International Journal on Artificial Intelligence Tools
Vol. 26, No. 3 (2017) 1760013 (18 pages)

DOI: 10.1142/S0218213017600132

Parallel Multi-Label Propagation Based on Influence Model and Its Application to Overlapping Community Discovery

Qirong Qiu

School of Economics & Management, Fuzhou University
Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing
Fuzhou 350116, China
qqrkyc@fzu.edu.cn

Wenzhong Guo*, Yuzhong Chen† and Kun Guo‡

College of Mathematics and Computer Science, Fuzhou University
Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing
Key Laboratory of Spatial Data Mining & Information Sharing, Ministry of Education
Fuzhou 350116, China
**fzugwz@163.com*
†*yzchen1979@163.com*
‡*gukn123@163.com*

Rongrong Li

College of Mathematics and Computer Science, Fuzhou University
Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing
Fuzhou 350116, China
ditaps@163.com

Received 28 March 2016
Accepted 9 March 2017
Published 23 June 2017

Finding communities in networks is one of the challenging issues in complex network research. We have to deal with very large networks that contain billions of vertices, which makes community discovery a computationally intensive work. Moreover, communities usually overlap each other, which greatly increases the difficulty of identifying the boundaries of communities. In this paper, we propose a parallel multi-label propagation algorithm (PMLPA) that enhances traditional multi-label propagation algorithm (MLPA) in two ways. First, the critical steps of MLPA are parallelized based on the MapReduce model to get higher scalability. Second, new label updating strategy is used to automatically determine the most valuable labels of each vertex. Furthermore, we study the improvement of PMLPA through considering the influence of vertices and labels on label updating.

‡Corresponding author

In this way, the importance of each label can be described with higher precision. Experiments on artificial and real networks prove that the proposed algorithms can achieve both high discovering accuracy and high scalability.

Keywords: Community discovery; multi-label propagation; influence model; overlapping community.

1. Introduction

Discovering communities in networks can be regarded as a kind of clustering problem that aims at finding groups of friends from social networks or protein structures from biological networks.[1] A community is a group of vertices that are more densely connected to each other than to the rest of the network. More formally, let C be a subgraph of a graph G that represents a network. We define the internal degree k_{in}^{C} of C as the sum of the internal degrees of its vertices and the external degree k_{out}^{C} of C as the sum of the external degrees of its vertices. The internal and external degree of a vertex is the number of edges connecting it to other vertices of C or to the rest of the graph, respectively. From k_{in}^{C} and k_{out}^{C}, we define the intra-cluster density ρ_{in}^{C} and inter-cluster density ρ_{out}^{C} of cluster C as follows:

$$\rho_{in}^{C} = \frac{k_{in}^{C}}{n_c(n_c - 1)}$$
$$\rho_{out}^{C} = \frac{k_{out}^{C}}{n_c(n_c - 1)/2} \tag{1}$$

where n_c is the number of vertices of C. For C to be a community, we expect ρ_{in}^{C} to be appreciably larger than ρ_{out}^{C}.

In the real world, communities may be difficult to recognize because they always overlap with each other. For example, some of a person's colleagues may also be his/her schoolmates. Therefore, discovering overlapping communities in networks has become a hot research topic.

Community discovery algorithms can be roughly divided into the algorithms based on graph partitioning,[1–3] the algorithms based on modularity optimization,[4–6] the algorithms based on label propagation,[7–9] etc.

Girvan and Newman first proposed a community detection algorithm named GN.[1] GN is an algorithm based on graph partitioning. It considers more about the edges connecting communities with high betweenness than those edges inside communities. Since its time complexity is $O(n_3)$ on sparse graphs, the algorithm is impractical to process large networks. However, the GN algorithm inspired many researchers to design more efficient community discovery algorithms. Later, researchers developed the algorithms[2,3] that can greatly reduce time cost. Nevertheless, some algorithms achieve speedup at the expense of losing discovering accuracy.

The algorithms based on modularity optimization aims at maximizing the modularity proposed by Newman *et al.*[4] Modularity is a widely applied index to measure the quality

of the community discovery algorithms. However, modularity has its own drawbacks like the resolution limit[10] and the extreme degeneracy.[11]

Label Propagation Algorithm (LPA) is one of the fastest community discovery algorithms that can process large networks.[7] Each vertex in LPA is initialized with a unique label. The label of each vertex is updated iteratively by selecting the most appropriate label of its neighbors. The principles of LPA are simple. It does not require setting the heuristic rules or a predefined object function. Neither does it need prior information about the communities. Therefore, LPA has soon attracted attentions of many researchers. Gregory first applied LPA to find overlapping communities and proposed Community Overlap PRopagation Algorithm (COPRA).[8] COPRA allows each vertex belong to at most v communities. However, it is not easy to set a suitable value of v for a particular network. Wu *et al.*[9] proposed a method named BMLPA to avoid restricting the number of communities a vertex can belong to. BMLPA uses a threshold parameter p to restrict the number of labels a vertex can have.

