Accepted Manuscript

PSPLPA: Probability and similarity based parallel label propagation algorithm on spark

Tinghuai Ma, Mingliang Yue, Jingjing Qu, Yuan Tian, Abdullah Al-Dhelaan, Mznah Al-Rodhaan

PII: S0378-4371(18)30236-X
DOI: https://doi.org/10.1016/j.physa.2018.02.130
Reference: PHYSA 19250

To appear in: *Physica A*

Received date : 16 October 2017
Revised date : 14 January 2018

Please cite this article as: T. Ma, M. Yue, J. Qu, Y. Tian, A. Al-Dhelaan, M. Al-Rodhaan, PSPLPA: Probability and similarity based parallel label propagation algorithm on spark, *Physica A* (2018), https://doi.org/10.1016/j.physa.2018.02.130

***Highlights (for review)**

The high lights are following :

1. propose a method to calculate the propagation probability and similarity between nodes with low complexity which utilized the improved K-shell decomposition,
2. proposed a new label updating strategy using the propagation probability and similarity between nodes,
3. propose a algorithm based on Spark and can handle the large-scale datasets efficiently.

Available online at www.sciencedirect.com

ScienceDirect

Physica A: Statistical Mechanics and its Applications 20 (2018) 1–14

PSPLPA: Probability and similarity based Parallel label propagation algorithm on Spark

Tinghuai Ma[a,b,*], Mingliang Yue[a], Jingjing Qu[a], Yuan Tian[d], Abdullah Al-Dhelaan[d], Mznah Al-Rodhaan[d]

[a] *School of Computer & Software, Nanjing University of information science & Technology, Jiangsu, Nanjing 210-044, China*
[b] *CICAEET, Jiangsu Engineering Center of Network Monitoring, Nanjing University of information science & Technology, Nanjing 210-044,China*
[c] *Huafeng Meteorological Media Group, Beijing 100080, China*
[d] *Computer Science Department, College of Computer and Information Sciences,King Saud University, Riyadh 11362, Saudi Arabia*

Abstract

With the rapid growth of social network, the cost of computation is increasing. Many existing algorithms are not suitable for the large-scale data. Apache Spark is an open-source cluster computing framework that empowers us to solve the problem of community detection in a cluster of computer. In this paper, we propose a novel label propagation algorithm on Spark, called PSPLPA (Probability and similarity based Parallel label propagation algorithm). PSPLPA employs a new label updating strategy using probability in the label propagation procedure during each iteration. First, weight calculation, which is based on k-shell, is integrated into the label initialization process. Second, parallel propagation steps are comprehensively proposed to utilize label probability efficiently. Third, randomness in label updating is significantly reduced via automatic label selection and similarity computation. Experiments conducted on artificial and real social networks demonstrate that the proposed algorithm exhibits high scalability and high accuracy.

Keywords: social network, community detection, label propagation, Spark, probability, similarity

1. Introduction

With the rapid development of Web 2.0 and the rise of online social networks, social networks have become an important tool for people to communicate with each other. Compared with traditional networks, social networks are more complex and have some special features. For example, the scale of social networks is enormous; the data of social networks are heterogeneous and from different data sources, etc. Since the most prominent property in a network is its community structure, the exploring the community structure of the large-scale social networks has become a hot aspect in social network analysis. There is no uniform definition of community structure, intuitively, a social network is typically modeled as a graph whose vertices and edges represent entities and relationships between entities respectively. Community structure in a network is always represented as a group of entities with dense connections within groups and sparse connections with other groups[1]. The community structure can be divided into two types, non-overlapping and overlapping. In real life, overlapping communities are more common than non-overlapping ones, such as a person is a student as well as a member of some clubs. Our algorithm is proposed to handle the discovery of overlapping communities.

*Corresponding author: thma@nuist.edu.cn

A great deal of community discovery algorithm have been proposed to mine reasonable community structure from complex social networks in the last decades. The Kernighan−Lin algorithm[2] discovers communities by minimizing the difference between intra-edges and inter-edges. Spectral methods determine the minimum cut to separate a graph and recursively divide a network into sections where communities emerge naturally[3]. The two methods mentioned above are two typical methods to discover community structure. Grivan and Newman first propose the concept of modularity, which was utilized to discover community structure[4]. After that, modularity is becoming a measure of community quality, based on the concept of modularity, a variety of methods are derived[1]. As the sharp increase in the amount of social network data, most of the existing community discovery algorithms are not applicable to handle such large-scale data. To uncover community structure in a near linear time, Raghavan et al. proposed a distinguished algorithm named label propagation algorithm (LPA) by employing a simple label propagation during each iteration[5]. LPA is practical and simple, due to the lower time consumption of LPA, it can be easily applied to large-scale data.

However, some drawbacks, such as the formation of a monster community, weak robustness and high randomness, remain in traditional LPA[5]. When community labels in a large community stop to propagate, labels in a small neighborhood nearby have not yet spread, and the label of the big community may affect the label of the small community as well, and the small communities will be devoured. After several such phenomena, a monster community may be formed. As a result, the quality of community discovery becomes very low. In addition, during the iterative process of the algorithm, nodes are updated randomly, so there may be some less influential nodes first updated and in turn affect the influential nodes, which leads to the phenomenon of countercurrent label. Furthermore, nodes randomly select tags when the neighbor nodes have the same labels. Thus, the result exhibits high randomness, which greatly affects the stability.

In order to solve the problem mentioned above, we propose a new LPA algorithm, at the same time, we must find a way to evaluate the significance of nodes in networks. Plenty of nodes ranking measures have been proposed such as local metrics, global metrics, and random-walk-based metrics[6]. Degree centrality is a typical type of local metric[7]. Local metrics are simple but ignoring the global structure. Global metrics contain betweenness centrality[8], closeness centrality[9], etc. Global metrics are suitable to uncover the important nodes but the computation cost is high. Random-walk-based metrics, composed by eigenvector centrality[10], PageRank[11], LeaderRank[12], etc, evaluate the significance of nodes using multiple iterative operations. All the methods above are not suitable to handle large-scale networks. K-shell decomposition algorithm is a fast node ranking method for large-scale networks, which is first put forward by Kitsak et al[6]. Traditional K-shell use the location information of nodes to calculate node influence. The most influential nodes are those located in the core of the network, which can be identified by the K-shell decomposition method. In our paper, we proposed a novel K-shell decomposition algorithm to rank nodes effectively.

