Cluster Comput
DOI 10.1007/s10586-015-0504-2

SIMPLE: a simplifying-ensembling framework for parallel community detection from large networks

Zhiang Wu[1] · **Guangliang Gao**[2] · **Zhan Bu**[1] · **Jie Cao**[1]

Received: 12 March 2015 / Revised: 24 October 2015 / Accepted: 26 October 2015

Abstract Community detection is a classic and very difficult task in complex network analysis. As the increasingly explosion of social media, scaling community detection methods to large networks has attracted considerable recent interests. In this paper, we propose a novel SIMPLifying and Ensembling (SIMPLE) framework for parallel community detection. It employs the random link sampling to simplify the network and obtain basic partitionings on every sampled graphs. Then, the K-means-based Consensus Clustering is used to ensemble a number of basic partitionings to get high-quality community structures. All of phases in SIMPLE, including random sampling, sampled graph partitioning, and consensus clustering, are encapsulated into MapReduce for parallel execution. Experiments on six real-world social networks analyze key parameters and factors inside SIMPLE, and demonstrate both effectiveness and efficiency of the SIMPLE.

Keywords Complex network · Community detection · Parallel computing · MapReduce · K-means

1 Introduction

Community detection has become one of the core problems in the realm of data mining and social network analysis [7]. It is concerned with the identification of communities or clusters within networks that are more densely linked, as compared to the rest of the network. There is a lot of evidence that community structures are crucial to the understanding of functional properties of networks and are also very helpful for developing intelligent services [20,21].

In the past decades, a great deal of solutions have been designed for both effective and efficient community discovery [2,3,18,21,25,32]. However, much of them are usually computationally expensive and are not scalable to networks in colossal sizes. For instance, two famous algorithms, i.e., FastNewman [18] and LPA [25], should take $O(m \log n^2)$ and $O(m+n)$ time respectively, which actually impedes them from discovering communities from large-scale networks. Furthermore, the emergence very-large real-world networks exerts big pressures to a single machine on both data storage and data computation. Hence, it is desiderative to put forth a feasible yet compact parallel framework for scaling the community discovery.

Some tentative exploration has been made towards parallel community detection. Speciafically, the global optimization on some criterion and the mining on some local structures are implemented in parallel or even distributed enviroment [8,10,24,37]. Thus, several famous community detection methods, such as local community detection [6], clique percolation method (CPM) [21] and modularity optimization [3,18], do have the corresponding parallel solutions [8,10,12]. To further address this important problem, this paper presents the so-called SIMPLifying-Ensembling (SIMPLE) framework for scaling the community discovery. Note that SIMPLE is not the parallel extension of any existing methods, but truly a novel design. In general, SIMPLE consists of two phases. Firstly, it employs simple random sampling to simplify the large network and obtain basic partitions on every sampled network, i.e., a much smaller graph. Secondly, based on multiple sampled networks and their derived basic partitionings, SIMPLE utilizes the K-means-based Consensus Clustering

✉ Zhan Bu
buzhan@nuaa.edu.cn

[1] Jiangsu Provincial Key Laboratory of E-Business, Nanjing University of Finance and Economics, Nanjing, China

[2] College of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China

Published online: 09 November 2015

to ensemble all of basic partitionings, in order to get high-quality global community structures.

MapReduce has long been a widely-used distributed data processing framework in a cluster, due to many of its consipicuous merits such as flexibility, scalability, efficiency, and fault tolerance [16]. Various platforms implementing MapReduce have emerged, among which the open-source Hadoop [30] gains the particular interests in practice. To parallelize SIMPLE, we adopt Hadoop as the basic middleware of the cluster, and Hadoop Distributed File System (HDFS) as the basic tool for data sharing. Thus, we encapsulate all of components of SIMPLE into MapReduce including random sampling, sampled graph partitioning, and consensus clustering. Experiments on six real-world large-scale networks validate the SIMPLE can discover high-quality communities in an efficient and scalable way.

The remainder of this paper is organized as follows. Section 3 gives some preliminaries and introduces the Simplifying and ensembling system. In Sect. 4, we discuss technical details in our SIMPLE framework. Experimental results will be given in Sect. 6. We present the related work in Sect. 2, and finally conclude this paper in Sect. 7.

2 Related work

In the literature, there have been a large body of studies done on community detection [7,22]. Most of the existing methods fall into two categories, in terms of whether or not explicit optimization objectives are being given [32]. The former views the community discovery problem as an optimization problem on a criterion, e.g., modularity optimization [3,18], WCC (weighted community clustering) optimization [23], and so on. The latter detects communities via mining and merging local structures, e.g., k-clique [21].

As the explosive increase of the scales of real-life networks, the scalability of community detection algorithms draws much attention. Using parallel and/or distributed computing paradigms to scale existing algorithms is the most intuitive way. Along this line, the modularity optimization [12], WCC optimization [24] and k-clique mining [8] are implemented in parallel or distributed computing framework. Moreover, the MapReduce programming framework has been employed for solving critical tasks insider community discovery, such as edge betweenness computation [17] and node-pair similarity computation [27].

To meet the computing challenge rising from large-scale networks, the exploration of elegant parallel and distributed strategies for scalable community detection keeps on booming. Our work tries to make a useful attempt towards this direction, and differs from existing solutions by designing a novel simplifying and ensembling scheme. Hence, the proposed SIMPLE framework, designed with distributed computing in mind, can take advantage of the MapReduce model.

3 Problem definition and system overview

Although being extensively studied [7,22,28], the preliminary knowledge on community detection is still provided in this section to make our paper be self-contained. We then overview the SIMPLE from the system perspective.

3.1 Problem definition

Given an *undirected* network $\mathcal{G} = \{V, E\}$, V is a set of n nodes and E is a set of m edges. The goal of community detection is to seek a good K-way crisp partitioning $\pi = \{C_1, \ldots, C_K\}$, where C_k is the kth community, and $C_1 \bigcup \cdots \bigcup C_K = V$, $C_k \bigcap C_{k'} = \emptyset \ \forall \ k \neq k'$. A network is commonly represented by a $n \times n$ adjacency matrix $\mathbf{A} = [A_{uv}]$, where $A_{uv} = 1$ $(u \neq v)$ if there is an edge between node u and v and 0 otherwise.

