $v(i)$ represents node i and $e(i, j) \in E$ indicates that there is a link between $v(i)$ and $v(j)$. $nv(i)$ stands for the neighbor node set of $v(i)$ and $d(i) = |nv(i)|$ represents the degree of $v(i)$.

By now, community definition has not been uniformed in scientific world. The most approved one is proposed by Yang [33]:

Definition 1 (*community*) a *community* within a network is a group of nodes, within which the links connecting nodes are dense but between which they are sparse.

Definition 2 (*intra link & inter link*) in a community structured network, a link connecting nodes within the same community is called an *intra link*, and the links connecting nodes from different communities are *inter links*.

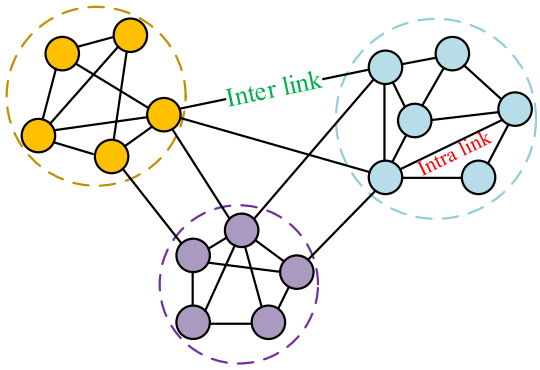

Fig. 1 An example of inter links and intra links in a community-structured network with 3 communities surrounded by the *dashed circles*

From Fig. 1, we can see that nodes within a community are densely connected by intra links and nodes of different communities are sparely connected by inter links. Therefore, for a community structured network, it is reasonable to view it as a collection composed of a series of communities and a number of inter links.

For the convenience of following discussion, we define some community related items in Table 1. In SMN data set, link number is far more than node number, i.e. $m \gg n$. Generally, we can store the node IDs in memory but we cannot store the links in memory. Assuming that there is 100 million nodes with average degree 100, each node needs 4 bytes and each links need at least 8 bytes to express in memory, then will can use 400 MB to record all the node IDs, however the memory consumption will be more than 8 GB for the links. Therefore, we assume that the node IDs of SMNs studied in this paper can resident memory. Besides, for simplicity, we assume that the networks studied here are connected.

2.2 Community structure mining problem

A community-structured SMN can be seen as a combination of a series of communities and a number of inter links. Mathematically, we represent the series of communities as a set $CS = \{C_1, C_2, \ldots, C_{NC}\}$, where $C_i = \{v(i1), v(i2), \ldots, v(in_i)\}$ means a community of n_i nodes and NC is the total number of communities in the SMN.

Community structure mining aims to solve the problem of how to identify all the communities in the big data SMNs as fast and accurately as possible. Networks studied in this paper are undirected and unweighted. More exactly, our object will be finding out the community set $CS = \{C_1, C_2, \ldots, C_{NC}\}$, where $C_i \cap C_j = \phi$ for $i \neq j$. However, without quantitative indicators, we will not be able to present an intuitive grasp of the problem. Therefore, with the help of map equation,

Table 1 Symbol definition and description

Symbol	Description
$d(C_i)$	$\sum d(j)$, where $v(j) \in C_i$, sum of degrees of nodes in C_i
$outLink(C_i)$	$\lvert \cup e(s,t) \rvert$, where $v(s) \in C_i$ and $v(t) \notin C_i$, sum of links departing from C_i
$inLink(C_i)$	$\lvert \cup e(s,t) \rvert$, where $v(s) \in C_i$ and $v(t) \in C_i$, sum of links within C_i
$interLink(C_i, C_j)$	$\lvert \cup e(s,t) \rvert$, where $v(s) \in C_i$ and $v(t) \in C_j$
$inDegree(C_i)$	$2*inLink(C_i)$, sum of degrees within C_i
$outDegree(C_i)$	$outLink(C_i)$, sum of out degrees of nodes of C_i

we convert the community structure mining problem into a mathematical optimization problem in Sect. 3.

3 Supporting theory and proposed framework

3.1 MapReduce framework

MapReduce has become an important big data processing tool recently. There are two procedures in MapReduce programs: Map() and Reduce(). The former performs filtering and sorting, and Reduce() performs a summary operation. Servers in MapReduce platform are organized in master/workers architecture. The master is responsible for interacting with users. It receives programs and splits them into map works and reduce works, then assigns the works to the distributed servers (called as mappers and reducers according to the work type). Workers can only communicate with the master via the heartbeat protocol.

Data processed in Map() and Reduce() should be in $\langle key, value \rangle$ pairs. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain: $\langle key1, value1 \rangle \rightarrow list\langle key2, value2 \rangle$. Map is applied in parallel to every pair in the input data set, and will generates a list of pairs for each call. Then, the framework collects all pairs with the same key from all lists and groups them together, creating one group for each key. Next, reduce is applied in parallel to each group, which in turn produces a collection of values in the same domain: $\langle k2, list(v2) \rangle \rightarrow \langle k2, list(v3) \rangle$.

3.2 Information coding theory

Information coding is a concept in electrocommunication field. Essentially, coding theory can be divided into two aspects according to purpose: source code and channel code. Source encode attempts to compress the data from a source in order to transmit it more efficiently [23], and it is that we are using. Channel code, by adding extra data bits, aims to make the transmission of data more robust even if some bits lost or tampered.

Source coding contains two code patterns: fixed-length code and variable-length code. Given an encoded message, the most important thing is to decode it to get the original information and the information should be unique. Fixed-length codes are always uniquely decipherable. However, the situation for variable-length codes is completely different. So, some useful regulations are proposed, of which prefix code rule is an important one:

Definition 3 (*prefix code*) A code is called a prefix (free) code if no codeword is a prefix of another one. for example, $\{a = 0, b = 110, c = 10, d = 111\}$ is a prefix code.

Prefix free code is uniquely decipherable code. Since no codeword is a prefix of any other we can always find the first codeword in a message, peel it off, and continue decoding. Therefore developing high efficiency prefix free code interests researchers. Given an alphabet $A = \{a_1, \ldots, a_n\}$ with probability distribution $p(a_i)$, expected value of bits per codeword needed for encoding a message of $n \sum_{a=1}^{n} p(a)$ characters by a binary prefix code C is:

$$B(A) = \sum_{i=1}^{n} p(a_i) * l(c(a_i)) \tag{1}$$

where $c(a_i)$ is the codeword for encoding a_i, and $l(c(a_i))$ is the length of the codeword $c(a_i)$.