Spark[13] is a fast and general open-source framework with the advantages of MapReduce for large-scale data processing, it stores the intermediate results in the memory. Spark use Resilient Distributed Dataset (RDD)[14] for sharing data in cluster applications. GraphX is a new component based on Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and aggregateMessages) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

This paper proposed a new label updating strategy to solve the above-mentioned problems. The contributions of this paper are summarized as follows:

- A method is proposed to calculate the propagation probability and similarity between each pair of nodes with a low computational complexity which utilized the iteration information in the improved K-shell decomposition.
- This paper proposed a new label updating strategy using the propagation probability and similarity between each pair of nodes, each node owns a label probability which stands for the probability of belonging to a community.
- The proposed algorithm is implemented on Spark and can handle the large-scale datasets efficiently.

The remainder of the paper is organized as follows. Section 2 mainly introduces the research background of community detection algorithms in networks. Section 3 provides a comprehensive discussion of the proposed parallel label propagation algorithm based on probability and similarity and its detailed implementation. The results of the

experiments conducted on artificial and real social networks are presented in Section 4. Section 5 concludes the study and presents an outline for future research.

2. Related Work

Community discovery has been a relatively hot topic in social network. So far, Internet data is getting bigger and bigger, traditional algorithms are not adjust to explore the community structure in large-scale data. Raghavan et al. proposed a famous algorithm called label propagation algorithm (LPA) by employing a simple label propagation during each iteration[5]. Over the past few years, several algorithms have been proposed to solve the limitations of the original LPA algorithm. A global parameter is used in COPRA[15] to allow each node to carry multiple labels, i.e. each node could belong to multiple communities, so COPRA can find overlapping community structure. To relieve the problem of monster communities, SLPA[16], and BMLPA[17] algorithms introduce an extra parameter to limit the number of labels that a node can hold. In order to deal with extremely large-scale networks, SLPA has been further extended to the MPI model to run parallel on a multi-core computer[18]. The PSCAN algorithm[19], which was designed on the Hadoop framework, provided another parallel scheme for the MapReduce model. The LPA−CNP algorithm[20] presents a new label updating strategy by considering the effect of neighbors that are more than one hop away. The CK−LPA algorithm[21] provides another label updating strategy that weighs the label of a vertex according to whether the node is in the community kernel. The LPA−S algorithm[22] suggests that a synchronous label propagation strategy is more effective than the default asynchronous strategy. R Li, et al. [23] proposed a parallel multi-label propagation algorithm to detect the overlapping communities in networks. Zhang Qishan, et al.[24] proposed a new parallel schema using grey relational analysis and achieve good performance on large-scale networks. Overlapping communities can also be found based on structural clustering, Tinghuai Ma, et al.[25] proposed an efficient overlapping community discovery algorithm named LED which shows the good performance in accuracy and efficiency. Recently, many researchers have attempted to find community structure in different ways such as subgraph method, density-based clustering, etc[26, 27]. Recently, N Chen, et al. [28] introduced the information entropy as the measurement of the relationship between nodes including direct and indirect neighbors and proposed a new belonging coefficient to describe the weight of the label. H Sun, et al. [29] presented an algorithm, called link-based label propagation algorithm (LinkLPA) which employs a new label propagation algorithm with preference on links instead of nodes to detect overlapping communities.

Kitsak et al.[6] put forward a fast node ranking method called k-shell decomposition for large-scale networks. The computational complexity of the k-shell method is O(n), where n represents the node number of the network. They argued that the node influence should be determined by the location of the node in the network. Influential nodes are those located in the core of the network. By decomposing a network with the k-shell decomposition method, the most influential nodes could be identified by the largest k-shell value. Sen et al.[30] evaluated several widely used centralities with a large scale online social community-LiveJournal, and claimed that the k-shell measure is more effective in finding nodes with a large influence. However, the traditional k-shell method assigns too many nodes with the same k-shell value. Later, researchers [31, 32, 33, 34, 35] proposed improved methods to overcome this shortcoming.All existing k-shell-based methods do not make use of the iteration information produced in k-shell decomposition. In fact, the iteration information is very important and meaningful for node influence capability evaluation.

Both of LPA and K-Shell are simple but efficient, the main idea of the two algorithms are utilized to guarantee the effectiveness and stability of our algorithm. Based on the traditional K-Shell method, we take advantage of the iteration information, the neighborhood attribute and the location attribute of each node to calculate the weight more precisely. After that, propagation probability and similarity between nodes are figured out according to the weight calculated before. Different from the LPA, our proposed method utilized the propagation probability between nodes and probability of nodes' labels to spread information, rather than randomly select labels which exsit most from neighbours. All steps of our algorithm are implemented by GraphX which is a module of Spark, the parallelized algorithm shows great performance on different kinds of dataset.

3. Parallel label propagation algorithm based on probability and similarity

3.1. Label propagation algorithm based on probability and similarity

In traditional LPA, each node updates its label according to the labels of its direct neighbors, label exists most among its neighbors will be chosen as the new label of a node. When the number of labels with maximum is more than one, the algorithm randomly selects one of the labels to assign to the node. Under the circumstance, nodes are treated equally. However, different neighbors may have different influences on a node, so it is reasonable to assign weight for every node. Accordingly, we select and update labels of a node in the following steps. First, the improved K-shell decomposition algorithm is used to measure the weight of nodes. Whereafter, propagation probability between each pair of nodes is calculated through the node weight, then, the similarity is figured up. Finally, labels with high probability are regarded as the new labels of a node. At the phase of label propagation and selection, every iteration can be regarded as a super step and label will be updated synchronously during each step. Traditional LPA algorithm can only find non-overlapping community because nodes choose only one label per iteration. In order to handle the problem of overlapping community discovery, nodes can carry multiple labels with probability.