To select an appropriate storage model to fit network data to HDFS, we do not use the commonly-used adjacency matrix, since it leads to data redundancy when the network data is sparse. We also give up the key-value storage model for its slow response to neighborhood queries. Instead, we finally employ the *transaction model* to represent a network. Formally, $\mathcal{G}$ can be formulated equivalently as a transaction data set $\mathcal{T} = \{T_u | \forall \ u \in V\}$, with $T_u = \{v | A_{uv} = 1\}$ being a transaction containing all the neighbors of u. Our method actually works on $\mathcal{T}$ rather than $\mathcal{G}$ for parallel community detection.

3.2 System overview

Figure 1 describes the system overview of the SIMPLE framework. SIMPLE holds in store the *divide-and-conquer* idea. That is, it first divides the whole large network into multiple pieces and uses a given community detection algorithm to obtain basic clusters in every piece. The global community structures can then be integrated from multiple sources. In what follows, we shall introduce key steps within SIMPLE.

Sharding This step *horizontally* splits $\mathcal{T}$ into a set of small parts named *shards*, each of which must keep the completeness of transactions. The `split` command in `Linux` is used for sharding, which is rather efficient in practice. Each shard is roughly 64M according to the configurations of the `HDFS`. All shards are then stored in `HDFS` distributed across multiple machines, and the storage location is transparent to users. Every computational node can share the shards by accessing `HDFS`. The sharding can significantly decrease the I/O cost of the successive `map` procedure. We will give the verification results in the experimental section.

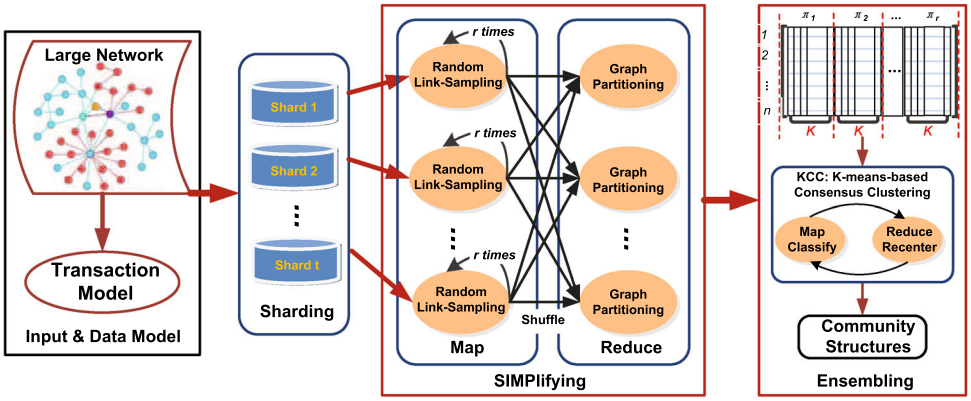


Fig. 1 The system overview of SIMPLE

Map Since a number of community detection algorithms are heavily dependent on the number of links, it is reasonble to reduce computational complexity by considering only a subset of links on an edge-intensive network. A random link sampling algorithm is thus encapsulated in the `map` function. Each `map` node works on a shard, i.e., a part of the input network, and repeats random sampling for r times. More details will be given in Sect. 4.

Reduce The output of `map` is arranged as "key/value" pairs, where $key[i]$ corresponds to the ith ($1 \leq i \leq r$) link-sampling and $value[i]$ records the part of sampled network. With the help of `shuffle` function, sampled fragments with the same key are mapped to a node, and thus a complete sampled graph is formed. Any existing graph partitioning methods (e.g., METIS and FastNewman) could be invoked to obtain community structures in the sampled network. Section 4.2 will introduce the implementation of the `reduce` function in SIMPLE.

Ensembling Based on repetitious random sampling, the `reduce` outputs r basic graph partitionings of $\mathcal{G}$, i.e., $\Pi = \{\pi_1, \pi_2, \ldots, \pi_r\}$. The task in this step aims to find a single partitioning, i.e., community structures, of $\mathcal{G}$ from r basic graph partitionings in Π. We formulate this task as the consensus clustering (a.k.a. cluster ensemble) problem, and thus utilize the so-called K-means-based Consensus Clustering (KCC) [31] to solve it. Similar to K-means, KCC is easily be parallelized in MapReduce. Two phases of K-means "classify" and "recenter" are encapsulated into `map` and `reduce` respectively, and an iterative MapReduce process is activated until centroids do not change. Section 5.1 will describe details of KCC.

4 SIMPLE: the simplifying phase

In this section, we describe technical details of the "simplifying" phase in our framework. It aims to decompose the large network into a set of small networks and partition each small network into a certain number of communities. This phase is implemented into MapReduce jobs to facilitate the distributed processing.

4.1 Map: random sampling

Since the very-large network is hard to be loaded into memory and be further processed, we hope to find a sampling technique that is qualified to this simplifying task. Though graph sampling techniques are extensively studied [9,13], we argue that the selected sampling technique should satisfy at least two conditions: (i) the sampling itself must be as simple as possible, otherwise it will become the bottleneck of the whole system; (ii) the sampling should only rely on the *partial* network rather than the whole network. Both conditions are in fact outlined based on the efficiency issue of large network processing.

Existing graph sampling algorithms can conceptually be divided into two groups [13]: randomly selecting nodes/edges and sampling by exploration. Since the community detection needs to assign labels for every node and many detection methods [11,18] are heavily dependent on the number of edges, we choose random *edge* sampling, which obviously satisfies both conditions mentioned above. As for other sampling techniques by exploration, such as random walk [13], forest fire [14] and metropolis sampling [9], they not only do drop a few nodes but also take too much time for computing various decision-making indices. Worse yet is, sampling by exploration is similar to random walk that needs the global view of the large network and thus it is difficult to be parallelized.

Once random edge sampling is selected, the arising problem is can a subgraph better preserve the community structure of the original graph. In the context of community detection, as a community can be simply viewed as a group of nodes with higher similariy, the sampled graph is expected to reserve node pairs with high-similarity as much as possible. Motivated by this, we first present a *null model* as the

reference model for the random edge sampling. The null model provides the expectation for the similarity measure of a given node in a pure random case. It is based on the following null hypothesis: the similarities between the node v and its neighbors are produced by a random assignment from a Bernoulli distribution [26]. Given a similarity threshold $\epsilon \in [0, 1]$, the high-similarity neighbors selection function f_v is therefore defined as:

$$f_v = \frac{1}{d_v} \sum_{u \in T_v} I(s_{v,u} > \epsilon), \tag{1}$$

where d_v is the degree of node v, $s_{v,u}$ ($s_{v,u} \in [0, 1]$) is the similarly between v and u, and $I(\cdot)$ is an indicator function. The value of f_v is the proportion of high-similarity neighbors of v. When the random link sampling is applied, f_v may be deteriorated. However, in what follows, we prove that this deterioration is not significant even when the sample size n_v is small.