Explicitly, how to develop a code minimizing $B(C)$ is the key problem. Huffman developed a minimum-cost (optimum) prefix code (Huffman code) with a greedy algorithm [11].

$Entropy$ is another important concept in information theory [12]. It is a measure of the uncertainty in a random variable. The entropy of A used above will be:

$$H(A) = \sum_{i=1}^{n} p(a_i) \log_2 \frac{1}{p(a_i)} \tag{2}$$

Kraft-McMillan theorem Let C be a code with n codewords with length $l(1), l(2), \ldots, l(n)$. If C is uniquely decodable, then

$$\sum_{i=1}^{n} 2^{-l(i)} \leq 1 \tag{3}$$

From (1), (2) and (3), we can deduce that the average length of codeword for an optimal prefix-free code C satisfies:

$$H(A) \leq B(A) \leq H(A) + 1 \tag{4}$$

Expression (4) means that the average bits per word $B(A)$ in Huffman code is bounded below by $H(A)$. Generally, for

a specified network with specified probability distribution, when encoded with global Huffman code, B will be a constant. Next, we will discuss two level Huffman code on a network.

3.3 Random walk and map equation

A random walk on a network can be seen as the trace of the walker at discrete time points. Assuming the position of the walker at time t is $s(t) = v(i)$, then, it will choose a node $v(j) \in nv(i)$ randomly to move.

Random walk can be used as a standard tool for modelling spreading processes in social media [28] and map equation just uses it as a proxy for real flow. SMN products community structure for the aggregation effect caused by users' establishing connections with others. When start a random walk on a social media network, it will imitate people's behaviors and would like to stay in a local community for a long when entering. Take $v(6)$ in Fig. 2 for example, when the walker want to leave, it will choose a node from $nv(6) = \{v(5), v(7), v(8), v(9)\}$ randomly with the same probability of 25 %. Nodes $v(6)$, $v(7)$, $v(8)$, $v(9)$ are of the same community, then the probability of the walker staying in the current community in next step will be 80 %. Next, if the walker choose $v(9)$, the probability will be 75 %, and so on. With this characteristic, we will be able to use a random walker as an agent to reveal the network structure and community structure [29].

From the data management point of view, multidimensional data management model is more efficient than low-dimensional model to some extent. Actually, multi-level data organization structure is more common rather than flat structure in real world. Recalling the administrative division in real life, we use hierarchical description to tell our friends where we are or where we live. In this way, low levels may share the same description. For example, we may use a hierarchical description model "country name + state name + city name + street name + building number" to specify a particular address. People living in the same city share the same country name and state name, therefore hierarchical description model is high efficient. Viewed in connection with SMNs, if given all the community information, we can use the similar way "community ID + node ID" ($CID + NID$), a two level description model, to specify a particular node. Nodes of different CID can share the same NID and nodes of the same CID share the same CID. When using random walk to reveal the community structure, we can generate a codeword sequence by the two-level model to describe the walker's trace.

The core of the community detection is based on the map equation proposed by Rosvall et. al. [30], which uses a two level Huffman code to encode the network. The first level encodes the CIDs and the second level encodes the NIDs.

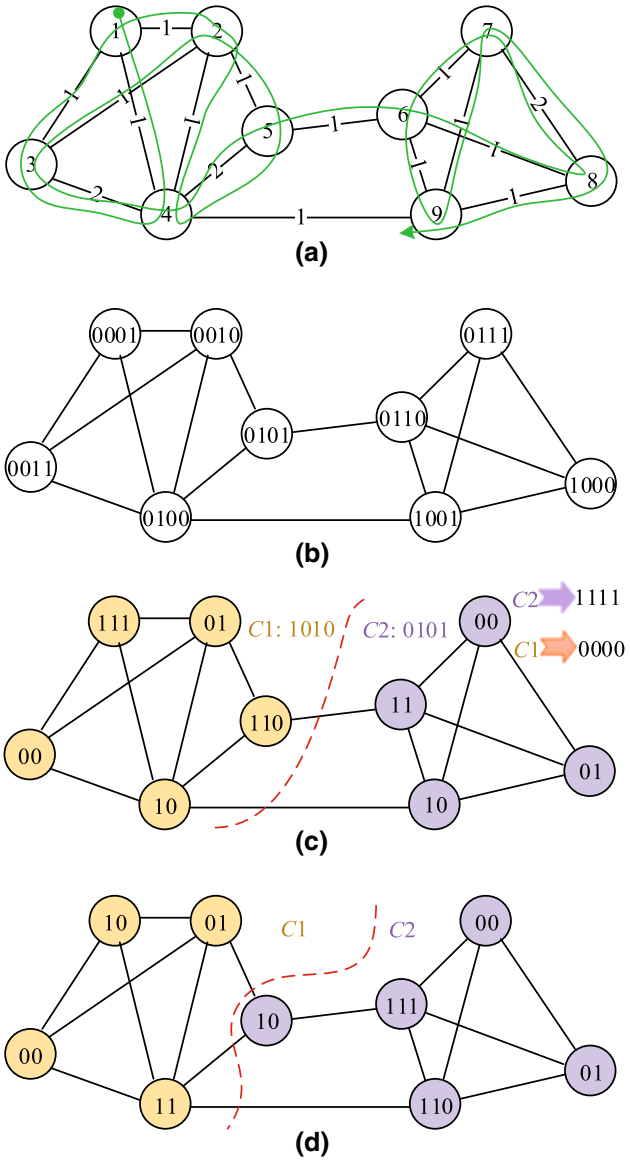

Fig. 2 A random walk on a weighted social media network of 9 nodes and with different information codes. **a** shows a random walk process with 18 steps from $v(1)$ to $v(9)$. **b** shows a global fixed-length binary code on the network. **c**, **d** are of two level Huffman code with different community structure divisions. "1010" and "0101" are the codewords of C_1 and C_2, corresponding exit codewords are "0000" and "1111". In the case (**c**), the codeword sequence will be "*1010* 10 00 111 01 110 10 01 00 10 110 0000 *0101* 11 01 00 11 10 00 01 10" and 51 bits will be totally needed. If community structure are divided as (**d**), the sequence will be "*1010* 10 11 00 10 01 11 0000 *0101* 10 1111 *1010* 01 00 11 0000 *0101* 10 111 01 00 111 110 00 01 11" with 69 bits