The general procedures of our algorithm are illustrated in Fig. 1.

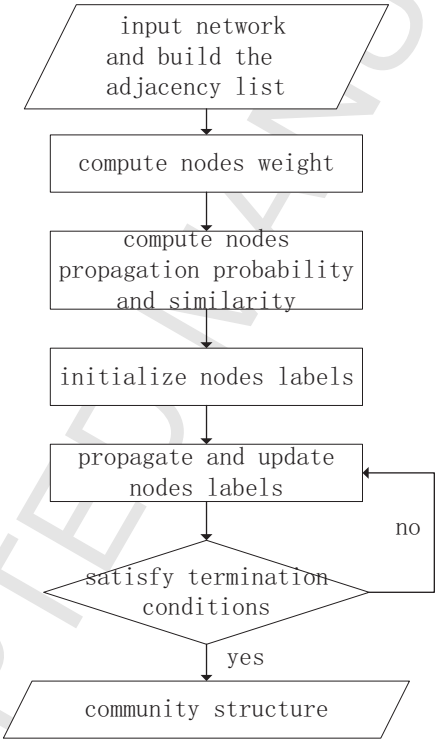

Figure 1. Procedures of our algorithm

The most influential nodes are those located in the core of the network, which can be identified by the k-shell decomposition method. However, we could not get more accurate result by simply using K-shell method.

As a typical type of local metric, degree centrality is simple but it ignores the information which nodes carry. For example, if we evaluate the significance only from the degree metric, all leaf nodes in the network will have the same importance. As a matter of fact, different leaf nodes should possess different influence. Meanwhile, location of nodes in the network also reflect the importance, i.e. influential nodes are those located in the core of the network. In this paper, neighborhood attribute and location attribute are integrated to calculate the neighborhood index, the position index, respectively. We combine the two indices to get the final weight of nodes.

Definition 1. Given a complex social network G(V,E), for any node $n_i \in V$, the influence capability of node n_i with respect to the neighborhood attribute, denoted as ICN(n_i), is given by

$$ICN(n_i) = \frac{d(n_i) + \sum_{n_j \in N(n_i)} d(n_j)}{max_{n_i \in V}(d(n_i) + \sum_{n_j \in N(n_i)} d(n_j))} \quad (1)$$

where, the degree of node n_i and n_j are d(n_i), d(n_j), respectively, N(n_i) represents the neighbor of n_i, ICN(n_i) denotes the neighborhood index and ICN(n_i)∈[0,1].

Take Fig. 2 as an example, node 1, 9, 14, 17 are leaf nodes with degree 1, node 1 has one neighbour node 6, so d(n_1)+d(n_6)=1+4=5, the total degree of node 10 and it's neighbours obtain the max value 23, so ICN(1)=5/23=0.22, ICN(9)=4/23=0.17, ICN(14)=7/23=0.30, ICN(17)=2/23=0.09. Thus, ICN can effectively distinguish the nodes with the same degree. Nodes with high value of ICN play an important role in the networks.

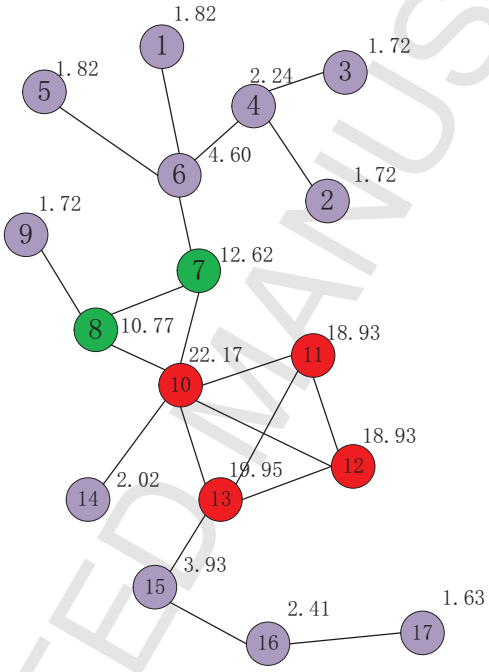

Figure 2. A schematic network

Definition 2. Given a complex social network G(V,E), each node is assigned with a K_s value by the k-shell decomposition. Assume node $n_i \in G$, and the K_s value of n_i is k. For the iterative process of the k-degree iteration, the total iteration number is m, and n_i is removed in the n^{th} iteration of the k-degree process, $1 \leq n \leq m$. $W(n_i)$ denotes the weight of node n_i, which is defined as follows:

$$W(n_i) = k * e^{ICN(n_i)+n/m} \quad (2)$$

In Equation (2), n is the iteartion order in each decomposition for a K_s, n/m denotes the position index. Table 1 shows the iteration process of k-shell decomposition. It is remarkable that the degree is a local metric, while the k-shell iteration order is a global metric, ICN(n_i)∈(0,1] and $n/m \in (0,1]$. In order to distinguish nodes more accurately, we combine the neighborhood index and position index to calculate the weight of nodes. The weight of nodes can be efficiently obtained with the linear time complexity O(n).Therefore, it is reasonable to believe that the new method is more efficient than other node ranking methods. For every K_s, ICN(n_i)+$n/m \in (0,2]$, so the weight vary from K_s to K_s*e^2.

Definition 3. Nodes in social networks can not only receive but also spread labels. Generally we take it that the more powerful nodes have the stronger ability, i.e., the higher probability to propagate labels, so we use the probability

Table 1. The iteration process of k-shell decomposition

Node degree	Iteration order	Deleted nodes	The value of Ks
1	1	node 1, node 2, node 3, node 5, node 9, node 14, node 17	1
1	2	node 4, node 16	1
1	3	node 6, node 15	1
2	1	node 7, node 8	2
3	1	node 10, node 11, node 12, node 13	3

to represent the ability of label propagation. Given a network G, $n_i \in G$, the probability between each pair of nodes is defined as follows:

$$P_{ij} = \frac{\log(1 + W_i)}{\log((1 + W_i) * (1 + W_j))} \tag{3}$$

$$P_{ji} = \frac{\log(1 + W_j)}{\log((1 + W_i) * (1 + W_j))} \tag{4}$$

where, W_i, W_j denotes the weight of n_i, n_j, respectively. P_{ij} represents the probability which n_i spreads its labels to n_j, as well as P_{ji} refer to the probability which n_i receive labels from n_j. After the calculation, each node have a probability list like $PList_i$={(j,P_{ji}), (k,P_{ki}), ...} where j, k, ... are neighbors of n_i.