Theorem 1 *Let $s = n_v$ be the sample size for a node $v \in V$ and $\{(v, u_1), \ldots, (v, u_s)\}$ be the set of sampled links among the incident links of v. Let $X_v^j = I(s_{v,u_j} > \epsilon)$ for each $j \in [1, s]$. Then, the estimation of f_v using the sampled incident links of v can be represented as $\bar{X}_v = 1/s \sum_{j=1}^{s} X_v^j$. The errors of this estimation are bounded by $Pr(|\bar{X}_v - f_v| \geq \delta f_v) \leq 1/d_v$ if $s \geq 3(\ln d_v + \ln 2)/f_v\delta^2$. Here, $\delta \in [0, 1]$ is the relative error, which is defined as the ratio between the absolute error and the value of f_v.*

Proof (Chernoff bound [4]) Let $X_v^1, \ldots, X_v^s$ denote a set of independent variables following the Bernoulli distribution with the success probability f_v. For any $\delta \in [0, 1]$, we can obtain that $Pr(|\bar{X}_v - f_v| \geq \delta f_v) \leq 2e^{-sf_v\delta^2/3} \leq 1/d_v$ if $s \geq 3(\ln d_v + \ln 2)/f_v\delta^2$. □

Theorem 1 implies that the estimation $\bar{X}_v$ is very likely to be close to the true value f_v, when we set the sampling size to be larger than $3(\ln d_v + \ln 2)/f_v\delta^2$. The probability that $\bar{X}_v$ is wrongly estimated can be controlled smaller than $1/d_v$, which is typically very small. Since the Chernoff bound [4] is a bound for the worst case, in practice it is more reasonable to determine the sample size to be even smaller than $3(lnd_v + ln2)/f_v\delta^2$. So, the value of n_v can be determined by

$$n_v = min\{d_v, \alpha(\ln d_v + \ln 2)\}, \tag{2}$$

where α is a nonnegative constant that is used to control the sample size. The parameter α can be seen as the sampling level. We will discuss how to set the value of α in Sect. 6.2. □

In SIMPLE, each shard stores part of nodes but with their complete records in $\mathcal{T}$, i.e., including all of neighbor nodes. The `map` function invoked on a shard randomly samples a certain number of neighbors of every node. With Eq. 2, all of incident links of a low-degree node (e.g., $d_v \leq \alpha(\ln d_v + \ln 2)$) are sampled, and the larger-degree of a node is, the lower proportion of links will be extracted. Note that the sampling is without replacement. Therefore, the more intensive links of a network is, the more greatly the network will be simplified.

4.2 Reduce: sampled graph partitioning

After every `map` node repeats random link sampling on one shard for r times, a set of "key/value" pairs are produced, where $key[i]$, $1 \leq i \leq r$ corresponds to the ith link-sampling and $value[i]$ records the part of sampled network. For example, assume a shard stores $l + 1$ nodes taking v as its first node, i.e., $\{T_v, T_{v+1}, \ldots, T_{v+l}\}$, and let $T_v^{(i)} \subseteq T_v$ denotes the result of ith sampling on v. Thus, the ith "key/value" pair can be represented as

$$\langle key[i] | value[i] \rangle \doteq \langle i | \{T_v^{(i)}, T_{v+1}^{(i)}, \ldots, T_{v+l}^{(i)}\} \rangle. \tag{3}$$

All of sampled fragments with the same key are mapped to the same node, and thus a complete sampled graph can be assembled. This process is automatically executed by the `shuffle` function. Normally, any community detection method could be used for sampled graph partitioning. However, in SIMPLE, we encapsulate METIS [11] in the `reduce` function.

The reason why we select METIS rather than other tools in SIMPLE can be summarized as the following aspects. First, it is commonly that $K \ll n$, so the top-down partitioning used by METIS is usually much faster than the bottom-up agglomeration methods (e.g., FastNewman). Second, since the sampled network might not be a connected graph, the bottom-up agglomeration method such as FastNewman cannot combine different connected components to form a big community. That is, if the sampled graph has $K' > K$ connected sub-graphs, FastNewman finally obtains at most K' communities while cannot obtain K communities. But METIS can always obtain K communities even on a disconnected network. Third, METIS uses some random strategies during coarsening and computing the initial partitioning, which tends to generate partitionings even on similar sampled networks with high-diversities.

5 SIMPLE: the ensembling phase

Here, we discuss the ensembling phase of SIMPLE. We first introduce the basic procedure of a consensus clustering method called KCC, and then describe the implementation of distributed KCC (d-KCC) on top of MapReduce.

5.1 KCC: the basic procedure

Every sampled network has been partitioned into K communities in `reduce`, and their labels can be integrated into a $n \times (rK)$ matrix $\Pi = \{\pi_1, \pi_2, \ldots, \pi_r\}$. Each vector $\pi_i \in \Pi$ is obtained based on the topological relation among nodes. So, the ensemble step only needs to find a high-quality partitioning based on multiple basic partitionings, without considering the topological relations, which typically is a consensus clustering problem.

As is well known, K-means is a prototype-based, simple partitional clustering method. The clustering process of K-means is a two-phase iterative heuristic, with data assignment (i.e., classify) and centroid update (i.e., recenter) staggering successively. The K-means-based Consensus Clustering (KCC) [31] successfully transforms the consensus clustering problem into a K-means problem, and could inherit all merits of K-means: accuracy, efficiency and flexibility. From the algorithmic perspective, KCC still employes the two-phase iterative heuristic, which is similar to K-means. However, the distance between the instance and the centroid should be carefully designed to satisfy the continuously differentiable function for the K-means distance. Here, we select the cosine distance for KCC, due to sparsity of Π. Formally, let m_k denote the centroid of the kth cluster in Π, which is a rK dimensional vector as follow:

$$m_k = \langle m_{k,1}, \ldots, m_{k,i}, \ldots, m_{k,r} \rangle, \tag{4}$$

where

$$m_{k,i} = \langle m_{k,i1}, \ldots, m_{k,ij}, \ldots, m_{k,iK} \rangle. \tag{5}$$

