




# A Fast Parallel Community Discovery Model on Complex Networks Through Approximate Optimization

Shaojie Qiao , Nan Han, Yunjun Gao , *Member, IEEE*, Rong-Hua Li, Jianbin Huang, Jun Guo, Louis Alberto Gutierrez, and Xindong Wu , *Fellow, IEEE*

**Abstract**—Community discovery plays an essential role in the analysis of the structural features of complex networks. Since online networks grow increasingly large and complex over time, the methods traditionally used for community discovery cannot efficiently handle large-scale network data. This introduces the important problem of how to effectively and efficiently discover large communities from complex networks. In this study, we propose a fast parallel community discovery model called *picaso* (a parallel community discovery algorithm based on approximate optimization), which integrates two new techniques: (1) Mountain model, which works by utilizing graph theory to approximate the selection of nodes needed for merging, and (2) Landslide algorithm, which is used to update the modularity increment based on the approximated optimization. In addition, the GraphX distributed computing framework is employed in order to achieve parallel community detection over complex networks. In the proposed model, clustering on modularity is used to initialize the Mountain model as well as to compute the weight of each edge in the networks. The relationships among the communities are then simplified by applying the Landslide algorithm, which allows us to obtain the community structures of the complex networks. Extensive experiments were conducted on real and synthetic complex network datasets, and the results demonstrate that the proposed algorithm can outperform the state of the art methods, in effectiveness and efficiency, when working to solve the problem of community detection. Moreover, we demonstratively prove that overall time performance approximates to four times faster than similar approaches. Effectively our results suggest a new paradigm for large-scale community discovery of complex networks.

**Index Terms**—Community discovery, complex networks, distributed computing, graph theory, approximate optimization

## 1 INTRODUCTION

COMPLEX networks have become ubiquitous in our daily life. Such examples include online social networks, publication citation networks, customer transaction networks, and so forth. Due to the complex relationships between nodes, and the large cardinality of networks, these networks are referred to as “complex network” [1]. Community structure, which originates from complex networks, refers to a group of nodes which are aggregated into tightly

connected groups, where there is a high density of within-group edges and a lower density of between-group edges [2]. It is important for the purposes of research to understand the structural features, the evolution of communities, the propagation of information, points of interest recommendation, and other significant features. Community discovery is one of the most important and fundamental tasks in network analysis, and has applications in functional prediction in Biology [3]. Early research in community discovery for complex networks focuses primarily on small networks with simple structures, this is due to the computational difficulties of storing and analyzing large-scale node and edge information.

Our research is motivated by the following observations: (1) as social networks become more and more embedded in our everyday lives, this intuitively has led to a critical mass of users, e.g., there are 13.5 billions users being active in Facebook each month [4]. With the growth of social networks, traditional community detection algorithms do not scale to the large number of users, the complex relationships between them, or the rapid flux their relationships. (2) These increasingly complex and undetected features of large social networks represent missed opportunities for analyzing, correlating, and ultimately predicting the behavior of the users for the purposes of marketing, advertisement and internet public opinion control. (3) The study of the inner and intra structural features of communities in large-scale complex networks has direct practical theoretical

- S. Qiao is with the School of Cybersecurity, Chengdu University of Information Technology, Chengdu 610225, China. E-mail: sjqiao@cuit.edu.cn.
- N. Han is with the School of Management, Chengdu University of Information Technology, Chengdu 610103, China. E-mail: hannan@cuit.edu.cn.
- Y. Gao is with the College of Computer Science and Technology, Zhejiang University, Zhejiang 310027, China. E-mail: gaoyj@zju.edu.cn.
- R.-H. Li is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: rhli@szu.edu.cn.
- J. Huang is with the School of Software, Xidian University, Xi'an 710071, China. E-mail: jhhuang@xidian.edu.cn.
- J. Guo is with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China. E-mail: 624942464@qq.com.
- L.A. Gutierrez is with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180. E-mail: louisgutierrez2002@gmail.com.
- X. Wu is with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70503. E-mail: xwu@louisiana.edu.

Manuscript received 3 Jan. 2017; revised 26 Dec. 2017; accepted 29 Jan. 2018.  
Date of publication 7 Feb. 2018; date of current version 3 Aug. 2018.

(Corresponding authors: Yunjun Gao and Nan Han.)

Recommended for acceptance by Y. Zhang.

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.

Digital Object Identifier no. 10.1109/TKDE.2018.2803818

applications. And such applications necessitate efficient and accurate algorithms. (4) There exists some parallelized community detection algorithm proposed to process large-scale data. The work done by Wickramaarachchi et al. [5] show that they can achieve five fold performance improvement when using 128 parallel processors, but in turn requires even more resources to process larger networks.

In this study, we propose *picaso*, which is a new community detection model that is much faster than the most state of the art solutions, and improves the quality of community detection. *Picaso* is capable of discovering communities with more than 1 million nodes by less than 4 seconds, yet using 16 computers having modest 4 GB RAM.

In order to address current suboptimal state of efficiency and accuracy in existing community detection approaches in large-scale complex networks, we make the following contributions in this study:

- (1) Utilize graph theory for approximate optimization techniques in discovering large communities in complex networks. This is accomplished by taking into full consideration the structural features of communities, and in turn proposing new concepts and algorithms including: 1) the boundary nodes, 2) the chain group for storing the weight of nodes, 3) the Mountain model for choosing nodes to combine, and 4) the Landslide algorithm used for updating the weights of the chain-group structure and the nodes in communities of the entire network.
- (2) With the goal of efficiently processing large-scale network data, we propose *picaso* that is a parallel community discovery algorithm integrating the Mountain model and Landslide algorithm. *Picaso* can handle Big complex networks (i.e., having more than 10 million), while traditional serial detection algorithms do not work.
- (3) In order to test, verify and measure the effectiveness and efficiency of our proposed methods and algorithms, we conducted a series of real and synthetic experiments across large-scale complex networks. The results were compared against traditional and parallel algorithms

## 2 RELATED WORK

With the increased popularity and prevalence of complex networks, the area of research involving the study of structural features within these networks continues to garner more attention. There have been several seminal community detection algorithms proposed since the inception of this area of research, e.g., Newman et al. proposed the GN algorithm [2], the Fast-Newman algorithm based on the idea of modularity optimization [6] and the CNM algorithm [7]. These methods have been widely used in detecting communities in networks [8]. In order to improve the efficiency of community detection, Qiu et al. [9] partitioned the communities using the spectral bisection method, and the Lapacian matrix. Ruan et al. [10] presented a simple approach of combining content and link information in graph structures. Wu et al. [11] proposed a query biased node weighting scheme to reduce the irrelevant sub-graphs and accelerate community detection.

More recently, Zhang et al. [12] recommended improvements to the CNM algorithm by optimizing the update process of modularity. Prat-Pérez et al. [13] proposed the weighted community clustering model, which takes the triangle, instead of the edge, as the minimal structural motif, which indicates the presence of a strong relation in a graph.

Ferreira et al. [14] proposed a method which works to transform a set of time series data into a comparable network using various distance functions, in order to identify groups of strongly connected nodes in complex networks. Shan et al. [15] designed an overlapping community search framework for group queries. Huang et al. [16] formulated the community detection as a problem of finding the closest truss community. Li et al. [17] proposed a framework to determine communities in a multi-dimensional network based on the probability distribution of each dimension computed from the network. To make the process of community discovery more robust, Mahmood et al. [18] proposed a sparse spectral clustering algorithm based on  $\ell_1$  norm constraints to find a community label for each node. Whang et al. [19] proposed an efficient overlapping community detection algorithm using a seed expansion approach. The aforementioned methods for community detection have proven integral in advancing both the areas of research and application, however they do not address a fundamental problem, of which we attempt to address in this research, of handling large-scale complex network data in an effective and efficient manner. Dinh et al. [20] proposed an additive approximation algorithm for modularity clustering with a constant factor and they proved that a community structure with modularity arbitrary close to maximum modularity might bear no similarity to the optimal community structure of maximum modularity. Shiokawa et al. [21] proposed a very fast modularity-based graph clustering algorithm by incrementally pruning unnecessary vertices/edges and optimizing the order of vertex selections. It requires only 156 seconds on a graph with 100 million nodes and 1 billion edges. Differently, *picaso* is a parallel algorithm by applying two strategies, i.e., the Mountain Model and the Landslide strategy, which can help obtain high detection accuracy with the guarantee of good runtime performance.

In order to address the difficulty of processing network data, which for the purposes of this research can be considered Big Data, parallel algorithms were utilized. Prat-Pérez et al. [22] proposed a high quality, scalable and parallel community detection approach for large graphs. However, due to certain limitations, it is not appropriate for detecting overlapping communities. Wickramaarachchi et al. [5] presented an efficient approach to detecting communities in large-scale graphs by improving the sequential Louvain algorithm and parallelizing it on the MPI framework. Varamesh et al. [23] proposed a clique percolation algorithm (CMP) based on MapReduce to meet the necessary requirements of memory, CPU and I/O operations. The results demonstrate that when the number of nodes are greater than forty thousand, the execution time exceeds 1,000 seconds. Recently, Staudt et al. [24] parallelized the Louvain method to efficiently discover communities in massive networks. Moon et al. [25] utilized vertex-centric with MapReduce and GraphChi to detect large graphs in social networks. Lu et al. [26] proposed a conductance-based

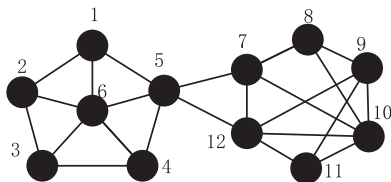


Fig. 1. Example of a simple network.

community detection algorithm for weighted networks, and designed an efficient data forwarding algorithm for delay tolerant networks. Qiao et al. [27] proposed a parallel algorithm for detecting communities in complex networks based on modularity, and designed new community merge and update strategies.

The parallel graph clustering models can be applied to detect communities. Meyerhenke et al. [28] proposed an effective parallel technique to partition large graphs of complex networks. Takahashi et al. [29] proposed a novel algorithm SCAN-XP that performs over Intel Xeon Phi to cluster large-scale graphs. In [30], an interactive and scalable graph clustering algorithm on multi-core CPUs was presented. Shun et al. [31] parallelized many of graph clustering algorithms in the shared-memory multicore setting. However, the proposed graph clustering models cannot be straightly applied to detect communities due to complex relationships between nodes in complex networks.