With this model and given a community partition pattern, we are able to uniquely describe where a particular node is. In consideration of coding efficiency and the feature that random walker prefers to stay in the community for a long time after entering, we (1) place a CID and a NID in the codeword sequence when the walker enters a new community to declare the walker has moved to a certain node of a

new community, (2) place a NID and a exit codeword when it leaves from current community, and (3) just use NID to describe the walker's position, because it shares the same CID with previous node. Then, the sequence will be "$CID\ NID \ldots NID\ exit\ codeword\ CID\ NID \ldots$". Map equation describes the average bits needed to describe a single step in the sequence. For a given social media network G with community partition pattern D of t communities, the average length of codeword Lc to describe a step in a steady random walk process on G with the two level Huffman code is:

$$Lc(D) = q_{out} H(Q) + \sum_{i=1}^{t} p^i H(P^i) \tag{5}$$

In Expression (5), there are two parts. The first part, $q_{out} * H(Q)$, means the average bits needed to describe the CIDs in each random walk step (bits of NID description needed of this situation is moved into the second part and combined with the first two cases). $q_{out} = \sum_{i=1}^{k} q_{i,out}$, where k represents the number of communities in current community partition pattern and $q_{i,out}$ means the probability of leaving from C_i. $H(Q)$ represents the entropy of the CIDs in current community partition pattern. According to the Shannon's source coding theorem [31], $H(Q)$ can be expressed as:

$$H(Q) = -\sum_{i=1}^{k} \frac{q_{i,out}}{\sum_{j=1}^{k} q_{j,out}} \log_2 \frac{q_{i,out}}{\sum_{j=1}^{k} q_{j,out}} \tag{6}$$

Figure 2c, d are examples of the two level description model. In Fig. 2c, to describe a step in the random walk, $\frac{51}{18} = 2.83$ bits will be averagely needed. However, $\frac{69}{18} = 3.83$ bits is needed in Fig. 2d, which is about 35 % longer than in Fig.2c. Then, what does longer Lc mean?

For a fixed-length random walk, longer Lc means longer total codeword sequence. Since the codeword can be divided into two parts, NID part and CID part, longer codeword sequence means longer NID part or/and longer CID part. Assuming a node $v(i)$ is assigned to a wrong community C_t while the groundtruth is that $v(i) \in C_s$ where $s \neq t$, connections between $v(i)$ with nodes in C_s will be denser than with nodes in C_t. Then from the point view of information code, $v(i)$ may result in longer codeword in C_s than in C_t, though $v(i)$'s codeword may be shorter. Just like $v(5)$ in Fig. 2, its connection in C_1 is denser than in C_2. When assigning $v(5)$ to C_2, the average length of NIDs in C_2 grows ($2.0 \rightarrow 2.2$). Besides, incorrect assignment will result in increase of the length of CID part. The random walker would have to back and forth migrate between communities, while this case will lead to longer CID part and make it more difficult for community mining. In Fig. 2d, due to $v(5)$'s assignment, more CIDs appear in the codeword sequence. Finally, comparing with the two community partition patterns in Fig. 2c, d, the latter needs more bits to describe a step in random walk. It is clearly that community partition in Fig. 2c is more reasonable than in Fig. 2d.

Assuming the optimal community pattern is D^*, with the discussion result above, we can infer that $Lc(D^*)$ will be the shortest. The problem of community structure mining will become how to find the optimal community partition pattern. Besides, because for a static network, Lcs for all Huffman coding strategies are constant. Then, the problem will be translated into minimum description length (MDL) problem or a mathematical optimization problem. In next section, we will detail how to combine MapReduce with map equation to solve the community structure mining problem.

3.4 Objective function

According to random walk theory, when the random walker try to move, the next node may be within current community or outside it. With the information provided in previous section, when computing how many bits are needed to describe this step, we know that these two situations are quite different. Behaviors of the walker can be divided into three categories, staying in current community, moving from current community, and entering a new community. For the first one, only NID is needed. The second situation needs "NID + exit codeword" and the third needs "CID + NID". Obviously, the second and the last one are concomitant, therefore they share the same probability. For the convenience of description, the first two situation are combined together to form the second part of Expression (5), while the third forms the first part.

For the convenience of calculation, exit codeword is treated as general codeword as NID and all the NID codewords situations are combined together into the second part in Expression (5). And therefore, the first part means the bits needed to describe the CID part in a random walk step and the second part represents the bits needed to describe the NID part in one step of the random walk. p^i represents the probability of staying in the current community C_i during the next step. Let $p(a)$ represent the probability of visiting $v(a)$, then $p^i = q_{i,out} + \sum_{v(a) \epsilon C_i} p(a)$, where $q_{i,out}$ means the exit codeword part and $1 \leq i \leq k$. $H(P^i)$ expresses the information entropy of the visiting probability of the vertices in C_i. $H(P^i)$ can be expressed as:

$$\begin{aligned} H(P^i) &= -\frac{q_{i,out}}{q_{i,out} + \sum_{v(b) \epsilon C_i} p(b)} \log_2 \frac{q_{i,out}}{q_{i,out} + \sum_{v(b) \epsilon C_i} p(b)} \\ &\quad - \sum_{v(a) \epsilon C_i} \frac{p(a)}{q_{i,out} + \sum_{v(b) \epsilon C_i} p(b)} \log_2 \frac{p(a)}{q_{i,out} + \sum_{v(b) \epsilon C_i} p(b)} \end{aligned} \tag{7}$$

In Expression (7), the first term represents the information entropy of the exit codeword, and the second term represents the entropy of the nodes in C_i, because all the three behaviors of the random walker need NID codeword when describing.