The instance in Π can also be represented as a rK dimensional vector denoted as x_l. Thus, the cosine distance between x_l and m_k is

$$Dist(x_l, m_k) = \sum_{i=1}^{r} w_i \big(1 - \cos(x_{l,i}, m_{k,i})\big), \tag{6}$$

where w_i is the weight of ith basic partitioning, and it can be set to $1/r$ without the prior knowledge or instance-weighting technique. As a result, the centroid m_k can be updated by:

$$\mu(m_{k,i}) = ||m_{k,i}||_2 - ||P^{(i)}||_2, \tag{7}$$

where $P^{(i)} = \langle p^{(i)}_{+1}, \ldots, p^{(i)}_{+K} \rangle$ is a constant vector w.r.t. Π. Particularly, $p^{(i)}_{+k} = n^{(i)}_{+k}/n$, where $n^{(i)}_{+k}$ denotes the number of instances labeled k in ith basic partitioning. Meanwhile, given a d-dimensional real vector y, $||y||_p$ denotes the L_p norm of y, i.e., $||y||_p = \sqrt[p]{\sum_{i=1}^{d} y_i^p}$.

With Eqs. 6 and 7, KCC could be naturally deduced for ensembling community structures. KCC starts by selecting K nodes as initial centroids. Based on the cosine distance, KCC forms K clusters (i.e., communities) by assigning each node to its closest centroid, and then recomputes the centroid of each cluster. This iterative process proceeds until centroids or community labels do not change.

5.2 d-KCC: the MapReduce implementation

If we think of the original network as a $n \times n$ matrix, the dimension of matrix Π has been reduced remarkably, since $r \cdot K \ll n$. Even so, when the network becomes very large, applying KCC on a single machine is probably to be out of memery or to cost much time. We therefore discuss the implementation of KCC in distributed environment. For the sake of simplicity, we still use MapReduce as the basic framework for implementing distributed KCC (d-KCC). As the iterative version of MapReduce is not a standard or not very efficient formulation, we reserve for future research on reforming d-KCC to fit the *in-memory* cluster computing framework [35].

In the literature [38], parallel K-means based on MapReduce is well designed. The `map` function performs the "classify" procedure of assigning every instance to the closest center, while the `reduce` function performs the "recenter" procedure of updating the new centroids. As mentioned in Sect. 5.1, the procedure of KCC is the same as K-means, and KCC differentiates from K-means only on the distance definition. Therefore, d-KCC could be easily implemented on MapReduce along the way that K-means on MapReduce. In practice, we use the open-source codes provided by Apache Mahout [1] and revise the distance computation according to Eq. 6.

6 Experimental results

In this section, we demonstrate both the effectiveness and efficiency of SIMPLE on detecting communities from six real-life social networks.

6.1 Experimental setup

Data sets Six complex networks are selected as experimental data sets. `Oklahoma` [29], `UNC` [29], `Twitter` [15], `Gowalla` [5] and `LiveJournal` [33] are five friendship networks derived from different social networking sites. `Skitter` [14] is an Internet topology graph collected from traceroutes run daily. We summarize some key features of these networks in Table 1, where $\overline{d_v} = 2|E|/|V|$ indicates the average degree, C is the average cluster coefficient, and $p(d_v)$ indicates the distribution of the degree. It was found that all the experimental networks display power law shaped

Table 1 Experimental data sets

Network	$\|V\|$	$\|E\|$	$\overline{d_v}$	C	$p(d_v)$
Oklahoma	17,420	892,524	102.47	0.23	$1.13d_v^{-1.40}$
UNC	18,158	766,766	84.45	0.20	$1.33d_v^{-1.47}$
Twitter	81,306	1,342,296	33.00	0.57	$2.61d_v^{-1.83}$
Gowalla	196,591	950,324	9.67	0.20	$0.38d_v^{-1.74}$
Skitter	1,696,415	11,095,298	13.08	0.26	$0.04d_v^{-1.41}$
LiveJournal	3,997,963	34,681,189	17.35	0.28	$12.7d_v^{-2.40}$

degree distribution($p(d_v) \sim A \cdot d_v^{-\eta}$), with exponents varying in the range $1 < \eta < 3$. Notice that a small value of η indicates that the given distribution is more heavy-tailed.

Competitive tools Five relevant community detection tools are selected as baselines for comparison: METIS [11], FastNewman (FN) [18], Parallel K-means (PaKmeans) in Mahout [1], BigClam [34] and SCD [24]. METIS, FN and Kmeans are three classical algorithms for community detection. BigClam and SCD are two effective yet scalable methods in the state of the art.

Quality measure The modularity (i.e., Q-function) proposed by Newman [19] is chosen to evaluate the quality of partitions, which is defined as

$$Q = \sum_{k=1}^{K} \frac{|E_k|}{|E|} - \left(\frac{d_k^C}{2|E|}\right)^2, \tag{8}$$

where $|E_k|$ is the number of edges within community C_k and $d_k = \sum_{v \in C_k} d_v$ is the sum of node degrees of C_k. It is expected that better communities of a given network will be with bigger Q values.

Hadoop cluster We construct a Hadoop cluster with eight nodes connected by a Gigabit Ethernet as the basic platform of SIMPLE. Each node comes with four quad-core E5-2650v2 processors (2.6 GHZ), 128 GB of RAM, 240 GB of SSD disk and 600 GB of SAS disk. The Hadoop version is 1.0.4. One node is used to run NameNode, JobTracker, DataNode and TaskTracker simultaneously. So, this cluster totally contains eight nodes for `map` or `reduce` tasks.

6.2 Parameters analysis

There are two important parameters in SIMPLE: the sampling level α in Eq. 2 to control the sample size n_v, and the number of random samplings r, i.e., the number of Basic Partitionings (BPs).