In order to address these fundamental challenges, the efficient discovery of communities, and in a timely and efficient manner, in this research we propose a novel community detection model based on approximate optimization, which is parallelized on the GraphX framework [32] to ensure fast computation. When compared with traditional algorithms, and parallel algorithms, we demonstrate that there is a clear and measurable increase in time performance. Additionally, prediction accuracy for this method is maintained at a very high level.

In the following sections, we will introduce the preliminaries and discuss the Mountain model and Landslide algorithm in Section 3. Section 4 addresses the implementation, and its effects on time complexity, of the parallel algorithm for the proposed model. In Section 5, we discuss the results of extensive experiments conducted on real and synthetic complex networks. Finally, we conclude our work and look forward into future work in Section 6.

### 3 MOUNTAIN MODEL AND LANDSLIDE STRATEGY

With the given constraints, the weight of edges and the index of communities, this paper proposes a new model designed at accelerating the phase of computing the modularity by implementing approximation optimization and graph theory. In addition, in order to make the process of updating weights more convenient, this research also introduces the new algorithm “Landslide”.

#### 3.1 Basic Concepts

A complex network is a graph with non-trivial topological features, it has the following properties: self-organization, self-similarity, small world, and scale-free.

Fig. 1 is an example of a network with twelve nodes and twenty three edges derived from a complex network.

**Definition 1 (Chain Group).** A Chain Group is denoted by  $CG=\{s, t, r\}$ , where  $s$  is the start node,  $t$  is the end node, and  $r$  is the weight between  $s$  and  $t$ , or the relation type.

It is worth to note that we use the chain-group structure to store the elementary network data in GraphX.

**Definition 2 (Boundary Node).** Given that  $BN=\{\sum (v_i, v_j) | v_i \in C, v_j \in C', e_{v_i v_j} \in E\}$  represents the set of boundary nodes, where  $v_i, v_j$  are distinct nodes from the communities  $C$  and  $C'$ , and  $e_{v_i v_j}$  is an edge in the edge set  $E$ .

In summary, the nodes between communities are boundary nodes, such as the nodes  $\{5, 7, 12\}$  in Fig. 1. It follows that the relationships among boundary nodes are more complex than the nodes in a community. In order to accurately distinguish the community where the boundary nodes belong to, we apply the following strategy: the proposed algorithm calculates the membership degree  $B(u, c) = k_u^c / k_u$  of the boundary node  $u$  belonging to some community  $c$ , where  $k_u^c$  represents the degree of node  $u$  in community  $c$ , and  $k_u$  is the degree of  $u$  in all communities. At last, we assign the node to the corresponding community in which it has the maximum membership degree.

**Definition 3 (Modularity).** Modularity is defined by the following equation [2]

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left( e_{ij} - \frac{d_i d_j}{2m} \right) \delta(c_i, c_j), \quad (1)$$

where  $e_{ij}$  represents the connected relation between node  $i$  and  $j$  in the adjacent matrix  $E$  of the network,  $m$  is the number of edges,  $V$  is the node set,  $d_i$  and  $d_j$  the degrees of node  $i$  and  $j$ , and  $c_i$  and  $c_j$  the communities where  $i$  and  $j$  stays in, respectively. If the community in which  $i$  belongs to is the same as that of which node  $j$  belongs to, then  $\delta(c_i, c_j)=1$ . Otherwise  $\delta(c_i, c_j)=0$ .

Given that relationships between communities is relatively difficult to identify from the global perspective, it follows that Eq. (1) is also difficult to calculate. Newman proposed a simplified equation as shown below [2]

$$Q = \frac{1}{2m} \sum_{i=1}^n \left( e_{ii} - \frac{d_i^2}{2m} \right), \quad (2)$$

where  $m$  represents the number of edges,  $i$  is the sequence number of a community,  $n$  is the number of communities,  $e_{ii}$  is the number of edges in the  $i$ th community, and  $d_i$  represents the sum of degrees of all nodes in the  $i$ th community. According to Ref. [2], when modularity reaches the maximum value, communities can best be detected.

#### 3.2 The Mountain Model

The Mountain model is integral in this research, and is based on modularity, approximate optimization, and graph theory. It sorts the chain groups by the weights of edges. Owing to the feature of community structures, some chain groups in a community may fall down while surrounding community may rise like mountains. Resolutely, a suitable number of chain groups at the top of mountains are chosen to form new communities.

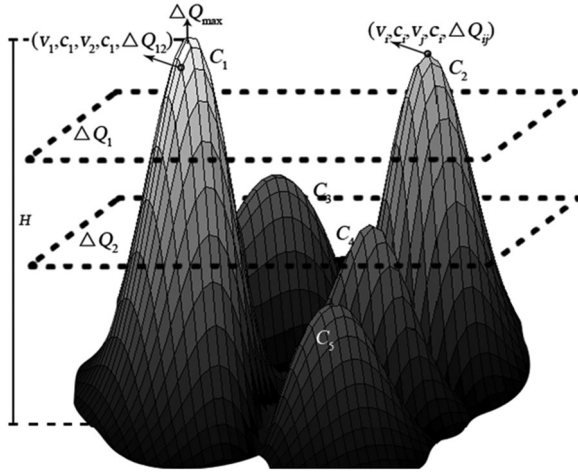


Fig. 2. Example of the mountain model.

**Definition 4 (Modularity Increment).** Assuming that communities  $i$  and  $j$  merged, the following equation [6] can be obtained to compute the modularity increment

$$\Delta Q = \frac{1}{m} \left( e_{ij} - \frac{d_i d_j}{2m} \right). \quad (3)$$

Where  $m$  represents the number of edges,  $e_{ij}$  denotes the number of edges between community  $i$  and  $j$ , and  $d_i, d_j$  represent the sum of degrees of all nodes in community  $i$  and  $j$ , respectively.

Based on Eq. (3), we can determine that the modularity increment grows with the modularity of  $Q$ .

**Lemma 1.** In any undirected graph, there must be two or more nodes which have the same degree.

**Proof.** If it is known that  $G$  is a network with  $n$  nodes ( $n \geq 2$ ), then there cannot exist any isolated nodes in  $G$ .

- $\because$  There are no isolated nodes,
- $\therefore$  The degree  $d$  of every node meets  $1 \leq d \leq n - 1$ ,
- $\therefore$  According to the Pigeonhole Principle,  $n$  nodes select  $n-1$  degree values, so there exists at least two nodes which have the same degree.  $\square$

**Property 1.** According to Lemma 1, there might exist several similar graph structures in a complex network. Therefore, it is true that there might exist some chain groups with similar modularity increments.

**Property 2.** According to the algorithm proposed in Ref. [6], nodes are clustered with the maximum  $\Delta Q$  at each iteration. Nodes are sorted sequentially by the  $\Delta Q$ s within a community, and the nodes with values larger than  $\Delta Q$  are chosen to be combined. This operation does not affect nodes which are outside of the given community, because the relationships of nodes from distinct communities are largely sparse.

Based on Properties 1 and 2, the picaso model calculates nodes' modularity increments and merge nodes with values larger than the minimum  $\Delta Q$ . According to the previously referenced theories, it can be inferred that when chain groups are sorted by  $\Delta Q$ , the chain groups with values larger than  $\Delta Q$ , within a given community, do not need to interact with other communities. Thus, the shape of the

network, after sorting, remains unchanged. This can be seen in Fig. 2, where the intersection represents a chain group,  $C$  represents a community, and  $H$  is the height of  $\Delta Q$ .

In Fig. 2, the summit of each mountain represents the maximum modularity increment  $\Delta Q$  w.r.t. each community. Each mountain is formed by the modularity increments of all nodes in this community. We can see five mountains formed by the community  $C_1, C_2, \dots, C_5$ . As shown in Fig. 2, when the maximum modularity increment is located, denoted by  $\Delta Q_{max}$ , only  $C_1$  is involved. Thus, there is only a need to merge the chain groups at the top of cluster  $C_1$ . If we take into account all of the CGs, and the intersection  $\Delta Q_1$  is chosen, it becomes the case where  $\Delta Q \geq \Delta Q_1$ ,  $C_1$  and  $C_2$  all become involved, but  $C_1$  remains independent from  $C_2$ . It follows then that all the nodes can be clustered to form two communities. Similarly, when the intersection  $\Delta Q_2$  is chosen, and  $C_1, C_2, C_3, C_4$  become involved, then all the nodes are clustered to form four communities.

**Definition 5 (Mountain Model).** The Mountain model is comprised of a five-tuple equation denoted by  $M = \{CG, D, H, \lambda, C\}$ . It sorts the CGs by  $\Delta Q$ , where CGs with similar  $\Delta Q$  values are placed on the same plane. It can be summarized as follows: CG: is the set of chain groups in a network,  $CG = \{CG_{uv} | u \in V, v \in V, e_{uv} \in E\}$ , where  $V$  is the vertex set,  $E$  is the edge set,  $CG_{uv} = (u, c_u, v, c_v, \Delta Q_{uv})$ , and  $c_u$  is the community index w.r.t. the node  $u$ ;  $D$ : is the degree set, where  $D = \{d_1, d_2, \dots, d_i\}$ , and  $d_i$  represents the degree of node  $i$ . When the CG needs to be updated,  $D$  is used to recalculate  $\Delta Q$ ;  $H$ : is the set of heights w.r.t. the mountains, where  $H = \{h_1, h_2, \dots, h_k\}$ ;  $\lambda$ : is a parameter which is used to determine how many CGs should be chosen. If  $\Delta Q \geq \Delta Q_\lambda$  ( $0 < \lambda < h_{max}$ ,  $\Delta Q_\lambda = CG_\lambda$ ), then, the corresponding CGs are chosen;  $C$ : is community set, and  $C = \{C_1, C_2, \dots, C_k\}$ ,  $C_k$  is expressed by  $\{v_1, v_2, \dots, v_q\}$ , where  $v_q$  is a vertex.