As network scale grows up, it becomes more and more difficult for traditional single-machine algorithms to discover communities in reasonable time. Parallel computation model like MapReduce[10] emerges as a good option to adapt the community discovery algorithms to large networks. Leung *et al.*[11] proved that LPA was suitable to be parallelized, as it did not need much information about network structures. Zhao *et al.*[13] proposed Parallel Structural Clustering Algorithm (PSCAN) for community detection in large networks. PSCAN implements parallel LPA and structural clustering based on the MapReduce model. However, PSCAN cannot find overlapping communities.

In our previous work,[14] we proposed a parallel MLPA for overlapping community detection in large social networks. In this paper, we extend this work and further study the influence of vertices and labels on label updating. A new influence model is proposed to describe the impact of vertices and labels on label updating. The major contributions of this study are:

(1) The MapReduce model is applied to the parallelization of traditional MLPA. Parallel operators provided by modern parallel computation frameworks are used to improve the ability to deal with large networks.
(2) New influence model that considers both the impact of neighboring vertices and the weights of vertex labels is proposed to assist the evaluation of labels during label updating.
(3) Comprehensive experiments are conducted to compare the proposed algorithms with the state-of-the-art algorithms on both artificial and real networks. The experimental results show the efficiency and effectiveness of our algorithms.

The remainder of the paper is organized as follows. Section 2 introduces the concepts of multi-label propagation. The proposed algorithms are discussed in detail in Section 3. Section 4 presents the experimental results on artificial and real networks. Section 5 concludes our work and poses directions for future research.

2. Multi-Label Propagation

Label Propagation Algorithm (LPA) was first proposed by Raghavan *et al.* in Ref. 7. The ideas behind LPA are as follows.

(1) Each vertex is initialized with a random label.
(2) Each vertex's label is updated according to its neighbors' labels. The labels selected by the most neighbors are usually preferred.
(3) Step (2) is run iteratively until the convergence condition is satisfied or the maximum iteration is reached.
(4) Finally, the vertices with the same label are collected into a community.

The advantages of LPA include that its run time is nearly linear to network size. Therefore, it is an algorithm that especially suitable for processing large networks. However, the LPA proposed in Ref. 7 uses asynchronous updating strategy that may cause instability of community detection.[15] Moreover, it cannot discover overlapping communities.

Multi-Label Propagation Algorithms (MLPA) were proposed to solve the problems of LPA, where COPRA proposed by Gregory is a well-known one. The improvements of COPRA include:

(1) Each vertex can retain more than one label in its label set.
(2) Each label of a vertex is connected with a belonging coefficient that describes the strength of the vertex to the label.
(3) A new parameter v is used to filter the labels whose belonging coefficients are smaller than it. In this way, many fragment communities are avoided.
(4) Synchronous updating strategy is used to achieve stable results.

With the above modifications, COPRA performs much better than traditional LPA. However, COPRA also has its weaknesses. For example, it is inclined to generate some huge communities that cover many real small communities.[9]

3. Parallel Multi-Label Propagation Based on the MapReduce Model

3.1. *Design ideas*

The key point of parallelizing the MLPA algorithm is that the resources required by any one of its step are separable from the resources required by the others. By inspecting the steps of MLPA thoroughly, we have the following property for the parallelization of MLPA.

Property 1. The critical steps of MLPA, i.e. adjacency list generation, vertex label initialization, label updating, are all parallelizable based on the MapReduce model.

Description: First, the network structure can be read parallelly when each line of the network file represents an edge. Second, since a vertex selects itself as its initial label, the step of vertex label initialization can be easily parallelized. Third, a vertex updates its

label set according to the opinions of its own neighbors. Therefore, each vertex can update its labels independently.

Detailed schemes for the parallelization of MLPA are listed in the following.

(1) Parallel adjacency list generation

A map/reduce job is used to read the network data and build the adjacency list. The input of the map task is (key = line offset, value = line string), which means that each line of the data file is read separately and parallelly. The output of the map task is (key = *i*, value = *j*) and (key = *j*, value = *i*) where *i* and *j* are neighbors to each other. The output of the map task is sent to the reduce task. The reduce task collects all the neighbors of each vertex and output (key = *i*, value =*NB*(*i*)) where *NB*(*i*) represents the neighbor set of *i*. After the map/reduce job is done, the adjacency list is built.