After the discussion, we can give the object with the map equation. It is minimizing the average amount of information to describe each step in the random walk process:

$$f(G) = min\{Lc(D)\} \tag{8}$$

From the discussion of Expression (5) and its terms, we can find that the steady visiting probability plays the key role in the calculation of Lc. The probability indicates the possibility of the the random walker moving from one node to another or leaving from current community. In our previous work, we first get the steady visiting probability distribution by pagerank calculation on MapReduce, which results in lots of time consumption. Then we divide the network into several subnetworks. Next we use the MapReduce to speed up the map equation on the subnetworks to detect the community structure independently. In our previous work, network partitioning produce unnecessary performance loss, however it is inevitable in that situation.

After our analysis and discussion, we find that we can use some simple calculation to replace the pagerank calculation to get the probability distribution. For example, the probability of node $v(a)$ should be proportional to its degree divided by the sum of node degree in the network, that is to say we can use $d(a)/sumDegree$ to approximate $p(a)$, where $sumDegree = \sum_{i=1}^{n} d(i)$. For a big data SMN, finding out all community partition pattern and computing all the Lcs is not realistic and not suitable for computing even with MapReduce. To discover all the communities in the network, we still use the greedy search method to find a suitable node for current node to combine, which is applicable on MapReduce.

3.5 Proposed framework

We analyze the practicability of combining map equation with MapReduce in this section and detail our new framework. It is clear that if we compute all the community partition patterns and then search in the partition pattern space, we will be able to compute the Lcs for each partition pattern. Finally, the partition pattern with the minimal Lc will be the optimal partition pattern and all the communities can be derived then. However, for a big data SMN, the size of the community partition pattern space will be huge. Take a SMN with just 100 nodes for example, supposing that there are two equal communities in the SMN, then there will be C_{100}^2 potential community partition patterns. For a larger network with more community number and less community size constraint, the potential community partition patterns will be much more than that. Therefore, this way to find all the communities is unworkable.

For simplicity, we refer to a node as a community and all the symbols in Table 1 and variables applied to nodes are still valid to communities. In the initialization phase, each node is considered as a community. Then an iterative process will be performed. In each iteration, randomly select several communities, and then carry out a local greedy search inspired from map equation in their neighbors to find a community for each community that minimizes ΔLc, where $\Delta Lc < 0$, then merge the community pairs to generate new communities.

For the expression $q_{out} = \sum_{i=0}^{k} q_{i,out}$ in Lc, we use $outLink(C_i)/\sum_{C_i \in CS} d(C_i)$ to replace $q_{i,out}$ and when C_i contains only one node, $outLink(C_i) = d(C_i)$.

$$q_{out} = \sum_{C_i \in CS} \frac{outLink(C_i)}{\sum\limits_{C_i \in CS} d(C_i)} = \frac{1}{2m} \sum_{C_i \in CS} outLink(C_i) \tag{9}$$

It is easy derive $H(Q)$ based on q_{out}:

$$H(Q) = -\sum_{C_i \in CS} \frac{outLink(C_i)}{\sum\limits_{C_i \in CS} outLink(C_i)} * log \frac{outLink(C_i)}{\sum\limits_{C_i \in CS} outLink(C_i)} \tag{10}$$

Next, we can write p^i and $H(P^i)$ as:

$$p^i = \frac{d(C_i) + outLink(C_i)}{\sum\limits_{C_i \in CS} d(C_i)} = \frac{d(C_i) + outLink(C_i)}{2m} \tag{11}$$

$$H(P^i) = -\frac{outLink(C_i)}{outLink(C_i) + d(C_i)} log \frac{outLink(C_i)}{outLink(C_i) + d(C_i)} - \sum_{v(a) \in C_i} \frac{d(a)}{outLink(C_i) + d(C_i)} log \frac{d(a)}{outLink(C_i) + d(C_i)} \tag{12}$$

We store the network in adjacency list format on HDFS. Each record is in "n(i), nv(i)" format containing a node and all its neighbors. To avoid computing the degrees for each node, we compute the degree of each node and write them into a file aforehand. In the initialization phase, we define and initialize some global variables. $cid[\,]$ represents the community id of each node during the iterations. $d[\,]$ records node degree. n integers from 1 to n are randomly stored in $rand[\,]$, which is used to ensure that the community is randomly selected and to avoid the situations that communities in front become too large and posterior communities keep starving. More details about initialization process are as follows:

1: **Initialization phase**:
2: read $degree.dat$ and initialize $d[\ ]$;
3: initialize $cid[\ ]$, set $cid[i] = i$;
4: initialize $rand[\ ]$;
5: initialize $lock[\ ]$, for all i, lock[i]=false;//means the node is not locked
6: initialize K // maximum of communities processed in each iteration

During the iterations, a community may be composed of several nodes and only K communities will be processed in each iteration, because too many communities assigned to a reduce task may cause overload problem. When two communities combine together, the cid value for each node in the two communities should be updated, and the inter links between the two communities will become intra links in the new community. For the cid update, if we update all the nodes of the two communities, it would result in congestion because many mapper clients are running at the same time and $cid[\ ]$ is a global variable. So, we use a new method to find and update the cid value for each node (When two communities, cid_1 and cid_2, are combined together, the ID, cid, assigned to the new community: $cid = min\{cid_1, cid_2\}$). The process is shown in line 3 and line 4 in the following description:

1: **Map phase 1**:
2: read a record $\langle offset, record\rangle$ and parse $record$ into node IDs and store them in $ids[]$, find all the true cid for elements in $ids[]$:
3: $while(cid[ids[i]]! = [ids[i]])$
4: $ids[i] = cid[ids[i]]$;
5: encode current source ID $ids[0]$ with the alphabet $rand[\ ]$, select the top-K smallest as candidates and emit the node with its links to map phase2:
6: $if(rand[ids[0] <= K)\ hs.add(\langle ids[0], ids\rangle)$;
7: $emit(ids[0], ids)$;

Here, we have to explain the record $\langle offset, record\rangle$ in the line 1 described above. It relates to the design detail of the MapReduce. When the network has been written onto HDFS, each line contains a record. The offset of the start position of the record is the default key, and the record itself is the $value$. When a mapper receives a record, it has to filter the long integer - offset, and get the real key and $value$ from the record.