In the Mountain model, we apply the following approximate optimization technique: we find a reasonable parameter  $\lambda$ , then cluster CGs whereby  $\Delta Q \geq \Delta Q_\lambda$  work to form a small community, and then merge the communities. The above operations can reduce the costs of computation, and help to improve the utilization of resources.

### 3.3 Landslide Update Strategy

Given that the modularity-based community detection method needs to iteratively compute the modularity increment, and additional elements, including  $\{CG, C, D\}$ , need to be updated as well, there exists the challenge of performing these operations in a timely manner. Current methods require that the modularity increment be recalculated for the whole network, which can prove costly in regard to time, especially for complex networks with a large number of nodes and edges. Thus, we propose the Landslide update strategy.

In the phase of initializing the network, each node is considered to be a community, and  $\Delta Q$  is obtained by Eq. (3). After the operation which merges communities,  $\Delta Q$  is calculated by the following equation:

$$\Delta Q = \frac{1}{2m^2} \left( 2m * \sum_{u \in X, v \in Y} e_{uv} - \sum_{u \in X} d_u * \sum_{v \in Y} d_v \right), \quad (4)$$

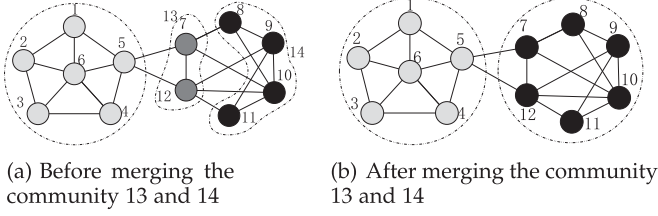


Fig. 3. Example of merging communities.

where  $m$  is the number of edges,  $e_{uv}$  denotes the edge between node  $u$  and  $v$ ,  $X$  and  $Y$  represent communities, and  $d_i, d_j$  represent the degrees of the node  $i$  and  $j$ , respectively.

**Property 3.** When the number of nodes and edges in the networks remain unchanged, after the community merging operation, the number of edges in the new community equals the sum of the edges in and between the two merged communities. Moreover, the number of edges between the new community and the other communities equals the sum of edges between the merged communities and other communities.

**Example 1.** Consider a simple network which includes twelve nodes as shown in Fig. 3a, where the cluster 13 and 14 represent two distinct communities. After merging communities 13 and 14, the results can be seen in Fig. 3b, where  $E_i$  represents a set of edges in the community  $i$ ,  $E_{i-j}$  represents a set of edges between the communities  $i$  and  $j$ , and  $e_{i-j}$  represents the edge between node  $i$  and  $j$ .

Before merging:

$$E_{13} = \{e_{7-12}\}, E_{14} = \{e_{8-9}, e_{9-10}, e_{10-11}, e_{9-11}, e_{8-10}\}, \\ E_{13-15} = \{e_{5-7}, e_{5-12}\}, E_{13-14} = \{e_{7-8}, e_{7-10}, e_{9-12}, e_{10-12}, \\ e_{11-12}\}.$$

After merging:

$$E_{16} = E_{13} \cup E_{14} \cup E_{13-14} = \{e_{7-12}, e_{8-9}, e_{9-10}, e_{10-11}, \\ e_{9-11}, e_{8-10}, e_{7-8}, e_{7-10}, e_{9-12}, e_{10-12}, e_{11-12}\}; \\ E_{15-16} = E_{13-15} \cup E_{14-15} = \{e_{5-7}, e_{5-12}\}.$$

**Corollary 1.** When the number of nodes and edges in a complex network remain unchanged, the  $\Delta Q$  between the new community and other communities can be determined based on the following: If a new community becomes connected with the already merged communities, the  $\Delta Q$  for this new community is the sum of its  $\Delta Q$  and that of the merged communities; Otherwise, the  $\Delta Q$  for this new community can be determined by subtracting the product of the sum of the fraction of the node's degree in  $Z$  and  $Y$  from the number of edges.  $\Delta Q$  can be obtained by Eq. (5)

$$\Delta Q_{XY} = \begin{cases} \Delta Q_{XY} + \Delta Q_{ZY}; < Z, Y > \in E, Z \subset X \\ \Delta Q_{XY} - 2a_Z * a_Y; < Z, Y > \notin E, Z \subset X \end{cases} \quad (5)$$

$$a_i = \frac{d_i}{2m} \quad (6)$$

$$a_Z = \sum_{i \in Z} a_i \quad (7)$$

$$a_Y = \sum_{j \in Y} a_j; Y \not\subset X, \quad (8)$$

where  $X$  represents the new community,  $E$  is the set of edges,  $Y$  the community that has not been merged,  $Z$  the community

TABLE 1  
Example of Updating the  $\Delta Q$

(a) Before merging the community 2 and 3

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0.035	-0.009	-0.009	0.029	0.029	-0.011	-0.009	-0.011	-0.014	-0.009	-0.014
2	0.035	0	0.035	-0.009	-0.014	0.029	-0.011	-0.009	-0.011	-0.014	-0.009	-0.014
3	-0.009	0.035	0	0.035	-0.014	0.029	-0.011	-0.009	-0.011	-0.014	-0.009	-0.014
4	-0.009	-0.009	0.035	0	0.029	-0.011	-0.009	-0.011	-0.014	-0.009	-0.014	-0.014
5	0.029	-0.014	-0.014	0.029	0	0.02	0.025	-0.014	-0.019	-0.024	-0.014	0.02
6	0.029	0.029	0.029	0.029	0.02	0	-0.019	-0.014	-0.019	-0.024	-0.014	-0.024
7	-0.011	-0.011	-0.011	-0.011	0.025	-0.019	0	0.032	-0.015	0.025	-0.011	0.025
8	-0.009	-0.009	-0.009	-0.009	-0.014	-0.014	0.032	0	0.032	0.029	-0.009	-0.014
9	-0.011	-0.011	-0.011	-0.011	-0.019	-0.019	-0.015	0.032	0	0.025	0.032	0.025
10	-0.014	-0.014	-0.014	-0.014	-0.024	-0.024	0.025	0.029	0.025	0	0.029	0.02
11	-0.009	-0.009	-0.009	-0.009	-0.014	-0.014	-0.011	-0.009	0.032	0.029	0	0.029
12	-0.014	-0.014	-0.014	-0.014	0.02	-0.024	0.025	-0.014	0.025	0.02	0.029	0

(b) After merging the community 2 and 3

	1	4	5	6	7	8	9	10	11	12	13
1	0	-0.009	0.029	0.029	-0.011	-0.009	-0.011	-0.014	-0.009	-0.014	<b>0.026</b>
4	-0.009	0	0.029	0.029	-0.011	-0.009	-0.011	-0.014	-0.009	-0.014	<b>0.026</b>
5	0.029	0.029	0	0.02	0.025	-0.014	-0.019	-0.024	-0.014	0.02	<b>-0.028</b>
6	0.029	0.029	0.02	0	-0.019	-0.014	-0.019	-0.024	-0.014	-0.024	<b>0.058</b>
7	-0.011	-0.011	0.025	-0.019	0	0.032	-0.015	0.025	-0.011	0.025	<b>-0.022</b>
8	-0.009	-0.009	-0.014	-0.014	0.032	0	0.032	0.029	-0.009	-0.014	<b>-0.017</b>
9	-0.011	-0.011	-0.019	-0.019	-0.015	0.032	0	0.025	0.032	0.025	<b>-0.022</b>
10	-0.014	-0.014	-0.024	-0.024	0.025	0.029	0.025	0	0.029	0.02	<b>-0.028</b>
11	-0.009	-0.009	-0.014	-0.014	-0.011	-0.009	0.032	0.029	0	0.029	<b>-0.018</b>
12	-0.014	-0.014	0.02	-0.024	0.025	-0.014	0.025	0.02	0.029	0	<b>-0.028</b>
13	<b>0.026</b>	<b>0.026</b>	<b>-0.028</b>	<b>0.058</b>	<b>-0.022</b>	<b>-0.018</b>	<b>-0.022</b>	<b>-0.028</b>	<b>-0.018</b>	<b>-0.028</b>	0

that has been merged with  $X$ ,  $\langle Z, Y \rangle$  the edges between  $Z$  and  $Y$ , and  $a_i$  the fraction of node  $i$ 's degree to the number of edges.

For the purposes of this research, Corollary 1 is utilized in order to help reduce the height of the mountains, as well as update  $\Delta Q$  in all  $CGs$ .

**Example 2.** In the network depicted in Fig. 1, after initialization, each node is viewed as a community, and  $\Delta Q$  is computed using Eq. (3). The result, before the communities have been merged, are shown in Table 1a, where the first row and first column represent the indices of the communities. Table 1b shows the results after communities two and three have been merged.

The calculation for the new  $\Delta Q$  of community 13 is denoted in bold in Table 1b. For example,  $\Delta Q$  of edge  $\langle 6, 13 \rangle$  (row 6, column 13) is 0.058, which equals the sum of 0.029 at  $\langle 2, 6 \rangle$ , and 0.029 at  $\langle 3, 6 \rangle$ , as shown in Table 1a.

As the communities gradually self-aggregate,  $\Delta Q$  in turn continues to decrease, ultimately converging to zero. As a result, smaller communities become clustered to form new larger communities, and the relationships that characterize these communities become more apparent and easier to understand.

In the Landslide algorithm, the approximate optimization technique is applied in order to approximate the boundaries that divide the nodes into different communities. This process can help improve the accuracy of community detection, and reduce unnecessary calculations for modularity increments.

Based on the above research, this paper works to present a new community discovery model for large-scale complex networks called "picaso" (a parallel community discovery algorithm based on approximate optimization), which is implemented using Spark along with GraphX. The primary steps include: 1) initializing the network based on Eq. (3), 2) computing the  $\Delta Q$  for each chain group, and establishing the Mountain model, 3) approximating  $\Delta Q$ , choosing multiple chain groups to form new communities, and updating  $\Delta Q$ , and finally 4) parallelizing the picaso model to discover community structures.

## 4 PARALLEL COMMUNITY DETECTION MODEL

The *picaso* algorithm is designed on the GraphX framework, but it cannot support the attributes of edges and nodes. In order to handle this problem, we store the node set  $V$  using a tuple  $(v, c)$  in *picaso*, where  $v$  represents the index of a node, and  $c$  is the index of the community which  $v$  belongs to. In addition, *picaso* stores the edge set  $E$  using a triplet  $(s, t, \Delta Q)$ , where  $s$  is the start node,  $t$  is the end node. The chain group can be obtained by computing the Cartesian product of  $V$  and  $E$ .