Definition 4. Community structure in a network is always represented as a group of entities with dense connections within groups and sparse connections with the other groups, those node in the same community should have high similarity. In our paper, we proposed a new measure to calculate the similarity between nodes in order to group the similar nodes to the same community, the formulation is defined as follows:

$$Sim_{i,j} = W_j/(W_i + W_j) + P_{ji} \tag{5}$$

The metric $Sim_{i,j}$ is proposed to consider the importance of node itself and the probability metrix. The higher the P_{ji}, the higher the probability n_j propagate labels to n_i is, which means n_i and n_j are more similar.

Fig. 2 is a schematic network. First, we use K-shell decomposition method to calculate the K_s value and the iteration information of nodes. Table 1 shows the assigned K_s value of each node and the iteration information. Subsequently, each node sends its degree information to its neighbors, then, each node receives and computes the neighborhood index using Equation (1). Finally, node weight is figured out combining the neighborhood index and location index, and the value at the top right of the node represent the weight in Fig. 2.

Definition 5. At the phase of label propagation and selection, every iteration can be regarded as a super step and label will be updated synchronously during each step. In a super step, label list is sent to each of their neighbors, for each node, when receives the label list from all of neighbors, node labels are selected according to equation bellows:

$$L(n_i) = \beta * \sum_{j \in N(i)} P_{ji} * L(n_j) + (1 - \beta) * L(n_k) \quad s.t. \max_{n_k \in N(n_i)} Sim_{i,k} \tag{6}$$

where P_{ji} is the probability which n_i receives labels from n_j, L refers to the label list that a node carry, n_k represents the most similar node with n_i, and β is a tunable parameter.

Fig. 3. shows an example of the selection procedure. Suppose that the central node has three neighbors. The label list of node 1, 2, 3 are L(1) = {(a,0.5),(b,0.5)}, L(2) = {(b,0.4),(c,0.4),(d,0.2)}, L(3) = {(a,0.4),(d,0.6)}, respectively. P_{1x}=0.4, P_{2x}=0.6 and P_{3x}=0.8. The blue node is the most similar node with node x within its neighbors. We use Equation (6) to calculate the label list L(x)={(a,0.42),(b,0.22),(c,0.06),(d,0.56)}. After finding the maximum probability 0.56, each probability is divided by the maximum value to obtain a ratio, the labels with ratio higher than threshold ρ are retained. In [17], ρ=0.75, so L(x)={(a,0.42),(d,0.56)}. Finally, normalizing the label list and obtain the final label list L(x)={(a,0.43),(d,0.57)}. For each label list L, the following step are shown as follows:

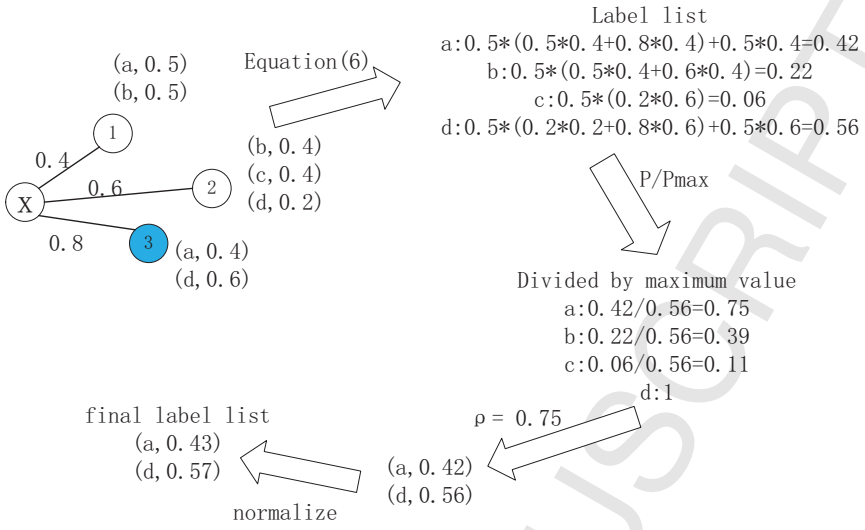

Figure 3. An example of the label selection procedure

(1) The labels in L are sorted according to the probability in a descending order.

(2) The labels should be retained is judged by $P/P_{max} \geq \rho$, where P is the probability of label and P_{max} refers to the max probability in a label list, ρ is the threshold parameter, and $\rho \in (0,1]$.

(3) The retained labels are normalized and make the sum of probability to 1.

3.2. Parallel label propagation based on probability and similarity

In our paper, we use GraphX to implement the parallelization. The parallelization of PSLPA(label propagation based on probability and similarity) consist five procedures: graph construction and neighbor collection, node weighting, probability and similarity calculation, node label initialization, and node label selection.

3.2.1. Parallel graph construction and neighbor collection

Social network is often too large to be saved on the memory of a single machine, and it is typically stored in a distributed file system, such as the HDFS (Hadoop Distributed File System) or HBase. Based on the parallel computation framework Spark, GraphX can read the network from the distributed file system and use the given operator to build an adjacency list. There are plenty of existing operators in GraphX, for example, *edgeListFile* operator provides a way to load a graph from a list of edges such as a txt format file in which each line contains two node IDs; *collectNeighbors* and *collectNeighborIds* operators are easier to express computation by collecting neighboring nodes and their attributes at each nodes; Once a graph is constructed, each node has a default value 1 as its property. In original graph, the tuple (a,1) means the node ID and its property.We can use some operators to change the default property of nodes, for example, *collectNeighborIds* operator is used to change the default value to NID which represents the neighbors ID of a node, all operations are depicted in Fig. 4.