The parameter α One direct way to measure the efficiency of sampled graph partitioning is to employ a parameter called sampling rate (denoted as γ), which is defined by the ratio between the number of the sampled links and the total number of links in the original network (see Eq. 9). Obviously, γ plays a crucial role in SIMPLE for balancing the effectiveness and the efficiency. More precisely, given $p(d_v)$ and α, the sampling rate γ can be estimated by

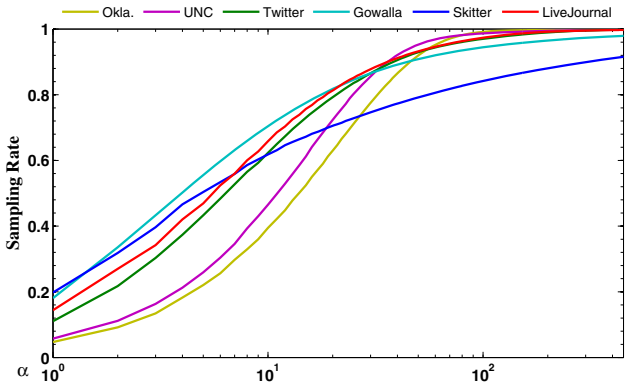


Fig. 2 Sampling rates with different values of α

$$\gamma = \frac{|E'|}{|E|} \approx \frac{\sum_{d_v} np(d_v)n_v}{\sum_{d_v} np(d_v)d_v} = \frac{\sum_{d_v} p(d_v)n_v}{\sum_{d_v} p(d_v)d_v}. \tag{9}$$

where $|E'|$ is the number of edges in the sampled graph. Figure 2 plots γ as a function of α on all networks. As can be seen, γ grows with the increase of α, which indicates that the value of α indeed has a great and positive impact to the sampling rate. Specifically, γ is lower than 60 % when $\alpha < 10$, to balance the effectiveness and efficiency, we set $\alpha \in [1, 10]$ and thus less than 60 % links will be maintained in the sampled graphs, which leads to the high efficient graph partitionings in the successive step. From We can also observe that γ increases slower on the more heavy-tailed network. For instance, the curve of `Skitter` rises faster than the `LiveJournal` curve up to $\alpha = 7$. This implies that the proposed sampling strategy might be more effective for the networks with significantly heavy-tailed.

The parameter r To illustrate the impact of sampling times r in `map`, we set α to be a random value in [1, 10], and linearly increase r from 20 to 120. For each sampled network, METIS is employed to partitioned $K = 10$ communities in `reduce`. Finally, those r basic graph partitioning are ensembled by KCC to obtain $K = 10$ crisp partition. Figure 3 shows the impact of r on Q values. Generally, the quality of

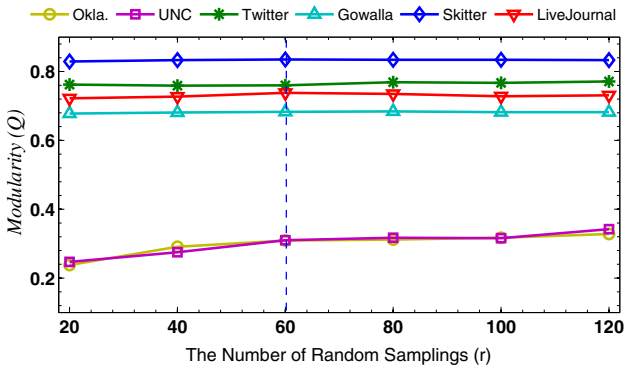


Fig. 3 Impact of the number of random samplings

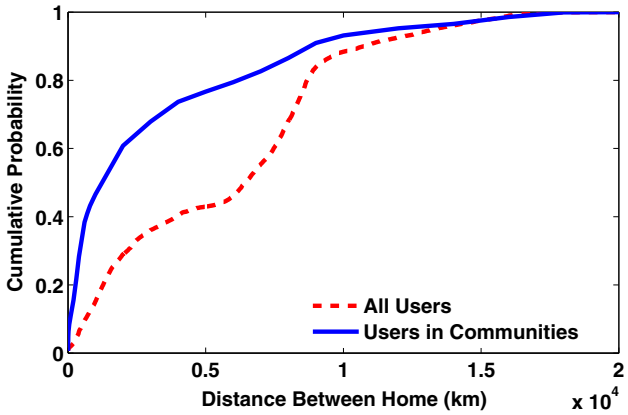


Fig. 5 Semantic validation on `Gowalla`

identified communities in terms of Q increases steadily with the increase of r. We find that the quality of SIMPLE tends to be satisfactory when r reaches 60, and is even better than Q values with 100 and 120 samplings on some networks, e.g., `Skitter` and `LiveJournal`. As a result, in the following experiments, unless stated otherwise, we set $r = 60$ and $\alpha \in [1, 10]$.

6.3 Comparison on effectiveness

In this subsection, we shall validate the performance of SIMPLE that is comparable to five baseline tools. We vary the number of communities K from 10 to 60 for SIMPLE, PaKmeans, METIS and FN. However, the K value is automatically set by SCD and BigClam. So we mark the K values in Fig. 4 for both methods. As the increase of scales, some tools run out of memory, which leads to missing lines in Fig. 4e, f.

As can be seen from Fig. 4 SIMPLE consistently shows the best performance in terms of the Q function. This well validates that SIMPLE can identify high quality community structures on various networks. Meanwhile, the Q values of METIS are obviously lower than those of SIMPLE, though SIMPLE invokes METIS in `reduce` for sampled graph partitioning. This implies the ensembling phase is critically important for the success of SIMPLE in community detection. Moreover, SCD tends to partition the network into a mass of small communities, which largely reduces the modularity. In contrast, though BigClam can limit the number of identified communities, its Q values are always minimum.

We exploit the users' check-in trajectory information provided by `Gowalla` for the semantic validation of identified communities. Since online friends tend to have closer home locations [5], we suppose that users in a same community also tend to live more closely. We use SIMPLE to partition the `Gowalla` network into 10 communities. To infer users' home locations from their check-in locations, we first discretize the world map into 0.25 by 0.25 cells (measured by latitude and longitude) and then defined the home location as the centroid of a cell with the most check-ins, as suggested by [5]. The home distance on earth's surface between any