The essential steps of the *picaso* algorithm include: (1) parameter initialization, (2) building the Mountain model, (3) merging the nodes and updating, and (4) community generation. Note that, in the first step, the network data is loaded and stored in memory, duplicated edges are eliminated, and the indexes of nodes are reordered.

### 4.1 Parameter Initialization

In this phase, the task is to calculate the parameters for modularity incrementation w.r.t. chain groups, i.e., the number of nodes  $n$ , the degree of each node denoted by  $d$ , the number of edges  $m$ , and  $\Delta Q$ .

---

#### Algorithm 1. Parameter Initialization

---

**Input:** The preprocessed network  $N$ .

**Output:** A graph  $G$ .

1.  $G = \text{graphLoader}(D)$ ;
  2.  $m = \text{getEdges}(G)$ ;
  3.  $n = \text{getNodes}(G)$ ;
  4. disseminate  $m$  to each machine;
  5. **for** each node  $i \in V$  **do**
  6.    $d_i = \text{getDegree}(G, i)$ ;
  7.    $cId = i$ ;
  8.  $T = V \times E$ ;
  9. **for** each  $t \in T$  **do**
  10.    $\Delta Q_{ij} = 2 * (\frac{e_{ij}}{2m} - \frac{d_i d_j}{4m^2})$ ;
  11. **output**  $G$ ;
- 

As shown in Algorithm 1, the first step is to load the network data into memory (line 1), then calculate the number of edges  $m$  (line 2) and the number of nodes  $n$  (line 3) and disseminate  $m$  to each machine (line 4). The second step is to compute the degree of each node (lines 5-6), and specifies the node's community index to be its node index (line 7). The third step is to form chain groups by using the Cartesian product of  $V$  and  $E$  (line 8), which determines  $\Delta Q$  w.r.t the chain group (lines 9-10). Lastly, the new graph  $G$  is outputted (line 11).

### 4.2 Constructing the Mountain Model

After initializing the chain group, the Mountain model is constructed, which works to sort the chain groups by their  $\Delta Q$ . According to Definition 5 and Corollary 1, it is known that the peak of each mountain is mutually-exclusive, thus suitable chain groups are chosen for merging at the top of the mountains so as to form smaller communities with an acceptable  $\lambda$  parameter. The new index is allocated to the new community. The algorithm is given below:

The basic idea of Algorithm 2 is given as follows:

- (1) Obtain the maximum height of mountains based on Definition 4 (line 1), compute the parameter  $\lambda$ , and determine the validity of  $\lambda$  (line 2).
- (2) Obtain the chain group set  $CG$  by the taking Cartesian product of  $V$  and  $E$  (line 3).
- (3) Choose the chain groups in  $CG$  where  $\Delta Q \geq \Delta Q_\lambda$ , and form a new set  $S$  (lines 4-6).
- (4) Compute the connect component of  $S$ , where nodes in the same connect component belong to the same community (line 9). Allocate a new index for the newly-formed community (line 10), remove the nodes that have been allocated (line 11), and output the preliminarily dividing community set  $C$  (line 12).

---

#### Algorithm 2. Mountain Model Construction

---

**Input:** The graph  $G = (V, E)$ .

**Output:** A preliminarily dividing community set

- $$C = (C_1, C_2, C_3, \dots).$$
1.  $H = \text{getHeight}(G)$ ;
  2.  $\lambda = 2 * |E|/|C|$ ;
  3.  $CG = V \times E$ ;
  4. **for** each  $t \in CG$  **do**
  5.   **if**  $\text{getAttr}(t) \geq \Delta Q_\lambda$  **then**
  6.      $VT = \text{insert}(t)$ ;
  7. **for**  $VT \neq \emptyset$  **do**
  8.    $n = n + 1$ ;
  9.    $S' = \text{connectComponent}(S)$ ;
  10.    $C = \text{insert}(n, S')$ ;
  11.    $S = \text{remove}(S, S')$ ;
  12. **output**  $C$ ;
- 

### 4.3 Community Merging and Update

---

#### Algorithm 3. Community Merging and Update

---

**Input:** The community set  $C$  that needs to be merged.

**Output:** The graph  $G$  after being updated.

1. **for** each edge  $e \in E$  **do**
  2.   **if**  $s \in C$  or  $e \in C$  **then**
  3.      $\{X, Y\} = \text{getCommunity}(s, t, C)$ ;
  4.     **for** each node  $i \in X$  and  $j \in Y$  **do**
  5.       **if**  $e_{ij} \in E$  **then**
  6.          $\Delta Q_{XY} = \Delta Q_{XY} + \Delta Q_{ij}$ ;
  7.       **else**
  8.          $\Delta Q_{XY} = \Delta Q_{XY} - \frac{d_i d_j}{2m^2}$ ;
  9.     **for** each  $c \subset C$  **do**
  10.      **for** each  $k \in c$  **do**
  11.        $d_c = d_c + d_k$ ;
  12.     **for** each  $v \in V$  **do**
  13.      **if**  $v \in C$  **then**
  14.        $cId = \text{getNewID}(v, C)$ ;
  15. **output**  $G$ ;
- 

An important next step is to merge and update the chain groups after finding the preliminary communities found by Algorithm 2. This process includes the community index of nodes, the degree of nodes in the communities, and  $\Delta Q$ . The main steps of community merging and update are given in Algorithm 3, which includes:

- (1) Find the communities  $X$  and  $Y$  that contain the start node  $s$ , and the end node  $t$  (lines 1-3). For the

communities  $i$  in  $X$  and  $j$  in  $Y$ , if there are edges connecting  $i$  and  $j$ , the value of  $\Delta Q$  for  $X$  and  $Y$  should be added by the  $\Delta Q$  between  $i$  and  $j$  (line 6); otherwise, it should be minus two fold the product of  $d_i/2m$  and  $d_j/2m$  (line 8).

- (2) Calculate nodes' degrees in new communities. For each new community in  $C$ , the degree equals the sum of nodes' degrees belonging to it (lines 9-11).
- (3) Lastly, the community in which the nodes belong to are determined. This is done by visiting each vertex  $v$  in  $V$ , if  $v$  belongs to a new community, obtain the index of this community, and specify the attribute of this vertex to be this index (line 14). Then, output the new graph  $G$  (line 15).

#### 4.4 Community Generation

After clustering the distinct communities, the redundant data is eliminated. This is because only the node and the sequence number of the community in which the node belongs to is needed. The data structures stored in GraphX include  $V=(vId, cId)$ ,  $E=(s, t, \Delta Q)$ ,  $D=(d_1, d_2, \dots, d_m)$ .

---

#### Algorithm 4. Community Generation

---

**Input:** The updated graph  $G$ .

**Output:** The community  $C$ .

1. **for** each  $v \in G$  **do**
  2.   **if**  $cId \in C$  **then**
  3.      $t = getCommunity(cId)$ ;
  4.      $c = insert(t, vId)$ ;
  5.      $C = insert(cId, c)$ ;
  6.   **else**
  7.      $C = insert(cId, vId)$ ;
  8. **output**  $C$ ;
- 

The main steps of Algorithm 4 include:

- (1) Visit all the nodes (line 1), if one node's community  $cId$  have been stored, add this node to the community with  $cId$  (lines 2-5); otherwise, create a new community to store it (lines 6-7).
- (2) Output the community set  $C$  which is stored in the HDFS file system (line 8).

#### 4.5 Parallel Community Discovery Based on GraphX

The GraphX-based parallel community discovery model has the following properties [32]: (1) a data model consisting of a series of chain-groups to graph data; (2) a coarse-grained data-parallel programming model composed of deterministic operators including map, group-by, and join; (3) a scheduler that divides each job into a directed acyclic graph of community detection tasks, where each task runs on a partition of data.

In the *picaso* model, the parallel community discovery in a distributed dataflow framework is viewed as a sequence of join operations and group-by operations.

In the join phase, vertex properties represented by  $V = (vId, cId)$ , and edge properties  $E = (s, t, \Delta Q)$  are joined to form the chain-group triplets consisting of each edge and its corresponding source and destination vertex properties.

In the group-by phase, the triplets are grouped by source or destination vertex to construct the neighborhood of each

vertex, and merge and update the chain-groups by Algorithm 3 after finding the preliminary communities by Algorithm 2.

By iteratively applying the above phases to calculate the modularity increment  $\Delta Q$  of each node and update the node properties until converging to the minimum modularity increment (the optimal value is zero).

#### 4.6 Time Complexity Analysis

For a complex network denoted by  $CN = (V, E)$ , with  $n$  nodes and  $m$  edges, the *picaso* algorithm visits all edges once in the phase of preprocessing. It needs to visit all edges and nodes again in order to obtain the attribute of edges and nodes in the parameter initialization phase. Thus, the time complexity of these two phases is equal to  $O(n + 2m)$ . For community generation, it needs to visit all nodes again, which makes the time complexity equal to  $O(n)$ .

The main phases of *picaso* include: (1) Mountain model construction and (2) Community merging and updating.

- (1) For the first phase, all edges are visited while the height of each mountain is obtained, the algorithm searches for the chain group that has a  $\Delta Q$  bigger than  $\Delta Q_\lambda$ . Next, the algorithm traverses nodes and edges in the subgraph one time in order to find the connected components. Assuming that there are  $x$  nodes and  $y$  edges that need to be merged in each round of operation, the time complexity of this step equals  $O(2m + x(x + y))$ .
- (2) For the second phase, *picaso* finds edges and nodes that need to be updated, and modifies their attributes after obtaining the new attributes. The time complexity for this process is  $O(2m + n)$ . In the phase of computing the new attributes, it is assumed that the number of communities that need be merged equals  $q$ . In general, the number of edges in these communities is equal to  $r$  times of the number of nodes, so the time complexity of this phase is  $O(r * x^3 / q^2)$ .