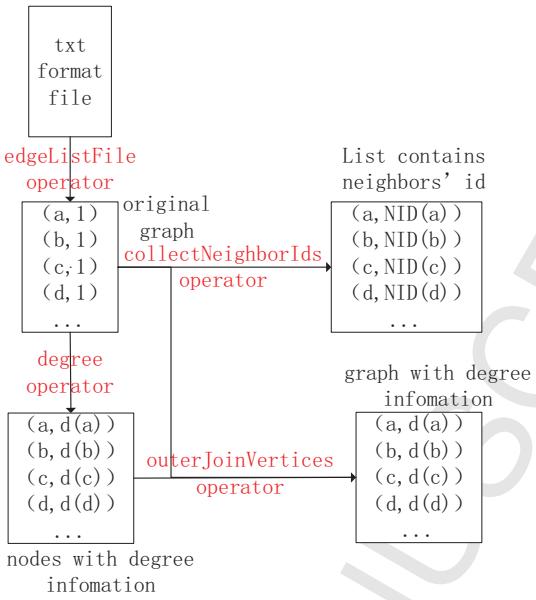

Figure 4. Details of graph construction and neighbors collection

3.2.2. *Parallel nodes weighting with respect to neighborhood index and position index*

In our method, parallelization of K-shell decomposition algorithm is implemented by *subgraph* operator in GraphX. After decomposition, each node owns a value of K_s and its iteration order during the iteration. We use *aggregateMessages* operator to send neighbors′ degree information, each node receives and calculates neighborhood index according to Equation (1). Combining the neighborhood index and position index, the weight of nodes are figured out using Equation (2).

3.2.3. *Parallel probability and similarity calculation*

Like the calculation for nodes weight, we also use *aggregateMessages* operator to aggregate messages from neighbors of nodes. Each node has a weight and send it to its neighbors. After that, each node collects weight from its neighbors, then, obtains the propagation probability and similarity using Equation (6).

3.2.4. *Parallel node label initialization*

Once the propagation probability and similarity between each pair of nodes are obtained, we use *mapVertices* operator to initialize the graph. Node i is initialized with a tuple $T(n_i)=(label_i,P)$, where $label_i$ represents the label, unlike the propagation probability, P refers to the probability that node i own. At first, each node is a single community, so $T(n_i)=(label_i,1.0)$ when the node is initialized.

3.2.5. *Parallel label propagation and selection*

The label list of a node can be passed to its neighbors by utilizing *Pregel* operator. In *Pregel* operator, we need to implement three custom functions. *sendMessage* function and *mergeMessage* function are used to send message and merge message, *vertexProgram* function is implemented for nodes selection. All the algorithms are iterated 1000 times to output the final result.

By combing all the parallel steps together, a parallel label propagation based on probability and similarity is developed by combining all the parallel steps in Algorithm 1.

3.3. *The efficiency of PSPLPA*

In this section, we conclude some key points of our algorithm. Different neighbors may have different influence on a node, so it is reasonable to assign weight for every node. In the preprocessing stage, we use traditional K-shell decomposition combined with neighborhood index and position index to obtain the final weight of nodes, we can

Algorithm 1 PSPLPA

Input: network G=(V,E),where V is the vertex set and E is the edge set
Output: community set C_i;
// Parallel graph construction and neighbor collection
graph : GraphLoader.*edgeListFile*(sc, path of file)
graphWithDegrees : graph.*outerJoinVertices*(graph.degrees)
output:construct a graph in which node property is its degrees
collectNeighborIds : graph.*collectNeighborIds*(EdgeDirection.Either)
output:RDD contains neighbors ID for each node
// Parallel nodes weighting
k=1,iter=1
while still has nodes in subgraph **do**
    find the outside nodes list verticesByKShell whose degree are k
    while verticesByKShell is not empty **do**
        put the nodes and its iteration information in vertexInfo
        iter : iter+1
        compute subgraph except the nodes added in vertexInfo
        find the outside nodes list verticesByKShell whose degree are k
    end while
    k : k+1
end while
compute nodes weighting according Equation(2)
// Parallel probability and similarity calculation
for nodes in graph **do**
    send node weight to its neighbors
end for
for nodes in graph **do**
    receive node weight and calculate propagation probability and similarity
end for
//Parallel node label initialization
for each node i in graph **do**
    $L(n_i)=(label_i,1.0)$
end for
// Parallel label propagation and selection
while do not reach the maximum number of iterations **do**
    for each node in graph **do**
        send nodes label list to its neighbors
    end for
    for each node in graph **do**
        receive label list and compute the final label list according Equation(6)
    end for
end while
output the communities

easily get the neighborhood index using degree of each node, the position index can be calculated by the iterative information during K-Shell decomposition. Since the time cost of K-shell is nearly linear, so the weight of nodes will be soon obtained. Then calculate the propagation probability and similarity between each pair of nodes, it's convenient to figure out through formulation 3,4 and 5, so the time cost of preprocessing stage is fast. In the parallel computation stage, we use GraphX framework to implement our algorithm, GraphX exposes a set of fundamental operators as well as an optimized variant of the Pregel API and supply convenient tools to support iterative computation, the intermediate results are stored in the memory instead of disks. In the stage of label selection, suppose that node A want to update it's labels, A receives labels from it's neighbors, then calculates the most similar node among it's neighbors, finally, updates it's labels by formulation 6. We also set the threshold parameter to control the speed of convergence. Thats why our algorithm so efficient.

4. Experiments

Comprehensive experiments were conducted on real and artificial networks to evaluate the performance of the proposed algorithm. Real networks are impossible to correctly measure which community an entity object belongs to, it is often measured by using modularity; the artificial ones are data sets with specified attributes that are constructed from pre-set parameters. After community partitioning, the accuracy of community discovery can be evaluated by calculating the NMI of two sets of data which are the results of the partitioning and the pre-set. This section shows the details of the analysis of the experimental results.