Next, after we get all the real $< key, value >$ for each record (key is a node and $value$ is the neighbors), we use the alphabet $rand[]$ to encode the keys to ensure each node has the same probability to combine with other nodes (shown in line 5). Then we can randomly select some nodes as candidate to combine with other nodes (shown in line 5, 6). Next, the selected records will be sent to the map phase 2 (shown in line 7).

When updating the link information of the new community, it will need lots of time consumption to write data. In our framework, we do not modify the network data, reduce phase collects the link information and reconstructs the community and its neighbors. To make it workable, we have to override the $getPartition()$ function of the partitioner in MapReduce, and to make it able to read a global variable $Set\langle int, hashSet\langle int\rangle\rangle\ hs$ to ensure a $\langle key, value\rangle$ can be sent to multiple reducers. Here, hs records the keys and the reducer IDs that the keys are assigned to (the first int is a cid, and the $hashSet\langle int\rangle$ records the its neighbors). $key.isInHs()$ returns whether the key is in the hs.

1: **Map phase 2**:
2: receive a record $\langle key, value\rangle$, i.e. $\langle v(i), nv(i)\rangle$;
3: emit the node with its links to reduce phase (partitioner guarantees to send the data to the right reducer):
4: $if(key.isInHs())\ emit(key, value)$;

Since only two communities are involved during a combination, we can calculate the Lc changes caused by the combination of the two communities. When trying to combine C_i and C_k, only some inter links between C_i and C_k become intra links. Then we will be able to compute the changes caused by the two communities. During this process, we assume that the other communities keep unchanged. Now, for a randomly selected community C_i, our object becomes to find a neighbor community satisfying:

$$min\{\Delta Lc\} = min\{Lc(C_{i,k}) - Lc(C_i) - Lc(C_k)\} \qquad (13)$$

where $C_k \in nb(C_i)$ and $\Delta Lc < 0$.

1: **Reduce phase**:
2: receive all the $\langle sid, list\langle value\rangle\rangle$, sort and compress them.
3: $Set\ st = getAssigned(sid)$ //partitioner will write the assignment to a global variable $Set\langle int, HashSet\langle int\rangle\rangle\ hs$
4: sparer received $value$s and construct the community C_{sid} and get all its inter links and intra links.
5: initialize variables, such as $inLink$, $outLink$, d, $inDegree$ and $outDegree$.
6: iteratively select a community C_j from $nv_{C_{sid}}$ and compute the δLc according to Expressions (9)–(13)
7: choose the community C_{min} that minimizes the δLc to combine, where $\delta Lc < 0$
8: if there is no community to make $\delta Lc < 0$, do
9: $lock[sid] = true$;
10: $else$ update $cid[min]$ and $cid[sid]$ and lock the combined one.
11: $cid[min] = cid[sid] = min(min, sid)$;
12: $lock[max(min, sid)] = true$;
13: $endif$

To explain the process clearly, we will present an example. Assuming there is a community cid_i and its neighbors $nv(i)$, during the second map phase, we can get cid and all its neighbors from hs. Then, all the links of the nodes belonging

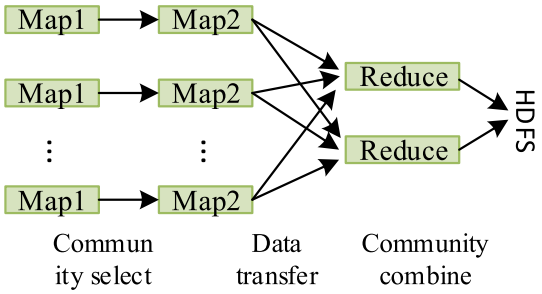

Fig. 3 Community mining process in the MapReduce framework

to $nv(i)$ will be sent to the reducer which is assigned to process the community cid. Then, during the reduce phase, the local connections between the community cid and its neighbors will be reconstructed. Next, the reducer will select one neighbor for cid to combine with the help of Expression (13)

The whole process is implemented with ChainMapper/ChainReducer strategy, and the main framework is shown in Fig. 3. The framework will startover new chain process before all nodes are locked. Finally, the value in $cid[\ ]$ will be the community partition result.

4 Empirical experiments and analysis

Experimental platform is a Hadoop-1.1.1 cluster of Antivision Software Ltd. The cluster consists of 20 PowerEdge R320 servers (Intel Xeon CPU E5-1410 @2.80 GHz, memory 8 GB) with 64-bit NeoKylin Linux OS, and servers are connected by a Cisco 3750G-48TS-S switch.

4.1 Accuracy tests and analysis

The algorithm proposed in this paper is compared with the Fast GN algorithm [21], the Infomap algorithm , Radetal method [26] and our previous work InfoMR. We take the normalized mutual information (NMI) [21] and recall rate as evaluating indicators. Recall rate is defined as the percentage of nodes assigned to the correct communities. NMI is usually used for precision contrast to evaluate the calculation accuracy of the algorithms, the equation is shown in Expression (14). Besides, because NMI is derived from matrix operation, it is not suitable for very large dataset. We use NMI to evaluate performance on small datasets and recall rate to evaluate the large datasets.

$$NMI(X;Y) = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}} \tag{14}$$

where $I(X;Y)$ represents the mutual information between two discrete random variables X and Y.

In order to test the accuracy of the algorithms, we use LFR [17] to generate some artificial networks with groundtruth information of all communities, and the configuration is shown in Table 2. Parameters are described as follows: m represents the number of links; d represents average degree; $max(d)$ represents the maximal degree; u represents the average ratio of inter-community links of a vertex, so, larger u corresponds to vaguer community structure; $min(c)$ represents the number of vertices in the smallest community; $max(c)$ represents the size of the largest community. Because the data sets in the accuracy test are small, both the number of reducers and K is set to 1 during the tests.

Figure 4 shows the results of experiments on the four data sets, NMI value changes with different u value. From Fig. 4a–d we can see, accuracy of the Fast GN algorithm decreases rapidly when network community structure becoming dim (u value increases). This is because there exists a "resolution" restriction in the objective function of the "modularity" based greedy searching algorithm, and the algorithm tends to discover communities with similar size.