Based on the above discussion, the time complexity of these two phases is equal to  $O(4m + n)$ . It can be concluded then that in the worst case, where there is only one small community having  $x$  nodes can be detected each time, the time complexity is equal to  $O((4m + n) * n/x)$ . In the best case, there are  $y$  communities which contain  $x$  nodes on average that can be detected each time, the time complexity is equal to  $O((4m + n) * n/(xy))$ .

## 5 EXPERIMENTS

### 5.1 Experimental Setup

In order to evaluate the effectiveness and efficiency of the proposed algorithm, a variety of datasets as shown in Table 2 were used during the experimentation: (1) five synthetic large-scale complex network datasets, randomly generated by the LFR benchmark algorithm [33]; (2) four real complex network datasets, obtained from the Stanford Network Analysis Project (SNAP) [34]; (3) five real small network datasets, which were used primarily for visualization of the discovered communities.

The LFR benchmark network generation algorithm was proposed by Lancichinetti, which can generate networks with real network features according to the input parameters. These types of datasets are especially useful in

TABLE 2  
Description of Datasets

(a) Synthetic complex network datasets				
Name	No. of nodes( $V$ )	No. of edges( $E$ )	$\mu$	Average degree( $2E/V$ )
v-1w	10,000	76,864	0.3	15.3728
v-10w	100,000	1,522,597	0.3	30.4519
v-50w	500,000	7,477,625	0.3	29.9105
v-100w	1,000,000	14,907,384	0.3	29.8148
v-1000w	10,000,000	154,831,275	0.3	30.9663
(b) Real complex network datasets				
Name	No. of nodes	No. of edges	Average degree	Description
com-DBLP	317,080	1,049,866	6.6221	DBLP collaboration network
com-Amazon	334,863	925,872	5.5299	Amazon product network
com-Youtube	1,134,890	2,987,624	5.2651	Youtube social network
com-LiveJournal	3,997,962	34,681,189	17.3494	LiveJournal social network
(c) Small real network datasets				
Name	No. of nodes	No. of edges	Average degree	Description
strike	24	38	3.1667	employees relationship network
polbooks	105	441	8.4	American politics book network
football	115	616	10.713	college football team network of USA
jazz	198	2,742	27.697	jazz musician collaborator network
facebook	5,000	8,194	3.2776	5,000 subnetworks derived from facebook

estimating the accuracy of community detection. The commonly used parameters in this algorithm include the following: the number of nodes  $n$ , the average degree of nodes  $k$ , the number of nodes in the smallest community  $C_{min}$ , the number of nodes in the largest community  $C_{max}$ , and the mixing parameter  $\mu$  (which can range from 0 to 1). The greater the  $\mu$ , the less obvious the community structure.

The *picaso* algorithm was developed by the Scala language on the Spark platform. The Spark cluster contains 16 computers (one master and 15 servant nodes) with Intel E5620 processor having 4G memory. Each algorithm runs three times, and the average value is used for evaluation. The algorithms which were compared include non-Overlapping Community Detection Idea (OCDI) proposed by Zhang et al. [12], Detecting Big Community based on Spark (DBCS) proposed by Qiao et al. [27], Parallel Louvain Method with Refinement (PLMR) designed by Staudt et al. [24], *picaso- $\alpha$*  which is a serial implementation of *picaso*, a parallel algorithm for community detection. OCDI is an efficient and accurate community discovery algorithm, but has difficulties scaling to large-scale network data. This algorithm, for the purposes of these experiments, is the main algorithm which is compared to *picaso- $\alpha$* . DBCS, which was also developed on Spark, is mainly compared with *picaso*. PLRM is a parallel Louvain method by an additional move phase after each prolongation.

The differences between DBCS and *picaso* lie in the following aspects: (1) the *picaso* model uses the Mountain model which is proposed by us based on modularity, approximate optimization, and graph theory. The Mountain model can partition the social graph into an initial community set, and use the Landslide algorithm to merge and update the community set, which can decrease the cost of community aggregation; (2) *picaso* chooses a large amount of nodes at the top of mountains to merge periodically, which helps reduce the cost of data transmission and makes use of Spark clusters to reduce the calculation delay and

waiting time; (3) *picaso* uses the proposed chain-group structure to store the elementary network data in GraphX.

**Definition 6 (Detection Accuracy).** *Detection accuracy* [27] is defined as the fraction of correctly detected nodes in communities to the number of all nodes, it is shown

$$DA = \frac{1}{n} \sum_{i=1}^k \max\{C_i \cap C_j | C_j \in C'_i\} (j = 1, 2, \dots, l), \quad (9)$$

where  $C_i$  represents the true community set,  $C'_i$  is the discovered community,  $\max\{C_i \cap C_j | C_j \in C'_i\}$  is the maximum public subset between  $C_i$  and  $C'_i$ ,  $n$  is the number of nodes,  $k$  is the number of real communities, and  $l$  is the number of discovered communities.

Clustering coefficient [35] is an important evaluation criteria in community discovery. It is often used to analyze the community structure and the search performance.

**Definition 7 (Clustering Coefficient, CC).** *Clustering Coefficient* is defined as follows:

$$C_k = \frac{2 \sum_{a,c \in N} |e_{ac}|}{d_k(d_k - 1)}. \quad (10)$$

Where  $C_k$ , given that  $C_k \in [0,1]$ , is the clustering coefficient of node  $k$ ,  $N$  represents the boundary node set,  $a$  and  $c$  represent two boundary nodes,  $e_{ac}$  represents the edge between  $a$  and  $c$ , and  $d_k$  is the degree of  $k$ . The *CC* of the entire network is equivalent to the average value of all nodes' *CC*

$$\bar{C} = \frac{1}{n} \sum_{i=1}^n C_i. \quad (11)$$

By Ref. [35], the fact does hold: if the *CC* of most communities is three fold of the entire network of *CC*, the detected community structure is significative and valuable.



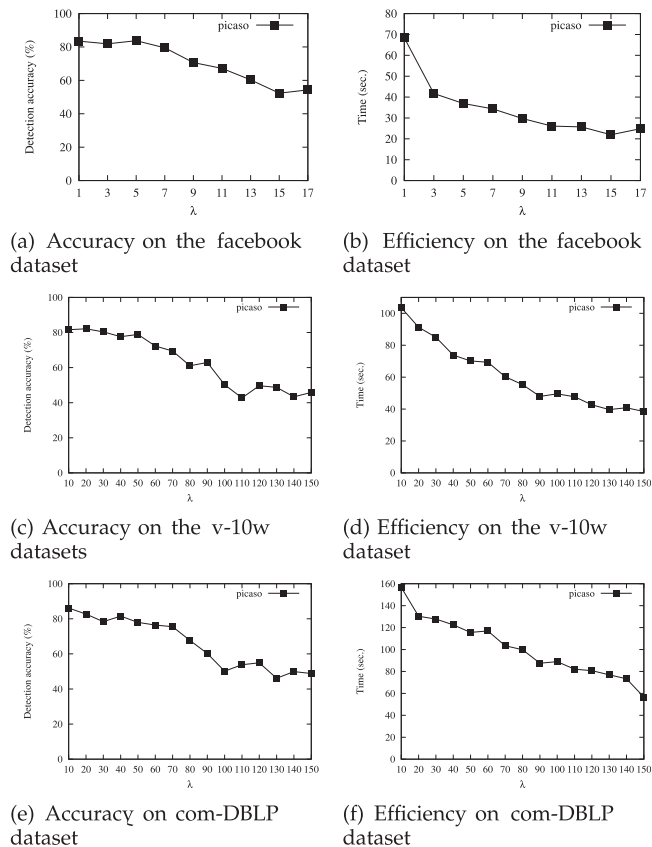


Fig. 4. Accuracy and efficiency of picaso by distinct  $\lambda$  values.

In this study,  $DA$  and  $CC$  are mainly used to evaluate the accuracy of community discovery.

## 5.2 Parameter Tuning

In an effort to make the comparison between the various algorithms used in the experiments, the parameter  $\lambda$  used in the Mountain model was adjusted accordingly. Picaso chooses chain groups at the top of Mountain model to approximately merge into communities by using the parameter  $\lambda$ , thus the selection of  $\lambda$  becomes integral to performance. In this set of experiments, it has been observed that varying the value of  $\lambda$  for picaso can have a distinct effect on the  $DA$  and execution time. The results of this experimentation are shown in Fig. 4.

According to Fig. 4  $\lambda$  increases, the  $DA$  of picaso gradually decreases under different datasets. In contrast, execution time appears to be reducing in the process. This is because picaso chooses chain groups to merge which have boundary nodes that minimize discrimination among communities. In particular, when the height of the Mountain model becomes low, picaso may choose too many chain

TABLE 3  
Detection Accuracy on Real Small Network Datasets

	strike	polbooks	footall	jazz	facebook
OCDI	100%	84.02%	89.28%	87.36%	84.43%
picaso- $\alpha$	100%	79.24%	86.16%	72.37%	81.92%
DBCS	100%	85.31%	90.59%	90.59%	83.09%
PLMR	100%	80.24%	79.27%	67.68%	78.51%
picaso	100%	82.36%	81.42%	70.73%	81.27%

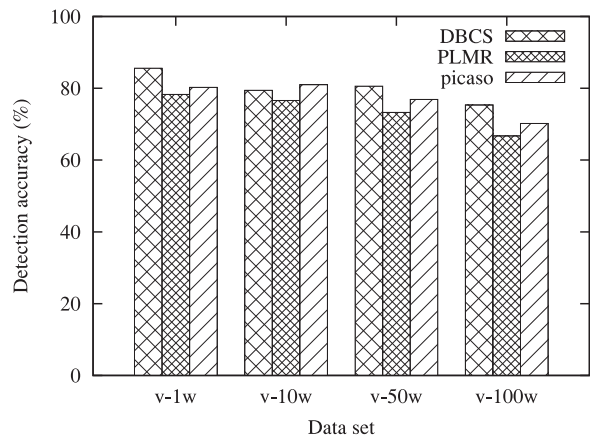


Fig. 5. Detection accuracy on synthetic network datasets.

groups to merge, which could increase the chance that an incorrect community partition. When  $\lambda$  grows, there are more chain groups to be selected, thus the computational resources can be fully utilized, and the number of merging operations can be greatly reduced. By Fig. 4, it can be concluded that, in regard to the Facebook dataset, when  $3 < \lambda < 7$ , the  $DA$  is relatively high, and runtime drops significantly. For the v-10w dataset, when  $10 < \lambda < 30$ , the  $DA$  is also high. For the com-DBLP dataset, when  $10 < \lambda < 40$ , results demonstrate that picaso works well.