All the experiments are conducted on a cluster of 3 virtual computers. Each computer is equipped with 2 cores 2.5 GHz processors, 4GB of memory, Spark version 1.5.1 and Hadoop 2.4.1.

The artificial networks are generated using the synthetic LFR benchmark networks[1], which developed by Lancichinetti and Fortunato. We also use several real social networks to evaluate the performance of our algorithm. Table 2 describes the property of the experimental datasets.

Table 2. Details of the experimental data sets.

Networks		Description
Artificial networks		n = 100k to 500k for small networks and 1 M to 5 M for large networks. mu=0.1 to 0.8, k= 20, maxk = 100, minc = 50, maxc = 100, on = 0.1n, om = 3
Real networks	Amazon	n=334863, m=925872
	DBLP	n=317080, m=1049866
	Youtube	n=1134890, m=2987642

4.1. Experiments on synthetic LFR benchmark networks

In LFR network generator, we can set the parameter *n*(the number of nodes), *m*(the number of edges), *mu*(the mixing parameters), *k*(the average degree of nodes), *maxk*(the maximum degree of nodes), *on*(the number of overlapping nodes), *om*(the number of memberships of the overlapping nodes), *minc*(the minimum community size) and *maxc*(the maximum community size) to generate the networks which meet our demand. In particular, *mu* stands for the complexity of a network. A network with a high value of *mu* means the community is difficult to be uncovered correctly. For the artificial networks, the structure of network is constructed according to the parameters we input, and we can obtain the correct community structure of the network. Generally, NMI is often used to evaluate the result of community discovery. We use the community structure obtained from the generator as the correct result, compare with the result obtained by the output of our algorithm. Finally, we use the two results to construct a matrix N and use the following formulation to compute the value of NMI.

$$NMI(X,Y) = \frac{-2\sum_{i=1}^{c_x}\sum_{i=1}^{c_y} N_{ij}\log(\frac{N_{ij}N}{N_iN_j})}{\sum_{i=1}^{c_y} N_j\log(\frac{N_j}{N}) + \sum_{i=1}^{c_x} N_i\log(\frac{N_i}{N})} \tag{7}$$

Where, N is the total number of nodes in the network, N_i and N_j denote the sum of i^{th} row and the sum of j^{th} cloumn in matrix N. c_x and c_y represent the number of communities in the real data and the number of communities

in which the algorithm runs. NMI embodies the similarity of two sets of data and its value range is between 0 and 1. The larger NMI value is, the closer the community partition will be to the real result. It is obvious that if all nodes are correctly partitioned, the value of NMI is 1, and if all nodes are not correctly partitioned, the value of NMI is 0.

The number of nodes in the generated synthetic networks varies from 100 k(one thousand) to 500 k for small-scale networks and 1 M(one million) to 5 M for large-scale networks. The degree of a node varies from 20∼100, the size of a community varies from 50∼100, and the mixing parameter ranges from 0.1∼0.8. We treat all directed edges as undirected edges to construct the corresponding undirected graphs. For each network, we use PCOPRA[15], PSLPA[16], PGLPA[24], PMLPA[23], LPA-E[28], LinkLPA[29] and PSPLPA to perform experiments and analyze the experimental results.

As shown in Fig. 5, the community partition accuracy of the PSPLPA algorithm is invariably better than the other algorithms except the LinkLPA algorithm on small-scale artificial networks, the accuracy of our method is almost the same with LinkLPA. Generally, our method shows better performance than other methods, the result demonstrates the accuracy when nodes select their labels according to the propagation probability and similarity and automatically drop the labels whose probability lower than the threshold. Fig. 5 also shows that PSPLPA performs well on large-scale artificial networks. The value of NMI is similar with that on small-scale networks. It indicated that PSPLPA has good expansibility and can adapt to the huge amounts of data.

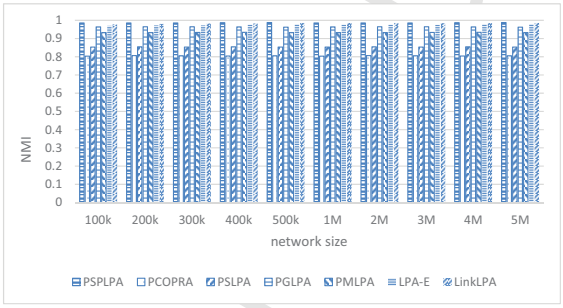

Figure 5. NMI on networks with small-scale and large-scale

Fig.6 shows the NMI values of five algorithms on artificial networks with different *mu*. When the mixing parameter *mu* is less than 0.5, PSPLPA, PMLPA, PGLPA, LPA-E and LinkLPA have higher NMI values. When *mu* is small, the community structure is very apparent, so the algorithm can often obtain relative better community partition result; When *mu* is large than 0.5, PSPLPA, PMLPA, PGLPA, LPA-E and LinkLPA begin to decrease in the value of NMI; When *mu*=0.7, PCOPRA, PMLPA and PSLPA cannot figure out the correct community partition. Compare PSPLPA with PGLPA, we can find when *mu* varies from 0.6 to 0.7, PGLPA declines sharply in the NMI value while PSPLPA can obtain relative good results. When *mu* vary from 0.7 to 0.8, only our method can find community structure well. It indicated that PSPLPA can handle more complex networks than the other four algorithms.

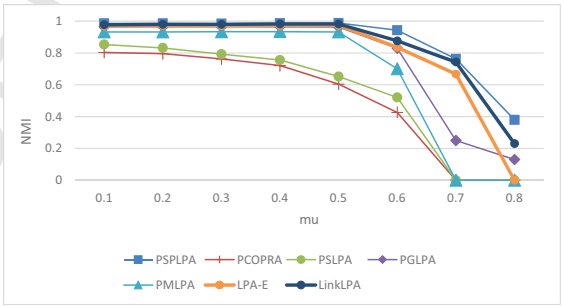

Figure 6. NMI on networks with different mu, n=1M

4.2. Experiments on real networks

There are plenty of real networks to evaluate the performance of the algorithm. Table 2. shows the details of the networks we selected. In this section, we use these five real networks to perform experiments and measure the performance of community discovery.