The infomap algorithm is one of state-of-the-art community detection method. It maintains a high accuracy before u grows to 0.7, and the performance is rarely affected by the community size. As u grows up, community structure becomes more and more unsharp. When $u > 0.7$, the network generated by LFR would be more like a random network rather than a community-structured network. The reason why NMI still keeps at high level even if $u$is more than 0.7 is that LFR will generate lots of small communities when u is high and all the community detected are small, too. Due to the

Table 2 Description of LFR generated datasets

Data set	m	n	d	$max(d)$	$min(c)$	$max(c)$
5000(S)	41,125	5000	20	40	10	40
5000(B)	45,722	5000	20	80	20	80
50,000(S)	832,980	50,000	40	80	10	80
50,000(B)	882,453	50,000	40	160	10	160
D1	5,000,000	100,950,931	40	160	40	200
D2	8,000,000	208,472,382	50	200	50	250
D3	10,000,000	232,038,266	45	200	50	250

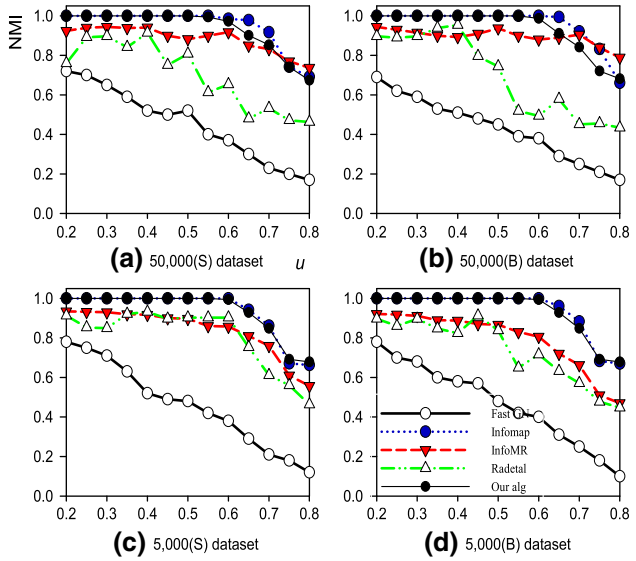

Fig. 4 Accuracy comparison of different algorithms on 4 data sets

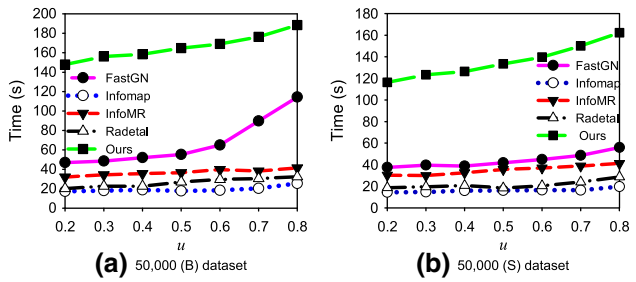

Fig. 5 Comparison of the methods on running time test on 50,000(B) and 50,000(S) data sets

NMI's shortcomings in processing lots of small communities, the NMI will remain at a high level.

Radetal method uses edge clustering to distinguish inter from intra links, its output consists of a dendogram reporting the internal hierarchical structure of the network analyzed. Splits on the dendogram are then considered as significant or not depending on whether the split communities satisfy or not the definition of weak (or strong) community.

InfoMR is our previous work, and it performs better than fast GN and radetal algorithms in all the four datasets. The new framework proposed in this paper has archived a better clustering precision both in the four different networks and approximates infomap. This indicates that our modified two level description model is able to highlight the community structures in the network and effectively help to discover the communities.

Next, we test the time consumption for each method on the two networks with 50,000 nodes. And the results are shown in Fig. 5

From the results shown in Fig. 5, we can see that our methods requires much more time than the others. Because

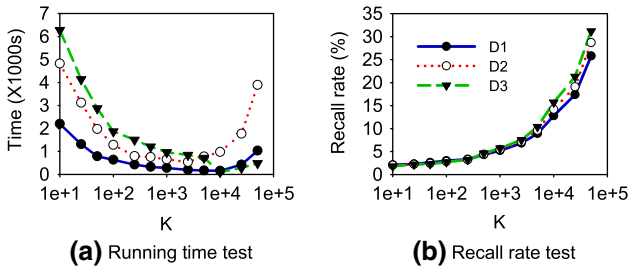

Fig. 6 Running time and recall rate tests of the new framework on 3 large-scale artificial data sets

the networks are so small that they can be stored in just one block on HDFS. So for InfoMR, we do not need to divide the network into several subnetworks. Then the process on InfoMR is like starting the MapReduce framework and then calls infomap to find the communities in the network. Considering the initialization of MapReduce framework needs about more than 10 s, the real time consumption for community detection in InfoMR is very close to infomap. FastGN, earlier than the other methods, is a greedy method and requires more time consumption. Our method needs more iterations and needs more data transmission among the servers, which will need more time consumption. If we use only one server, the time consumption will be much more than that shown in the figure.

Furthermore, we conduct an experiment on large-scale datasets to evaluate the recall rate of the new framework affected by different K values. Results are shown in Fig. 6. During the experiments, the number of reducer is set to 20.

From Fig. 6a, we can find that running time is not steadily dropping as K increases and all curves present carryover effects. When K is small, such as the case $K = 50$, very few communities are combined. For a certain network, number of communities and the community structure in it are changeless. Smaller K corresponds more iterations and will cause more time consumption. In the new framework, reducer is responsible for community combination. To do this, a reducer have to receive a community with all its neighbors from map phase. When K grows to very large value, such as $K = 50,000$, each reducer have to handle about 2500 communities. Taking the amount of neighbors of each community into consideration, the amount of communities assigned to one reducer will be far more than 2500. On one hand, receiving so many community data and reconstruct all the communities will take lots of time. On the other hand, if the data is too large, it has to be written to the local disk of the reducer, which will introduce a great deal of unnecessary time consumption. Finally, lose more than gain. Besides, optimal K varies in different datasets and different clusters with different configurations. In Fig. 6b, recall rate increases as K grows. That is because the new framework adopts an asynchronous update mechanism during the iter-

Table 3 Three real social network used in accuracy test

Data set	n	m
LiveJournal	3,997,962	34,681,189
ASkitter	1,696,415	11,095,298
com-Orkut	3,072,441	117,185,083

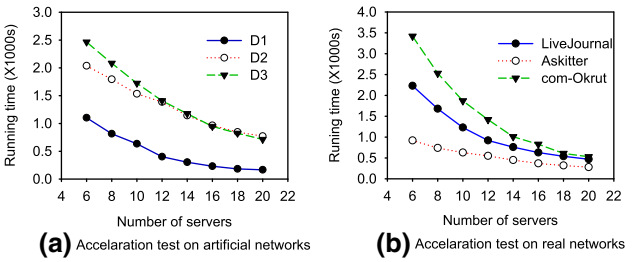

Fig. 7 Time consumption tests on 6 data sets with different number of servers

ative process. Assuming one community C is used by more than one reducer, if status of C is updated in one reducer, the others are still using the old C.