It is of interest to note that the average degree of nodes ( $\frac{2|E|}{|C|}$ ) appears within a reasonable range of the  $\lambda$  values for multiple datasets. To keep the generality of the algorithms, we specify  $\lambda$  to  $\frac{2|E|}{|C|}$ , where  $|E|$  is the number of edges, and  $|C|$  is the number of communities.

## 5.3 Community Detection Accuracy Comparison

In this study, we use detection accuracy to evaluate the quality of community discovery. Table 3, Figs. 5 and 6 show the  $DA$  of each algorithm for various datasets.

According to Table 3, Figs. 5 and 6, this research can conclude the following:

1) For small network datasets, OCIDI, DBCS and picaso can obtain high  $DA$  values, usually more than 80 percent. The average  $DA$  of picaso is only about 5.86 percent lower than that of OCIDI, only about 6.76 percent lower than that of DBCS, and 2.02 percent higher than that of PLMR. This is because picaso chooses chain groups at the top of the

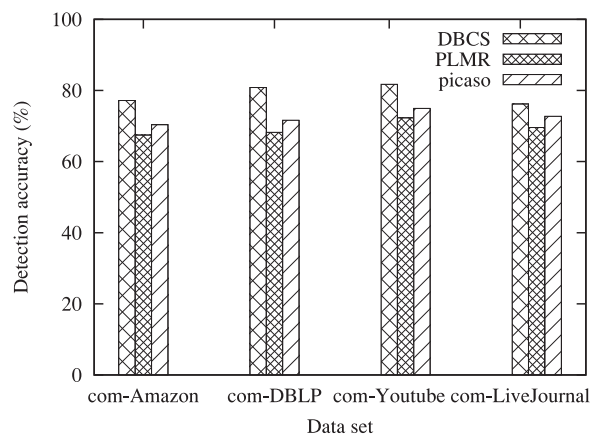


Fig. 6. Detection accuracy on large-scale real network datasets.

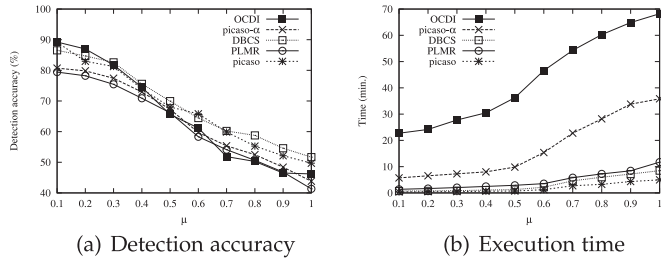


Fig. 7. Impact analysis of the  $\mu$  parameter on v-1w dataset.

Mountain model to form new communities via approximate optimization. When the communities gradually merge, the height of the mountains are reduced. Picaso has a lower recognition rate for the boundary nodes at the bottom of mountains, which causes  $DA$  to decrease. However, the gap between picaso, OCDI, DBCS and PLMR is very small, because the portion of boundary nodes are small in number. Thus the majority of nodes can still be correctly partitioned. For the jazz dataset, the  $DA$  of picaso is about 20 percent less than that of OCDI and DBCS, since the average node degree of jazz is 27.697. This is too large to detect most of nodes correctly, so  $\lambda = \frac{2|E|}{|V|}$  is not an optimal choice for this dataset, an appropriate value must be specified, which will be further discussed in the following section.

2) In Table 3, it can be seen that the serial algorithm picaso- $\alpha$  has a lower  $DA$  than picaso. This is because picaso- $\alpha$  selects a large number of nodes to merge, and works similar to that of the OCDI algorithm, making it unnecessary to obtain the connected branch subgraphs like picaso. So, picaso- $\alpha$  differs from picaso in that how the modularity increment is calculated, and as a result its  $DA$  is lower that of picaso.

3) For the synthetic data sets as shown in Fig. 5, the  $DA$  of picaso is nearly the same as DBCS, only about 3.165 percent less than that of DBCS, and 3.343 percent higher than that of PLMR on average. This is because the average node degree of each synthetic dataset is relatively large, and the community structure of the network is intuitive. In other words, the  $\mu$  value is relatively small. Ownership of the boundary nodes for the given communities is easy to see, and this is because picaso accurately predicts the communities for which the nodes at the top of mountains belong to. This makes it suitable to handle large-scale network data. In addition, the new update strategy applied in picaso, when compared with other algorithms, can obtain more accurate  $\Delta Q$  values, this all but guarantees that picaso produces a consistently high  $DA$  value.

4) In Fig. 6 the  $DA$ s of picaso, DBCS and PLMR both exceed 70 percent, with a small advantage to DBCS over picaso, and picaso outperforms PLMR with a small gap. This is because the connections between communities is very complex in real network datasets, and the connectivity between boundary nodes tends to be sparse. This makes it difficult to identify boundary nodes for picaso and PLMR, and since the average node degree of real communities tends to be small, it follows that the gap between the modularity increment and the height of the Mountain model are also small. Thus the number of boundary nodes increases, and picaso has a slight disadvantage when compared to DBCS.

When using the LFR benchmark program to generate the network data, the parameter  $\mu$  determines whether or not

the network has any clear community structures. The greater the value for  $\mu$ , the more unclear the community structure may be. Therefore, for the purposes of these experiments, various  $\mu$  values are generated for various networks that go beyond the v-1w dataset, so as to observe the impact of  $\mu$  on  $DA$  and runtime efficiency when using the given algorithms.

Fig. 7a shows the  $DA$  for different algorithms on the v-1w dataset as  $\mu$  is increased, while the number of nodes and the average node degree remain unchanged. As we can conclude the following from Fig. 7a: (1) the  $DA$  of each algorithm drops as  $\mu$  increases. This is because when  $\mu$  grows, the community structure become less obvious, and in turn it becomes difficult to partition nodes into the correct communities. it can be concluded that  $\mu$  has a strong effect on  $DA$ . (2) When  $\mu$  is small,  $DA$  of both picaso and DBCS are lower than that of OCDI. However, when  $\mu$  equals 0.45 or higher, the  $DA$  of picaso and DBCS are higher than that of OCDI. An improvement of 4.877 and 6.343 percent on average, respectively. This implies that picaso performs better when handling network data with ambiguous community structure, when compared to the traditional algorithms. The reason for this is that picaso uses the Mountain model to cluster representative nodes at the peak into communities. In addition, the proposed Landslide algorithm can help improve accuracy for calculating the modularity increment. (3) the  $DA$  of picaso has a slightly lower value than that of DBCS and has a slightly higher value than that of PLMR. The reason for this is that the community structure increases in ambiguity as  $\mu$  increases, which renders the ownership of boundary nodes relatively hard to distinguish.

By Fig. 7b, we can see that the execution time of each algorithm grows with  $\mu$ . This can be explained by the reason that as  $\mu$  increases which means there are several overlapping nodes and the community structures are hard to distinguish, all algorithms need to spend more time on partitioning these overlapping nodes. Additionally, we find that the parallel picaso, PLMR and DBCS models outperform the serial OCDI and picaso- $\alpha$  models with a big gap, which shows the advantage of parallel computing models on multiple processors.

#### 5.4 Community Recognition Quality Analysis

Fig. 8 shows the whole network clustering coefficient and the average clustering coefficient for picaso in real large-scale networks.  $CC$  is used to represent the clustering coefficient. For each network, five communities are randomly selected, and  $CC$  is calculated for each one. It is worthwhile to note that the formula of community selection is  $i = k^*$  ( $n\%512$ ), where  $i$  is the community sequence number,  $k = 1, 2, \dots, n$ , and  $n$  is the number of communities.

Fig. 8 shows the following: (1) the results of the community  $CC$  and whole network  $CC$  are disparate, and all community  $CC$ s are greater than the whole network  $CC$ ; (2) most of the community  $CC$ s are three fold higher than the whole network  $CC$ , only  $c_3$  in Fig. 8a,  $c_2$  and  $c_5$  in Fig. 8d are less than three fold the whole network  $CC$ . This strongly suggests that picaso has high-quality community recognition rate. The reason for this is that picaso merges nodes at the top of Mountains one at a time, and distributes most nodes into the correct communities. For picaso, it may be difficult to handle boundary-nodes which are not involved

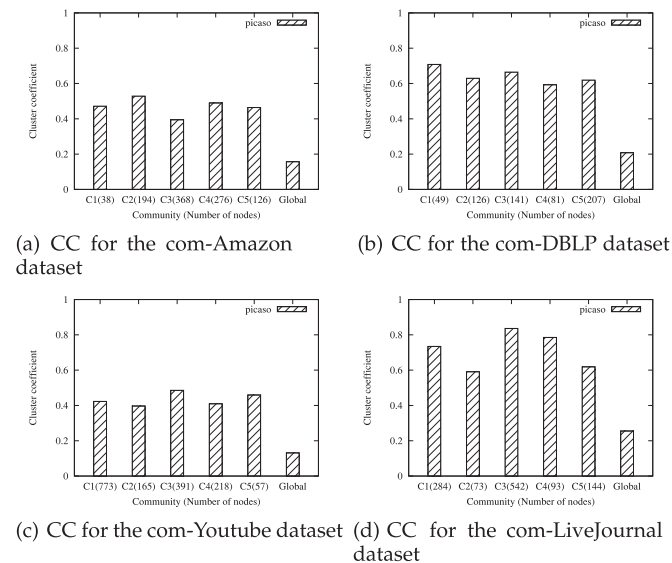


Fig. 8. Clustering coefficient of real large-scale complex networks.