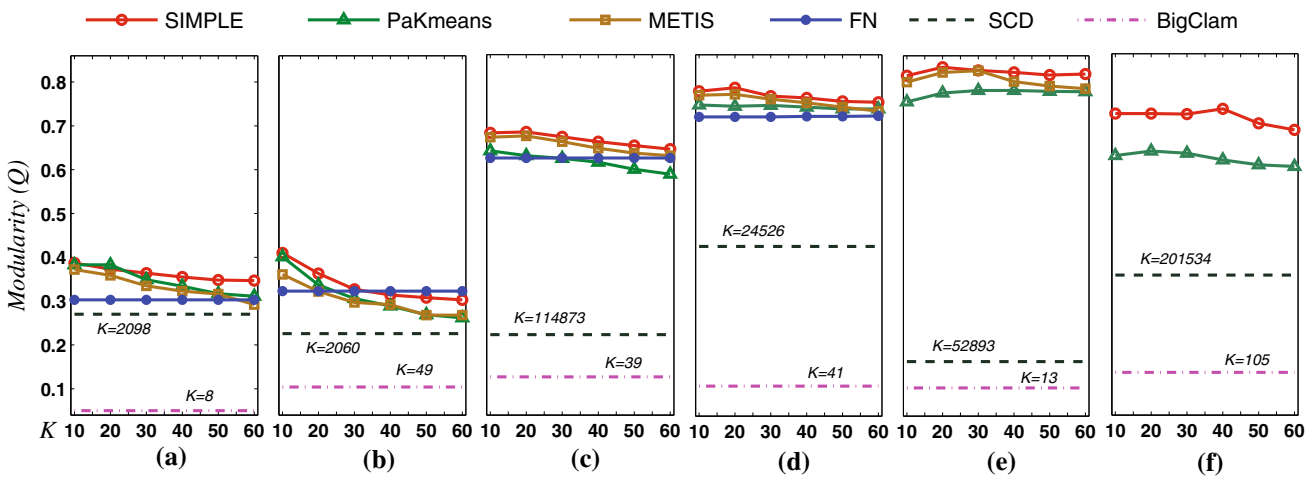


Fig. 4 Performance comparison in terms of modularity. **a** `Oklahoma`, **b** `UNC`, **c** `Twitter`, **d** `Gowalla`, **e** `Skitter` and **f** `LiveJournal`

two users was computed by using the `pos2dist` function in MATLAB. Figure 5 shows the cumulative distributions of all users and users within the same community. As can be seen, the home distance between any two users in a same community is statistically shorter than the distance between any two users of the whole network. This indicates that the communities identified by SIMPLE from `Gowalla` is semantically cohesive.

To sum up, compared with five baseline methods, SIMPLE could discover higher quality communities in terms of Q. As indicated by a case study on `Gowalla`, the communities returned by SIMPLE are semantically meaningful.

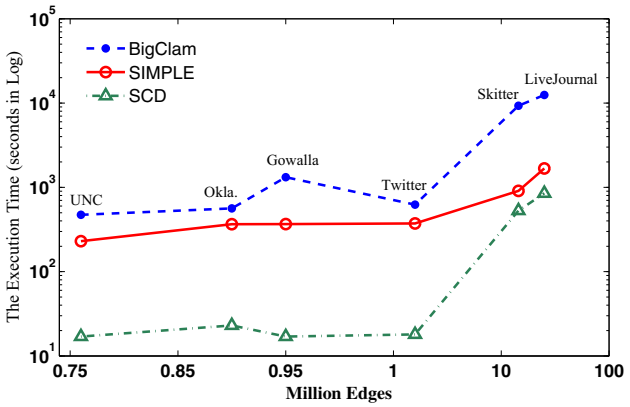


Fig. 6 Comparison on overall efficiency

6.4 Efficiency analysis

To analyze the efficiency of SIMPLE, we select BigClam and SCD for comparison. BigClam is a scalable matrix factorization method on a single machine, while SCD is a parallel WCC optimization method adopting the in-memory computing technique. As can be seen from Fig. 6, SIMPLE is generally more scalable than BigClam, yet less scalable than SCD. When the network becomes larger (e.g., `Skitter` and `LiveJournal`), SIMPLE reaches the same order of magnitude of SCD in terms of running time. The main reason is that the overhead cost of SIMPLE is much expensive than SCD, since SIMPLE uses MapReduce for distributed computing but SCD utilizes parallel computing on sharing memory. This indeed inspires us to employ the in-memory computing paradigm (e.g., Spark [36]) to further improve the scalability of SIMPLE in the future.

We further take a closer look at the efficiency of every phase in SIMPLE. Note that the sharding phase usually consumes a thimbleful of time (e.g., about 2 s), which is negligible here. Since the `map` phase handles large networks directly, we divide this phase into three sub-phases including reading, sampling and writing. Figure 7 shows the execution time of three phases (i.e., `map`, `reduce`, and d-KCC) on two large networks (i.e., `Skitter` and `LiveJournal`). From Fig. 7, we can observe several important facts. First, the execution time of `map` is mainly spent on writing the out-