4.2 Acceleration performance test

We use 6 different data sets including 3 real data sets—LiveJounrnal,[1] ASKitter,[2] com-Orkut[3] and three artificial networks—D1, D2 and D3, to test the acceleration performance, and the number of severs in the cluster varies from 6 to 20. LiveJournal is an online dating blog network. ASkitter is a network describing the topology of Internet. Orkut is a free online social network where users form friendship with each other. Table 3 shows the details of the three real SMN.

Figure 7 shows the result of the tests on the 6 datasets. From Fig. 5, we get the information that for the artificial networks of several million nodes, when K varies from 1000 to 10,000, we can get a relative balance between running time and recall rate. Therefore, during the acceleration test, we set $K = 5000$, and the number of reducers is set to equal the number of servers.

From Fig. 7, we can conclude: for a certain network, the running time and the number of reducers is linear approximation, running time decreases as the number of reducer increases. The running time of the new framework will be affected significantly when the number of reducers is small for two reasons: (1) for a fixed K, when reducer number is small, the average number of communities assigned to each reducer will be very large. When increasing the reducer number, the load on each reducer will decrease steeply. (2) Add new severs into the cluster means the computational capabilities of the cluster will be enhanced, then the time consumption will decrease. As number of reducer increases, speedup of gains from these reasons given above will slow down. Besides, MapReduce needs some time to initialize, and the ratio of MapReduce's initialization time to the total running time changes as reducer number changes. In case of smaller number of reducers, for a specific data set, the new framework will need longer running time, and MapReduce's initialization time will account for lower proportion of the total running time. When reducer number increases to a "critical point", the proportion of the initialization time will not be negligible.

From Fig. 7, we can get that the running time of the three LFR generated networks is less than that of the three real networks, which signifies that the new framework runs faster on artificial networks than on real networks. From the statistical information collected from the data sets, we find that community structure in real networks is much more complex than in LFR generated networks. For example, the maximum of degree in LiveJounal is 785,930, which means that more than 20 % of the vertices are connecting with a certain node. While in D1, although the total number of node reaches up to 5 million, the maximum of degree is only 160, the degree distribution is limited in a relatively narrow range as a result of LFR.

All in all, the results of tests on both artificial data sets and real networks show that the new framework has good scalability and is well workable on community structure mining in big data social media networks.

5 Related work

During the past several years, researchers in several research domains, such as computer science, complex system, information science have proposed lots of work on community detection [24], including standalone methods, parallel methods and distributed methods.

Most of community mining methods [3,8] are standalone, which just use one core/processor of the server to detect the community structure in networks. Standalone method has been studied for a long period. The earliest methods are based on clustering like k-means [20] and others applications [16]. Later, divisive algorithms mainly based on hierarchical clustering, modularity-based algorithms [5,9,21] and other similar algorithms have been proposed. And then, spectral algorithms [25], information coding based methods [28–30] come out. At the beginning, the methods are used to process real-world small network, such as Zachary's karate club network, dolphins networks,et al. And the methods usually have high computational complexity (GN's complexity is $O(m^3)$

[1] http://snap.stanford.edu/data/com-LiveJournal.html.

[2] http://snap.stanford.edu/data/as-skitter.html.

[3] http://snap.stanford.edu/data/com-Orkut.html.

[9]) and the accuracy is low (GN, FastGN [21], et al.). Then, researchers try to improve the complexity and the accuracy, and try to mining community structure in large networks with millions of nodes, and have achieved excellent results (such as infomap [29]). Comparison of the methods described above has been discussed in [18,19,22].

Then, some parallel methods [27,32] have been proposed to deal with large scale networks. Most of them are running on a single HPC server even supercomputer, and all the processors share the same centralized huge capacity memory. The network and its statuses are maintained in the memory during the process. We can think that these methods just speed up the community mining process, and are highly dependent on the hardware.

MapReduce has become widely used in big data processing scenarios. Recently, some researchers try to use MapReduce to speedup existed methods (such as Shingling method, GN method) to reveal the community structure in very large scale networks [4,15]. However, MapReduce is not good at graph processing, due to the complex connections within the nodes. The network is stored in distributed memory, so how to exchange the nodes and links information among the workers, and how to get the global information about the graph is a challenge.

6 Conclusion and future work

SMNs are expanding rapidly and playing important roles in people's daily life. Research on community structure has major theory significance and strong application value. In the big data age, how to quickly identify the hidden communities has become a challenge. Based on our previous work and the map equation, we design and implement a distributed community structure mining framework using MapReduce. Empirical experiments show that the new framework is able to tackle the challenge, and its performance approximates the state-of-the-art methods.

During the experiments, we found that the greedy search method has some room for improvement. We will solve this problem and improve the performance in the future work.

Acknowledgments The authors would like to express our sincere gratitude to Professor Philip S. Yu from University of Illinois at Chicago, Mr. Zhang Yuchao from Beijing Institute of System Engineering for providing great assistance through the entire research process. Besides, this work was supported in part by the National High-Tech Research and Development Program of China (2012AA012600), National Natural Science Foundation of China (61202362, 61472433).