In real networks, it is hard to obtain the true communities. Generally, we use modularity Q[36] to evaluate the quality of the communities which the algorithm found. The modularity Q is defined as follows:

$$Q = \sum_{c=1}^{n_c} [\frac{l_c}{m} - (\frac{d_c}{2m})^2] \tag{8}$$

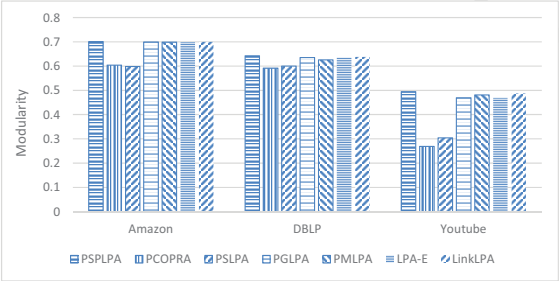

Figure 7. Modularity on real networks with large-scale

where m, n_c, l_c, and d_c are the edge number, the number of communities, the number of internal edges in community c, and the sum of degrees in community c, respectively.

Generally, communities in the real networks are more hard to uncover than those in the artificial networks. Fig. 7 shows the result of these seven algorithm on real network Amazon, DBLP and Youtube. Our algorithm obtain the same value of modularity with LPA-E and LinkLPA on the three datasets, and the same value of modularity with PGLPA on Amazon, it achieves higher modularity than the other four algorithms on DBLP and Youtube. The result shows that our algorithm performs well not only on artificial networks but also on real networks.

As mentioned before, β is a tunable parameter in the step of label selection duration. Fig. 8 shows that when β ranges from 0.4 to 0.6, all algorithms can obtain a good modularity on real networks. In our algorithm, β is set to 0.5 to get a better solution. Our method can find overlapping communities by allowing each node to own a label list. Finally, each node will carry a label list $L(n_i)=\{(label_1,P_1), (label_2, P_2), \dots\}$.

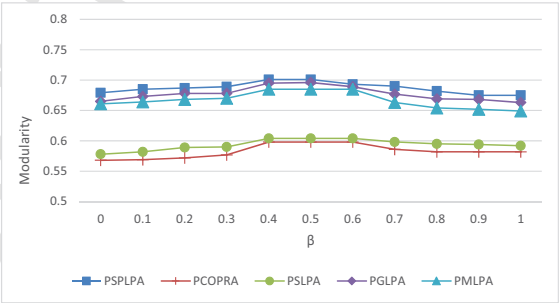

Figure 8. Modularity on real networks with different β

Fig.9 shows that PSPLPA has a good speedup performance. Once a graph build from the dataset, all operations are executed in memory. What's more, nodes use the probability of labels to update their labels, rather than randomly select labels which exist most from neighbours. That's why our algorithm is fast and stable. As the number of CPU cores increases, the running time decreases. What's more, the parallel framework would spent most of the running time

to exchange information among machines on small-scala dataset. Consequently, The proposed algorithm obviously has better performance on large-scale networks.

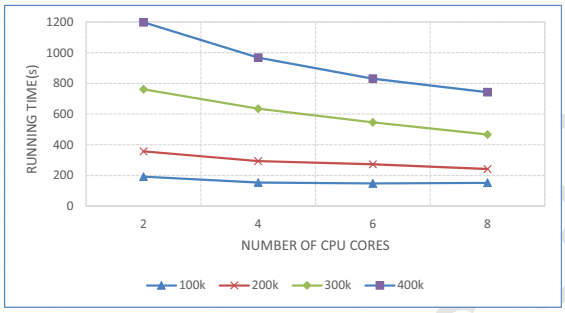

Figure 9. Running time on different networks and different cores of PSPLPA

5. Conclusion

LPA and K-shell decomposition algorithm are two efficient algorithm with simple concept. In this study, we combine the advantages of the two algorithm and implement the parallelization of the procedure of our algorithm. Propagation probability and similarity are integrated into the iteration of label propagation and selection. All the procedures run in parallel, and multiple label with probability are allowed to be assigned to a node. Hence the algorithm can discover overlapping communities. The experiments on large-scale networks shows the stability and extension of our proposed method. In this section, we conclude some key points of our algorithm. Different neighbors may have different influence on a node, so it is reasonable to assign weight for every node. In the preprocessing stage, we use traditional K-shell decomposition combined with neighborhood index and position index to obtain the final weight of nodes, we can easily get the neighborhood index using degree of each node, and the position index can be calculated by the iterative information during K-Shell decomposition. Since the time cost of K-shell is nearly linear, so the weight of nodes will be soon obtained. Then calculate the propagation probability and similarity between each pair of nodes, it's convenient to figure out through formulation 3,4 and 5, so the time cost of preprocessing stage is fast. In the parallel computation stage, we use GraphX framework to implement our algorithm, GraphX exposes a set of fundamental operators as well as an optimized variant of the Pregel API and supply convenient tools to support iterative computation, the intermediate results are stored in the memory instead of disks. In the stage of label selection, suppose that node A want to update it's labels, A receives labels from it's neighbors, then calculates the most similar node among it's neighbors, finally, updates it's labels by formulation 6. We also set the threshold parameter to control the speed of convergence. That is why our algorithm so efficient.

Several issues remain for further research. First, the calculation of similarity between nodes can be enhanced to get more stable results. Second, the algorithm can be improved to be practical by considering the dynamic network. Third, with the rapid emergence of new parallel computation frameworks aside from Hadoop and Spark, efficient parallel operators may be utilized to improve the performance of the algorithm.

6. Acknowledgements

This work was supported in part by National Science Foundation of China (No. 6157060479, No. U1736105), Special Public Sector Research Program of China (No. GYHY201506080).

The authors extend their appreciation to the Deanship of Scientific Research at King Saud University for funding this work through research group no. RGP-VPP-264.

References

[1] A. Lancichinetti, S. Fortunato, Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities, Physical Review E 80 (1) (2009) 016118.