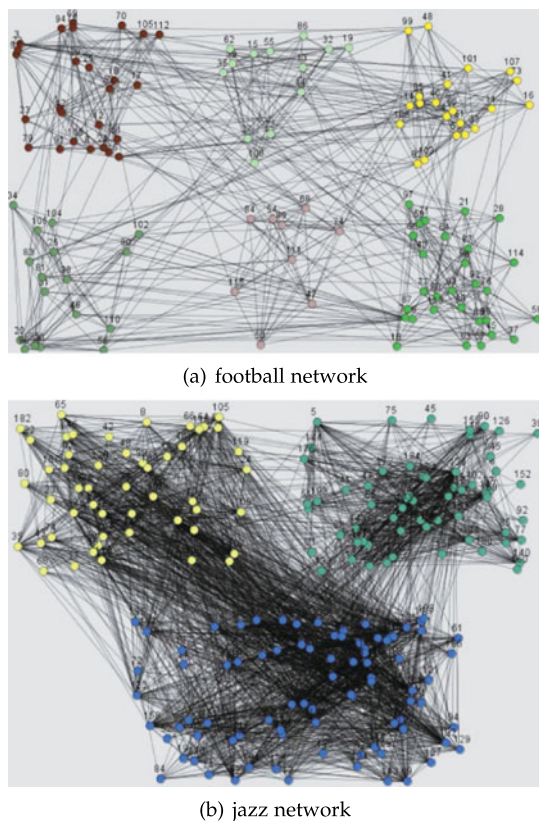


Fig. 9. Community structure of real network datasets.

in the merge operation. However, the Landslide algorithm provides an effective update strategy for calculating the modularity increment.

Fig. 9 visualizes the community structure of two small real networks by *picaso*. We can see that community structure of all these networks can be clearly identified, which strongly suggests the effectiveness of *picaso*.

## 5.5 Efficiency Analysis

*Picaso* is a community discovery algorithm which runs in parallel on the Spark platform, and is designed to handle

TABLE 4  
Execution Time of Real Small Network Datasets (sec.)

	strike	polbooks	football	jazz	facebook
OCDI	0.022	0.048	0.09	0.26	113.617
<i>picaso-<math>\alpha</math></i>	0.006	0.014	0.084	0.191	21.332
DBCS	1.847	2.674	3.87	6.249	61.823
PLMR	3.183	5.725	6.239	9.461	79.491
<i>picaso</i>	1.708	2.621	3.005	4.286	19.712

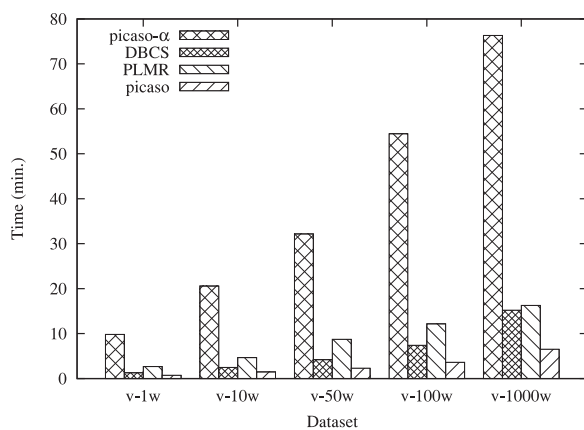


Fig. 10. Execution time on large-scale synthetic networks.

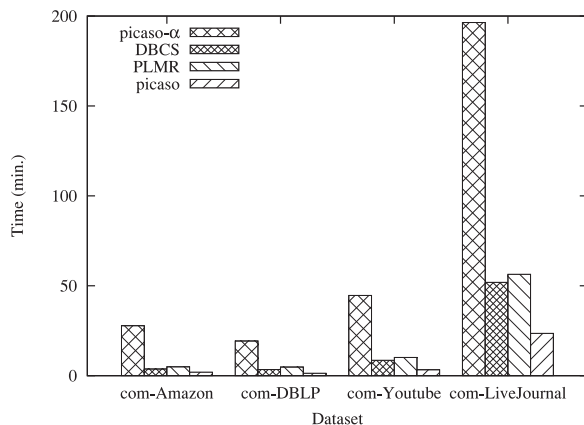


Fig. 11. Execution time on large-scale real networks.

large-scale complex networks. Given the size and scale of the networks, the runtime of the algorithm becomes almost as important as the accuracy. The following experiments were conducted on datasets of varying size and complexity. The results are shown in Table 4, Figs. 10 and 11.

Table 4 shows the case when the cardinality of the dataset is small, the results show that OCIDI and *picaso- $\alpha$*  are faster than *picaso* and DBCS. This is because, in terms of DBCS and *picaso*, the phases of task allocation and data transmission among Spark clusters occupies most of time for processing small-scale data. According to Table 4, the results show that the runtime for *picaso* is nearly equivalent to DBCS when the size of datasets are relatively small, this is because predominantly these two algorithms are mainly used in data transmission and file reading and writing. *Picaso*'s performance advantage becomes clear when the cardinality of data grows gradually.

When data from Facebook was used, *picaso* demonstrates a 3.14 and 4.03 times advantage in speed over DBCS

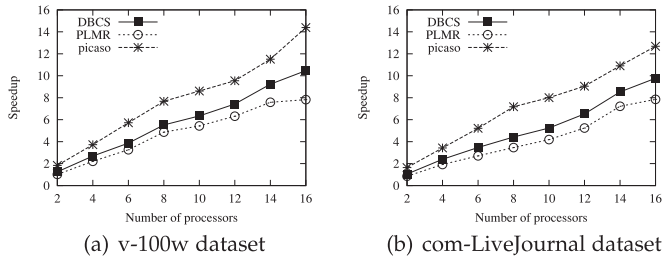


Fig. 12. Speedup comparison of parallel models on large-scale datasets.

and PLMR, respectively. This is because picaso chooses nodes with large modularity increments at the top of mountains and merges them periodically, which greatly reduces the calculation of community aggregation and data transmission. This method can make use of Spark clusters to prevent computation delay and waiting. In addition, picaso also uses the Landslide algorithm to calculate the modularity increment of communities, which also contributes to the reduction in computation time.

Findings in this research show that the execution time of picaso- $\alpha$  demonstrates an improvement of about 51.58 percent when compared to OCDI. Picaso- $\alpha$  also performs better than OCDI on smaller datasets, primarily because picaso- $\alpha$  aggregates a large number of nodes in each iteration so the total number of iterative calculations are reduced. This step helps to save both time and space.

As the number of nodes grows to more than 10 million, OCDI and picaso- $\alpha$  both do not work, thus the results obtain are only valid for runtime performance, which are shown in Figs. 10 and 11. We can see that the execution time of picaso is 1.8 and 2.3 times faster than that of DBCS for the synthetic and real networks, respectively, and picaso exceeds PLMR on runtime performance for 3.49 and 4.05 times on the synthetic and real datasets, respectively. In particular, the execution time of picaso is about 14.1 and 12.6 times faster than the serial picaso- $\alpha$  algorithm on the synthetic and real datasets, respectively. The reason for this advantage is that picaso chooses a large amount of nodes at the top of mountains to merge periodically, which greatly decreases the cost of community aggregation and data transmission, and can again make use of Spark clusters to reduce the calculation delay and waiting time by comparing with DBCS and PLMR algorithms. In addition, picaso employs the GraphX distribution graph computing framework to discover communities in a parallel manner which greatly improves the runtime performance when compared to picaso- $\alpha$  on a single processor.

We can see that the execution time of picaso shows an improvement of 59.8 and 54.0 percent on the v-1000w dataset when compared to the PLMR and DBCS algorithms, respectively. This proves that picaso can achieve good efficiency on handling very large complex networks with billions of edges.

Notice that, as shown in Fig. 11, the execution time of the compared algorithm PLMR is higher than 50 minutes, which is much slower when compared with the results of PLMR [24]. This is because PLMR does not work well when there are several overlapping nodes between communities. In our experiments, we specify  $\mu$  to 0.7 which is a large value implying there are several overlapping nodes and the community structures are hard to distinguish.

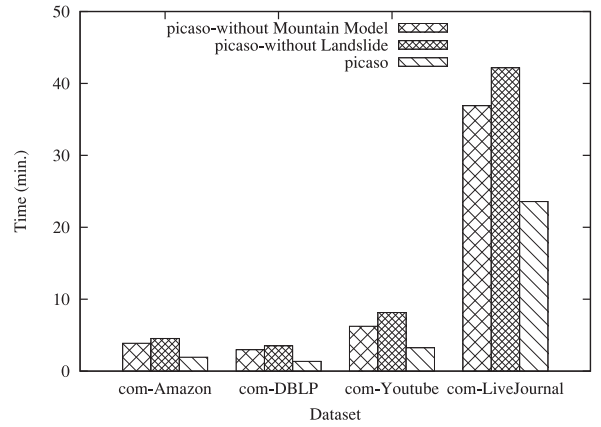


Fig. 13. Execution time comparison of different picaso algorithms.

In order to evaluate performance of parallel computing on multi-core processors, we observe the *speedup* factor of the DBCS, PLMR and picaso algorithms on the largest synthetic dataset (i.e., v-100w) and the largest real dataset (i.e., com-LiveJournal). The results are shown as follows.

According to Fig. 12, we can see that the speedup factor of picaso wins DBCS and PLMR under different number of processors. This is because picaso utilizes the chain-group structure to store the network data in GraphX on the Spark platform, which can accelerate the distribution computing and reduce the calculation delay.

## 5.6 Efficiency Estimation of Two Strategies in Picaso

As for picaso, the Mountain model is a heuristic strategy, which merges the nodes with  $\Delta Q$  larger than  $\Delta Q_\lambda$  to obtain an initial community result very quickly, and the Landslide update strategy is an approximate scheme to efficiently compute  $\Delta Q$ . In this section, we report how much each strategy can contribute to the runtime performance of picaso by applying them separately, and the results are given in Fig. 13, where picaso-without Mountain Model and picaso-without Landslide represent the picaso algorithm which only applies the Landslide update strategy and the Mountain Model, separately.

From Fig. 13, we can see that the Landslide update strategy contributes much to the efficiency on each real dataset. This is because the Landslide update strategy approximates the boundaries that divide the nodes into different communities, which can greatly reduce unnecessary computations for modularity increments.

## 6 CONCLUSION

In this research, we have presented a parallel community discovery algorithm for large-scale complex networks, named picaso. Picaso functions by integrating multiple innovations, which include the Mountain model, a new update strategy called the Landslide algorithm, which is based on approximate optimization techniques and graph theory. Picaso functions by finding the nodes that meet the condition of aggregation based on the Mountain model, then forms new communities and calculates the modularity increment between the newly formed communities and other communities. The experiments to test the validity of the proposed methods were conducted on synthetic and

real large-scale complex network datasets. The results demonstrate that *picaso* is more effective and efficient on detecting big communities in complex networks.