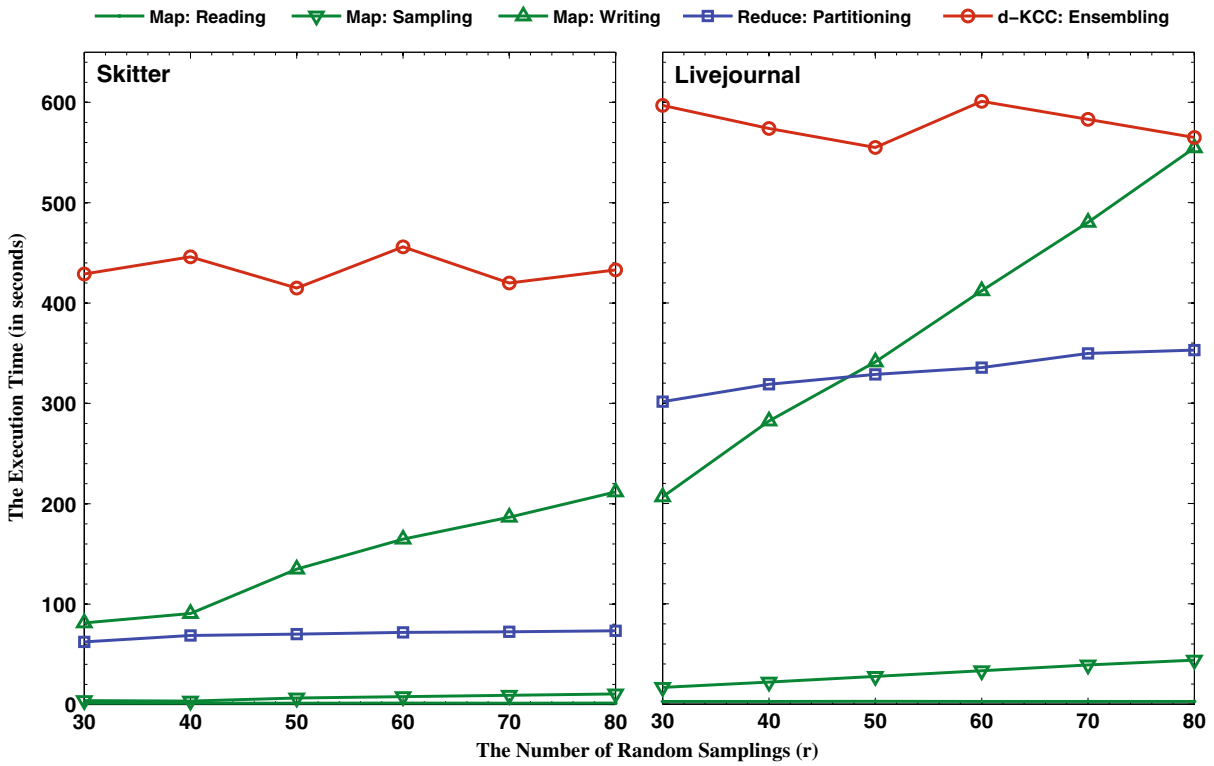


Fig. 7 Efficiency of every phase within SIMPLE

Table 2 Performance comparison on three sampling techniques

Network	SIMPLE-RAN	SIMPLE-FF	SIMPLE-MS
Oklahoma	**0.389**	0.365	0.364
UNC	0.400	**0.404**	0.386
Twitter	**0.770**	0.674	0.594
Gowalla	**0.677**	0.534	0.427

put results to HDFS. Second, the efficiency of reduce is stable and efficient, though the number of random samplings is increasing. This demonstrates the advantage of selecting METIS as the graph partitioning tool in this phase. Third, the execution time of d-KCC fluctuates slightly due to the random selection of the initial centroids.

In summary, SIMPLE exploits the MapReduce framework to obtain a satisfactory scalability, though being inferior to in-memory computing based method (e.g., SCD). However, SIMPLE can identify higher quality communities than SCD, as indicated in Sect. 6.3, which is enough to make up the slight inferiority on efficiency.

6.5 Comparison on sampling

Here, we show in experiments the advantage of random sampling used in SIMPLE. Four networks Oklahoma, UNC, Twitter and Gowalla are used, and two state-of-the-art sampling strategies, ForestFire sampling (FF) [14] and Metropolis sampling (MS) [9] are chosen as baselines. We run three different sampling strategies to generate 60 subgraphs with $\gamma = 0.3$ each. For each sampled network, METIS is employed to partition $K = 10$ communities. Finally, those 60 basic graph partitioning are ensembled by KCC to obtain a single $K = 10$ crisp partition. Note that FF has two parameters: forward (p_f) and backward (p_b) burning probability, both of which can be regulated to meet a given γ. As for MS, we can adjust the degree distribution ($p(d_v)$) to fit the given γ.

The overall performance comparison on three sampling techniques is listed in Table 2, also in terms of Q. Note that the optimal Q values are in bold. The random sampling performs slightly worse than FF on UNC, a small dense network, but best on other networks. Especially, the gap on the sparse network Gowalla becomes much more sensible. To illustrate why the random sampling always performs best, we further showcase in Fig. 8 the boxplots of Q values of 60 BPs generated by different sampling. Note that the Q value in Fig. 8 is computed on sampled graph with about 30 % edges. Results in Fig. 8 are consistent with Table 2, that is, higher quality of BPs leads to final communities with higher quality. Furthermore, we can also observe that the stability of random sampling is superior to other sampling techniques.

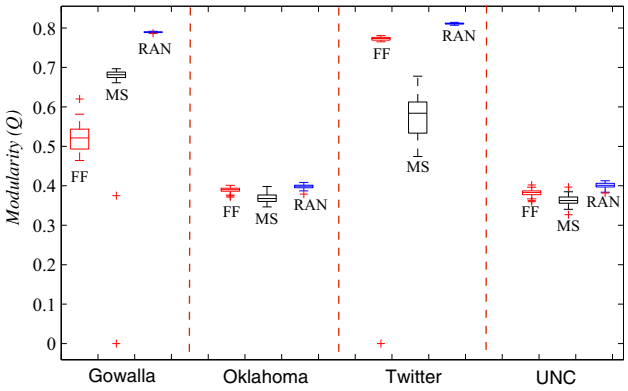


Fig. 8 Stability comparison on sampling techniques

7 Conclusion

This paper presents the SIMPLifying and Ensembling (SIMPLE) framework on top of MapReduce for parallel community detection. SIMPLE is indeed a flexible framework that is composed of three main phases: link sampling, sampled graph partitioning, and ensembling. These three phases are all implemented in MapReduce to ensure SIMPLE can handle large networks. Technical details of three phases are discussed. Particularly, we prove the errors introduced by random sampling can be controlled into a small range with a high confidence, even as the ramdom rate is very low. We also illustrate the reason that why we select METIS for sampled graph partitioning and KCC for ensembling. Experimental results on six real-world social networks have demonstrated that SIMPLE can successfully discover higher quality (indicated by Q) and meaningful (indicated by semantic analysis) communities. From the efficiency validation, SIMPLE exhibits good scalability and can efficiently handle large networks with ten millions of nodes and edges.

Acknowledgments This research was partially supported by National Natural Science Foundation of China under Grants 71571093, 71372188 and 61502222, National Center for International Joint Research on E-Business Information Processing under Grant 2013B01-035, National Key Technologies R&D Program of China under Grants 2013BAH16F01 and 2013BAH16F04, Industry Projects in Jiangsu S&T Pillar Program under Grant BE2014141, Natural Science Foundation of Jiangsu Province of China under Grant SBK2015042593, and Key/Surface Projects of Natural Science Research in Jiangsu Provincial Colleges and Universities under Grants 12KJA520001, 14KJA520001, 14KJB520015, 15KJB520012 and 15KJB520011.

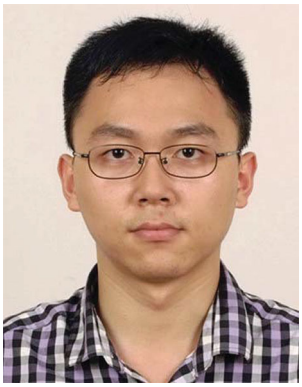
References

1. Apache Software Foundation: Apache Mahout: Scalable machine-learning and data-mining library. http://mahout.apache.org
2. Bernard, T., Bui, A., Pilard, L., Sohier, D.: A distributed clustering algorithm for large-scale dynamic networks. Clust. Comput. **15**(4), 335–350 (2012)

3. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. **2008**(10), P10008 (2008)
4. Chernoff, H.: A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. 493–507 (1952)
5. Cho, E., Myers, S.A., Leskovec, J.: Friendship and mobility: user movement in location-based social networks. In: Proceedings of KDD, pp. 621–628 (2011)
6. Clauset, A.: Finding local community structure in networks. Phys. Rev. E **72**, 026132 (2005)
7. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3–5), 75–174 (2010)
8. Gregori, E., Lenzini, L., Mainardi, S.: Parallel k-clique community detection on large-scale networks. IEEE Trans. Parallel Distrib. Syst. **24**(8), 1651–1660 (2013)
9. Hubler, C., Kriegel, H.P., Borgwardt, K., Ghahramani, Z.: Metropolis algorithms for representative subgraph sampling. In: Proceedings of the 2008 IEEE 7th International Conference on Data Mining, pp. 283–292. IEEE (2008)
10. Hui, P., Yoneki, E., Chan, S.Y., Crowcroft, J.: Distributed community detection in delay tolerant networks. In: Proceedings of 2nd ACM/IEEE International Workshop on Mobility in the Evolving Internet Architecture, p. 7. ACM (2007)
11. Karypis, G., Kumar, V.: Multilevel k-way hypergraph partitioning. In: Proceedings of the 36th Conference on Design Automation, pp. 343–348 (1999)
12. LaSalle, D., Karypis, G.: Multi-threaded modularity based graph clustering using the multilevel paradigm. J. Parallel Distrib. Comput. **76**, 66–80 (2015)
13. Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 631–636. ACM (2006)
14. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over time: densification laws, shrinking diameters and possible explanations. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, KDD '05, pp. 177–187. ACM (2005)
15. Leskovec, J., Mcauley, J.J.: Learning to discover social circles in ego networks. In: Advances in Neural Information Processing Systems, pp. 539–547 (2012)
16. Li, F., Ooi, B.C., Ozsu, M., Wu, S.: Distributed data management using mapreduce. ACM Comput. Surv. **46**(3), 31 (2013)
17. Moon, S., Lee, J.G., Kang, M.: Scalable community detection from networks by computing edge betweenness on mapreduce. In: International Conference on Big Data and Smart Computing (BIGCOMP), pp. 145–148. IEEE (2014)
18. Newman, M.E.: Fast algorithm for detecting community structure in networks. Phys. Rev. E **69**(6), 66–113 (2004)
19. Newman, M.E.: Modularity and community structure in networks. Proc. Natl. Acad. Sci. **103**(23), 8577–8582 (2006)
20. Newman, M.E.J.: The structure of scientific collaboration networks. Proc. Natl. Acad. Sci. **98**(2), 404–409 (2001)
21. Palla, G., Derenyi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435**, 814–818 (2005)
22. Papadopoulos, S., Kompatsiaris, Y., Vakali, A., Spyridonos, P.: Community detection in social media. Data Min. Knowl. Discov. **24**(3), 515–554 (2012)
23. Prat-Pérez, A., Dominguez-Sal, D., Brunat, J.M., Larriba-Pey, J.L.: Shaping communities out of triangles. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, pp. 1677–1681. ACM (2012)
24. Prat-Pérez, A., Dominguez-Sal, D., Larriba-Pey, J.: High quality, scalable and parallel community detection for large real graphs. In: 23rd International World Wide Web Conference, WWW '14, Seoul, 7–11 April, pp. 225–236 (2014)
25. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Phys. Rev. E **76**(3), 036–106 (2007)
26. Serrano, M.Á., Boguñá, M., Vespignani, A.: Extracting the multiscale backbone of complex weighted networks. Proc. Natl. Acad. Sci. **106**(16), 6483–6488 (2009)
27. Shi, J., Xue, W., Wang, W., Zhang, Y., Yang, B., Li, J.: Scalable community detection in massive social networks using mapreduce. IBM J. Res. Dev. **57**(3/4), 12-1 (2013)
28. Tang, L., Liu, H.: Community Detection and Mining in Social Media. Morgan & Claypool Publishers, San Rafael (2010)
29. Traud, A.L., Kelsic, E.D., Mucha, P.J., Porter, M.A.: Comparing community structure to characteristics in online collegiate social networks. SIAM Rev. **53**(3), 526–543 (2011)
30. White, T.: Hadoop: The Definitive Guide: The Definitive Guide. O'Reilly Media (2009)
31. Wu, J., Liu, H., Xiong, H., Cao, J.: A theoretic framework of k-means-based consensus clustering. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 1799–1805. AAAI Press (2013)
32. Wu, Z., Cao, J., Wu, J., Wang, Y., Liu, C.: Detecting genuine communities from large-scale social networks: a pattern-based method. Comput. J. **57**(9), 1343–1357 (2014)
33. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. In: Proceedings of ICDM, pp. 745–754 (2012)
34. Yang, J., Leskovec, J.: Overlapping community detection at scale: a nonnegative matrix factorization approach. In: Proceedings of the sixth ACM international conference on Web search and data mining, pp. 587–596. ACM (2013)
35. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. NSDI'12, pp. 2–2. USENIX Association, Berkeley, CA (2012)
36. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot topics in Cloud Computing, vol. 10, p. 10 (2010)
37. Zhang, Y., Wang, J., Wang, Y., Zhou, L.: Parallel community detection on large networks with propinquity dynamics. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 997–1006. ACM (2009)
38. Zhao, W., Ma, H., He, Q.: Parallel k-means clustering based on mapreduce. In: Proceedings of the 1st International Conference on Cloud Computing, CloudCom '09, pp. 674–679. Springer (2009)

Zhiang Wu received his Ph.D. degree in Computer Science from Southeast University, China, in 2009. He is currently an associate professor of Jiangsu Provincial Key Laboratory of E-Business at Nanjing University of Finance and Economics. He is the member of the ACM, IEEE and CCF. His recent research interests include distributed computing, social network analysis and data mining.

Guangliang Gao is currently pursuing the Ph.D. degree in Nanjing University of Science and Technology, Nanjing, China. His main research interests include data mining and social networks.



Zhan Bu received his Ph.D. degree in Computer Science from Nanjing University of Aeronautics and Astronautics, China, in 2014. He is currently a lecturer of Jiangsu Provincial Key Laboratory of E-Business at Nanjing University of Finance and Economics. He is the member of CCF. His recent research interests include social network analysis, complex network and data mining.

Jie Cao received his Ph.D. degree from Southeast University, China, in 2002. He is currently a professor and the dean of School of Information Engineering at Nanjing University of Finance and Economics. He has been selected in the Program for New Century Excellent Talents in University (NCET) and awarded with Young and Mid-aged Expert with Outstanding Contribution in Jiangsu province. His main research interests include cloud computing, business intelligence and data mining.