[2] B. W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, The Bell system technical journal 49 (2) (1970) 291–307.
[3] P. Symeonidis, N. Iakovidou, N. Mantas, Y. Manolopoulos, From biological to social networks: Link prediction based on multi-way spectral clustering, Data & Knowledge Engineering 87 (2013) 226–242.
[4] M. Girvan, M. E. Newman, Community structure in social and biological networks, Proceedings of the national academy of sciences 99 (12) (2002) 7821–7826.
[5] U. N. Raghavan, R. Albert, S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, Physical review E 76 (3) (2007) 036106.
[6] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, H. A. Makse, Identification of influential spreaders in complex networks, arXiv preprint arXiv:1001.5285.
[7] S. Gao, J. Ma, Z. Chen, G. Wang, C. Xing, Ranking the spreading ability of nodes in complex networks based on local structure, Physica A: Statistical Mechanics and its Applications 403 (2014) 130–147.
[8] L. C. Freeman, Centrality in social networks conceptual clarification, Social networks 1 (3) (1978) 215–239.
[9] G. Sabidussi, The centrality index of a graph, Psychometrika 31 (4) (1966) 581–603.
[10] S. P. Borgatti, Centrality and network flow, Social networks 27 (1) (2005) 55–71.
[11] S. Brin, L. Page, Reprint of: The anatomy of a large-scale hypertextual web search engine, Computer networks 56 (18) (2012) 3825–3833.
[12] L. Lu, Y.-C. Zhang, C. H. Yeung, T. Zhou, Leaders in social networks, the delicious case, PloS one 6 (6) (2011) e21202.
[13] A. Spark, Apache spark: Lightning-fast cluster computing (2016).
[14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, 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, USENIX Association, 2012, pp. 2–2.
[15] S. Gregory, Finding overlapping communities in networks by label propagation, New Journal of Physics 12 (10) (2010) 103018.
[16] J. Xie, B. K. Szymanski, Towards linear time overlapping community detection in social networks, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2012, pp. 25–36.
[17] Z.-H. Wu, Y.-F. Lin, S. Gregory, H.-Y. Wan, S.-F. Tian, Balanced multi-label propagation for overlapping community detection in social networks, Journal of Computer Science and Technology 27 (3) (2012) 468–479.
[18] K. Kuzmin, S. Y. Shah, B. K. Szymanski, Parallel overlapping community detection with slpa, in: Social Computing (SocialCom), 2013 International Conference on, IEEE, 2013, pp. 204–212.
[19] W. Zhao, V. Martha, X. Xu, Pscan: a parallel structural clustering algorithm for big networks in mapreduce, in: Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on, IEEE, 2013, pp. 862–869.
[20] H. Lou, S. Li, Y. Zhao, Detecting community structure using label propagation with weighted coherent neighborhood propinquity, Physica A: Statistical Mechanics and its Applications 392 (14) (2013) 3095–3105.
[21] Z. Lin, X. Zheng, N. Xin, D. Chen, Ck-lpa: Efficient community detection algorithm based on label propagation with community kernel, Physica A: Statistical Mechanics and its Applications 416 (2014) 386–399.
[22] S. Li, H. Lou, W. Jiang, J. Tang, Detecting community structure via synchronous label propagation, Neurocomputing 151 (2015) 1063–1075.
[23] R. Li, W. Guo, K. Guo, Q. Qiu, Parallel multi-label propagation for overlapping community detection in large-scale networks, in: International Workshop on Multi-disciplinary Trends in Artificial Intelligence, Springer, 2015, pp. 351–362.
[24] Q. Zhang, Q. Qiu, W. Guo, K. Guo, N. Xiong, A social community detection algorithm based on parallel grey label propagation, Computer Networks 107 (2016) 133–143.
[25] T. Ma, Y. Wang, M. Tang, J. Cao, Y. Tian, A. Al-Dhelaan, M. Al-Rodhaan, Led: A fast overlapping communities detection algorithm based on structural clustering, Neurocomputing 207 (2016) 488–500.
[26] Y. Lv, T. Ma, M. Tang, J. Cao, Y. Tian, A. Al-Dhelaan, M. Al-Rodhaan, An efficient and scalable density-based clustering algorithm for datasets with complex structures, Neurocomputing 171 (2015) 9–22.
[27] H. Rong, T. Ma, M. Tang, J. Cao, A novel subgraph k^+ -isomorphism method in social network based on graph similarity detection, Soft Computing (2017) 1–19.
[28] N. Chen, Y. Liu, H. Chen, J. Cheng, Detecting communities in social networks using label propagation with information entropy, Physica A Statistical Mechanics & Its Applications 471 (2017) 788–798.
[29] H. Sun, J. Liu, J. Huang, G. Wang, X. Jia, Q. Song, Linklpa: A linkbased label propagation algorithm for overlapping community detection in networks, Computational Intelligence 33 (2) (2017) 308–331.
[30] S. Pei, H. A. Makse, Spreading dynamics in complex networks, Journal of Statistical Mechanics: Theory and Experiment 2013 (12) (2013) P12002.
[31] A. Zeng, C.-J. Zhang, Ranking spreaders by decomposing complex networks, Physics Letters A 377 (14) (2013) 1031–1035.
[32] J.-G. Liu, Z.-M. Ren, Q. Guo, Ranking the spreading influence in complex networks, Physica A: Statistical Mechanics and its Applications 392 (18) (2013) 4154–4159.
[33] Z.-M. Ren, J.-G. Liu, F. Shao, Z.-L. Hu, Q. Guo, Analysis of the spreading influence of the nodes with minimum k-shell value in complex networks.
[34] B. Hou, Y. Yao, D. Liao, Identifying all-around nodes for spreading dynamics in complex networks, Physica A: Statistical Mechanics and its Applications 391 (15) (2012) 4012–4017.
[35] J. Bae, S. Kim, Identifying and ranking influential spreaders in complex networks by neighborhood coreness, Physica A: Statistical Mechanics and its Applications 395 (2014) 549–559.
[36] M. E. Newman, M. Girvan, Finding and evaluating community structure in networks, Physical review E 69 (2) (2004) 026113.