Future work will include addressing the case when the size of network nodes and edges become extremely large, e.g., more than 1 billion nodes. The proposed algorithm cannot guarantee real time performance in such a case, and will necessitate further innovations to produce efficiency computing of the modularity increment. Another challenge that will be addressed in future work is overlapping community recognition. This will require new methods for which will likely be implemented on the Spark platform.

In conclusion, the methods proposed in this research work to contribute to a larger effort targeted at advancing the study of complex community evolution. Understanding the evolution of network structures, analysing, processing and ultimately predicting the behavior of participants in large-scale social networks has and will continue to have a profound impact on society and technology.

## ACKNOWLEDGMENTS

This work is partially supported by the National Natural Science Foundation of China under Grant No. 61772091, 61100045, 61363037; the Planning Foundation for Humanities and Social Sciences of Ministry of Education of China under Grant No. 15YJAZH058; the Scientific Research Foundation for Advanced Talents of Chengdu University of Information Technology under Grant Nos. KYTZ201715, KYTZ201750; the Scientific Research Foundation for Young Academic Leaders of Chengdu University of Information Technology under Grant No. J201701 the Innovative Research Team Construction Plan in Universities of Sichuan Province under Grant No. 18TD0027.

## REFERENCES

- [1] A. Barabasi, R. Albert, H. Jeong, and G. Bianconi, "Power-law distribution of the world wide web," *Sci.*, vol. 287, no. 5461, 2000, Art. no. 2115.
- [2] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, no. 2, 2004, Art. no. 026113.
- [3] J. Lee, S. P. Gross, and J. Lee, "Improved network community structure improves function prediction," *Sci. Rep.*, vol. 3, no. 2, 2013, Art. no. 2197.
- [4] Wearesocial, "Gigital in 2016," 2016. [Online]. Available: <http://www.wearesocial.com>
- [5] C. Wickramaarachchi, M. Frincuy, P. Small, and V. K. Prasannay, "Fast parallel algorithm for unfolding of communities in large graphs," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2014, pp. 1–6.
- [6] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Phys. Rev. E*, vol. 69, 2004, Art. no. 066133.
- [7] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, no. 2, 2004, Art. no. 066111.
- [8] M. E. J. Newman, *Networks: An Introduction*. Oxford, U.K.: Oxford Univ. Press, 2010.
- [9] J. Qiu, J. Peng, and Y. Zhai, "Network community detection based on spectral clustering," in *Proc. Int. Conf. Mach. Learn. Cybern.*, 2014, pp. 648–652.
- [10] Y. Ruan, D. Fuhry, and S. Parthasarathy, "Efficient community detection in large networks using content and links," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 1089–1098.
- [11] Y. Wu, R. Jin, J. Li, and X. Zhang, "Robust local community detection: On free rider effect and its elimination," *Proc. VLDB Endowment*, vol. 8, no. 7, pp. 798–809, 2015.
- [12] X. Zhang, et al., "Overlapping community identification approach in online social networks," *Physica A*, vol. 421, pp. 233–248, 2015.
- [13] A. Prat-Pérez, D. Dominguez-Sal, J.-M. Brunat, and J.-L. Larriba-Pey, "Put three and three together: Triangle-driven community detection," *ACM Trans. Knowl. Discovery Data*, vol. 10, no. 3, 2016, Art. no. 22.
- [14] L. N. Ferreira and L. Zhao, "Time series clustering via community detection in networks," *Inf. Sci.*, vol. 326, pp. 227–242, 2016.
- [15] J. Shan, D. Shen, T. Nie, Y. Kou, and G. Yu, "Searching overlapping communities for group query," *World Wide Web*, vol. 19, no. 6, pp. 1179–1202, 2016.
- [16] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *Proc. VLDB Endowment*, vol. 9, no. 4, pp. 276–287, 2015.
- [17] X. Li, M. K. Ng, and Y. Ye, "MultiComm: Finding community structure in multi-dimensional networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 929–941, Apr. 2014.
- [18] A. Mahmood and M. Small, "Subspace based network community detection using sparse linear coding," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 3, pp. 801–812, Mar. 2016.
- [19] J. Whang, D. Gleich, and I. Dhillon, "Overlapping community detection using neighborhood-inflated seed expansion," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 5, pp. 1272–1284, May 2016.
- [20] T. N. Dinh, X. Li, and M. T. Thai, "Network clustering via maximizing modularity: Approximation algorithms and theoretical limits," in *Proc. IEEE Int. Conf. Data Mining*, 2015, pp. 101–110.
- [21] H. Shiokawa, Y. Fujiwara, and M. Onizuka, "Fast algorithm for modularity-based graph clustering," in *Proc. 27th AAAI Conf. Artif. Intell.*, 2013, pp. 1170–1176.
- [22] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey, "High quality, scalable and parallel community detection for large real graphs," in *Proc. 23rd Int. Conf. World Wide Web*, 2014, pp. 225–236.
- [23] A. Varamesh, M. K. Akbari, M. Fereiduni, S. Sharifian, and A. Bagheri, "Distributed clique percolation based community detection on social networks using MapReduce," in *Proc. 5th Conf. Inf. Knowl. Technol.*, 2013, pp. 478–483.
- [24] C. L. Staudt and H. Meyerhenke, "Engineering parallel algorithms for community detection in massive networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 171–184, Jan. 2016.
- [25] S. Moon, J. G. Lee, M. Kang, M. Choy, and J. W. Lee, "Parallel community detection on large graphs with MapReduce and graphchi," *Data Knowl. Eng.*, vol. 104, pp. 17–31, 2016.
- [26] Z. Lu, X. Sun, Y. Wen, G. Cao, and T. L. Porta, "Algorithms and applications for community detection in weighted networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 11, pp. 2916–2926, Nov. 2015.
- [27] S. Qiao, J. Guo, N. Han, X. Zhang, C. Yuan, and C. Tang, "Parallel algorithm for discovering communities in large-scale complex networks," *Chin. J. Comput.*, vol. 40, no. 3, pp. 687–700, 2017.
- [28] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel graph partitioning for complex networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2625–2638, Sep. 2017.
- [29] T. Takahashi, H. Shiokawa, and H. Kitagawa, "SCAN-XP: Parallel structural graph clustering algorithm on Intel Xeon Phi coprocessors," in *Proc. 2nd Int. Workshop Netw. Data Analytics*, 2017, Art. no. 6.
- [30] S. T. Mai, M. S. Dieu, I. Assent, J. Jacobsen, J. Kristensen, and M. Birk, "Scalable and interactive graph clustering algorithm on multicore CPUs," in *Proc. 33rd IEEE Int. Conf. Data Eng.*, 2017, pp. 349–360.
- [31] J. Shun, F. Roosta-Khorasani, K. Fountoulakis, and M. W. Mahoney, "Parallel local graph clustering," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1041–1052, 2016.
- [32] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph processing in a distributed data-flow framework," in *Proc. 11th USENIX Symp. Operating Syst. Des. Implementation*, 2014, pp. 599–613.
- [33] S. F. A. Lancichinetti, "Limits of modularity maximization in community detection," *Phys. Rev. E*, vol. 84, no. 6, 2011, Art. no. 066122.
- [34] J. Leskovec, "SNAP: Stanford large network dataset collection," 2016. [Online]. Available: <http://snap.stanford.edu/data/index.html>
- [35] M. E. J. Newman, "The structure and function of complex networks," *SIAM Rev.*, vol. 45, no. 2, pp. 247–256, 2003.



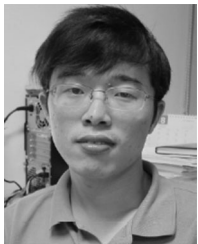
**Shaojie Qiao** received the BS and PhD degrees from Sichuan University, Chengdu, China, in 2004 and 2009, respectively. From 2007 to 2008, he worked as a visiting scholar in the School of Computing, National University of Singapore. He is currently a professor in the School of Cybersecurity, Chengdu University of Information Technology, Chengdu, China. He has led several research projects in the areas of databases and data mining. He authored more than 40 high quality papers, and coauthored more than 90 papers. His research interests include complex networks and trajectory data mining.



**Nan Han** received the MS and PhD degrees from Chengdu University of Traditional Chinese Medicine, Chengdu, China. She is a lecturer in the School of Management, Chengdu University of Information Technology, Chengdu, China. Her research interests include trajectory prediction and data mining. She is the author of more than 20 papers and she participated in several projects supported by the National Natural Science Foundation of China.



**Yunjun Gao** received the PhD degree in computer science from Zhejiang University, China, in 2008. He is currently a professor in the College of Computer Science and Technology, Zhejiang University, China. His research interests include spatial and spatio-temporal databases and spatio-textual data processing. He is a member of the ACM and the IEEE, and a senior member of the CCF.



**Rong-Hua Li** received the PhD degree from the Chinese University of Hong Kong, Hong Kong, in 2013. He is currently an associate professor in the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. His research interests include social network analysis and graph data management.



**Jianbin Huang** received the PhD degree in pattern recognition and intelligent systems from the Xidian University, in 2007. He is a professor in the School of Software, Xidian University of China. His research interests include data mining and knowledge discovery.



**Jun Guo** received the master's degree from the School of Information Science and Technology, Southwest Jiaotong University. His current research area include community discovery in complex networks.



**Louis Alberto Gutierrez** received the PhD degree in computer science from Rensselaer Polytechnic Institute, in 2014. He was a National Science Foundation GK-12 fellow, Mickey Leland Energy fellow and CHCI 2012 scholar. His research areas include social computing and mobile technologies.



**Xindong Wu** received the PhD degree in artificial intelligence from the University of Edinburgh, in 1993. He is a professor of computer science with the University of Louisiana at Lafayette, Lafayette. His research interests include data mining and knowledge-based systems. He is the editor-in-chief of the *Knowledge and Information Systems*, and the *Advanced Information and Knowledge Processing*. He is a fellow of the IEEE and the AAAS.

▷ For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).