References

1. Andreev, K., Racke, H.: Balanced graph partitioning. Theory Comput. Syst. **39**(6), 929–939 (2006)
2. Borthakur, D.: HDFS architecture guide, HADOOP APACHE PROJECT. http://hadoop.apache.org/common/docs/current/hdfs_design (2008)
3. Cambria, E., Rajagopal, D., Olsher, D., Das, D.: Big social data analysis. Big Data Comput. 401–414 (2013)
4. Chen, Y., Huang, C., Zhai, K.: Scalable community detection algorithm with MapReduce. Commun. ACM **53**, 359–366 (2009)
5. Clauset, A., Newman, M.E., Moore, C.: Finding community structure in very large networks. Phys. Rev. E **70**(6), 066111 (2004)
6. Cooper, S.: The largest social networks in the world include some big surprises, Business Insider, New York, USA. http://www.businessinsider.com/the-largest-social-networks-in-the-world-2013-12 Accessed Jan 2014
7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
8. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3), 75–174 (2010)
9. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. **99**(12), 7821–7826 (2002)
10. Gleiser, P.M., Danon, L.: Community structure in jazz. Adv. Complex Syst. **6**(04), 565–573 (2003)
11. Huffman, D.A.: A method for the construction of minimum redundancy codes. Proc. IRE **40**(9), 1098–1101 (1952)
12. Ihara, S.: Information Theory for Continuous Systems. World Scientific, Singapore (1993)
13. Jin, S., Li, A., Yang, S., Lin, W., Deng, B., Li, S.: A MapReduce and information compression based social community structure mining method, IEEE 16th International Conference on Computational Science and Engineering (CSE), 2013, pp. 971–980. (2013)
14. Jin, S., Yu, P., Li, S., Yang, S.: A parallel community structure mining method in big social networks, mathematical problems in engineering, (in Press) http://downloads.hindawi.com/journals/mpe/aip/934301 (2014)
15. Kalyanaraman, R.A.: An efficient MapReduce algorithm for parallelizing large-scale graph clustering,In: ParGraph—Workshop on Parallel Algorithms and Software for Analysis of Massive Graphs, Held in conjunction with HiPC'11. Bengaluru, India (2011)
16. Kernighan, B.W., Lin, S.: An efficient Heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(2), 291–308 (1970)
17. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E **78**(4), 046110 (2008)
18. Lancichinetti, A., Fortunato, S.: Community detection algorithms: a comparative analysis. Phys. Rev. E **80**(5), 056117 (2009)
19. Leskovec, J., Lang, K. J., & Mahoney, M.: Empirical comparison of algorithms for network community detection. In: Proceedings of the 19th international conference on World wide web, 631–640 (2010)
20. MacQueen, J.: Some methods for classification and analysis of multivariate observations, In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. 1(14), 281–297 (1967)
21. Newman, M.E.: Fast algorithm for detecting community structure in networks. Phys. Rev. E **69**(6), 066133 (2004)
22. Orman, G.K., Labatut, V., Cherifi, H.: Comparative evaluation of community detection algorithms: a topological approach. J. Stat. Mech. Theory Exp. **2012**(08), P08001 (2012)
23. Pasco, R.C.: Source coding algorithms for fast data compression. Stanford University, Ph.D. dissertation (1976)
24. Plantié, M., Michel, C.: Survey on Social Community Detection, Social Media Retrieval. Springer, London (2013)
25. Pothen, A., Simon, H.D., Liou, K.P.: Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Matrix Anal. Appl. **11**(3), 430 (1990)

26. Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., Parisi, D.: Defining and identifying communities in networks. Proc. Natl. Acad. Sci. USA **101**(9), 2658–2663 (2004)
27. Riedy, E.J., Meyerhenke, H., Ediger, D., Bader, D.A.: Parallel community detection for massive graphs. In: Parallel Processing and Applied Mathematics, pp. 286–296. Springer, Berlin Heidelberg (2012)
28. Rosvall, M., Esquivel, A., Lancichinetti, A., West, J., Lambiotte, R.: Memory in network flows and its effects on spreading dynamics and community detection. Nat. Commun. **5**, 2014, doi:10.1038/ncomms5630
29. Rosvall, M., Bergstrom, C.T.: An information-theoretic framework for resolving community structure in complex networks. Proc. Natl. Acad. Sci. **104**(18), 7327–7331 (2007)
30. Rosvall, M., Bergstrom, C.T.: The map equation. Eur. Phys. J. Spec. Top. **178**(1), 13–23 (2009)
31. Shannon, C.E.: A mathematical theory of communication. ACM SIGMOBILE Mobile Comput. Commun. Rev. **5**(1), 3–55 (2001)
32. Staudt, C.L., Meyerhenke, H.: Engineering parallel algorithms for community detection in massive networks. arXiv:1304.4453 (2014)
33. Yang, B., Liu, D., Liu, J.: Discovering communities from social networks: methodologies and applications. In: Furht, B. (ed.) Handbook of Social Network Technologies and Applications, pp. 331–346. Springer, New York, USA (2010)

Songchang Jin received the B.S. degree in automation at Tsinghua University in 2008, and received the M.S. degree in computer science and technology at National University of Defense Technology (NUDT) in 2010. Now he is a Ph.D. candidate in Computer science and technology at NUDT. He visited the University of Illinois at Chicago in 2014. His research interests include big data analysis, social network analysis. He is an IEEE member.

Wangqun Lin received his M.S. degree and Ph.D. degree in computer science and technology at National University of Defense Technology in 2007 and 2012, respectively. He visited the University of Illinois at Chicago. He is an engineer in the Beijing Institute of System Engineering. His research interests include graph mining, social network analysis.

Hong Yin received the M.S. in School of Computer in National University of Defense Technology in 2011 and is currently a Ph.D. candidate at this University, Changsha, China. He was a teacher at Military College of Economics in 2006–2008. His research interests include satellite fault diagnosis, cloud computing, massive data mining.

Shuqiang Yang received the Ph.D. degree in computer science and technology from National University of Defense Technology in 1996. He has been a professor in the National University of Defense Technology since 1998. His research interests include cloud computing, database.

Aiping Li received the B.S. and Ph.D. degrees in computer science and technology at National University of Defense Technology (NUDT), China in 2000 and 2004, respectively. He has been a professor in the College of Computer, National University of Defense Technology, since 2006. He visited the University of New South Wales, Australia. His research interests include uncertain databases, data mining, spatial databases and time series databases. He is a member of the IEEE.

Bo Deng received his Ph.D. degree in the computer science and technology at National University of Defense Technology in 2000. He is a senior engineer in the Beijing Institute of System Engineering. His research interests include complex networks and graph analysis.