(2) Parallel vertex label initialization

A vertex's label is a tuple in the format of (*l*, *bc*) where *l* is the label and *bc* is the belonging coefficient that represents the strength of the vertex membership with community *l*. We use the rough core algorithm[9] to build a rough core set Set_{rc} and initialize each vertex's label set with the labels in Set_{rc}. The belonging coefficient of each label is initialized to 1.0. By broadcasting the Set_{rc} to each machine, the parallelization of vertex label initialization can finish the job in a single map task. The input of the map task is (key = *i*, value = *NB*(*i*)). The output of the map task is (key = *i*, value = *L*(*i*)) where *L*(*i*) represents the set of tuples (l, bc_l) and *l* is a label of vertex *i* selected from Set_{rc}.

(3) Parallel label updating

A vertex's labels are updated according to the weight of the belonging coefficients of its neighbors. Therefore, in a map/reduce job, we should join a vertex's neighbor set and its label set at first. Then, a map task is used to send the vertex's label set to its neighbors. Finally, a reduce task is used to calculate the new label set for each vertex. The detailed of the parallel label updating is illustrated in Fig. 1.

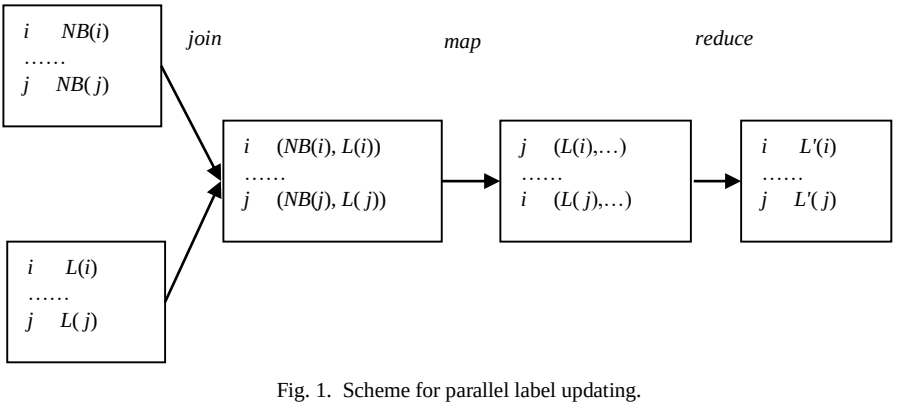

Fig. 1. Scheme for parallel label updating.

As shown in Fig.1, the parallel label updating job requires three operators: *join*, *map* and *reduce*. The map task is used to let each vertex know the labels of its neighbors. The reduce task uses an automatic label selecting strategy introduced in Ref. 16 to determine the new labels of each vertex. The critical ideas of the strategy are as follows.

(a) The labels are sorted according to their accumulated belonging coefficients in a descending order.
(b) The ankle value a_k of the labels and its corresponding label index i_a are computed.
(c) Labels whose indices are greater than i_a are discarded.
(d) The left labels form new label set $L'(i)$.

The strategy has the advantage that there is no need to use a parameter to filter the labels like the v parameter of COPRA.

3.2. *The PMLPA algorithm*

By combining the parallel steps discussed above together, a Parallel Multi-Label Propagation Algorithm (PMLPA) is designed as in Algorithm 1.

Algorithm 1. PMLPA

```
Input: network G=(V,E), where V is the vertex set and E is the edge
 set, maximum iterations maxIter
Output: Community set {Ci}
    //PARALLEL ADJACENCY LIST GENERATION
1:  map(key=line offset, value=line string)
2:   output (key=i, value=j);
3:   output (key=j, value=i);
4:  reduce (key=i, values={j, k, …})
5:   output (key=i, value=NB(i));
6:  calculate rough core set;
    //PARALLEL VERTEX LABEL INITIALIZATION
7:  map (key=i, value=NB(i))
8:   output (key=i, value=L(i));
9:  iter=1;
10: while iter <= maxIter
    //PARALLEL LABEL UPDATING
11:  join (key=i, value=NB(i)), (key=i, value=L(i)))
12:    output (key=i, value (NB(i), L(i)));
13:  map (key=i, value=(NB(i), L(i)))
14:   for each j∈NB(i) output (key=j, value= L(i));
15:  reduce (key=i, values={L(j)})
16:   output (key=i, value=L'(i));
17:  if termination conditions are satisfied then
18:   break;
19:  end if
20:  iter=iter+1;
21: end while
22: extract community set {Ci} from vertex labels;
23: remove communities contained by other communities from {Ci};
24: return {Ci};
```

3.3. *Complexity analysis*

The parallel adjacency list generation needs $O(m/n_c)$ time where m is the number of edges and n_c represents the number of machines in the cluster. The vertex label initialization needs $O(n/n_c)$ time. The vertex label updating procedure needs $O(m/n_c)$ time. Therefore, the total time complexity of PMLPA is $O(m/n_c)$. Since each parallel computation procedure needs to store the whole network in the distributed memory, the total space complexity of PMLPA is $O(m)$. However, the space occupation is distributed among all the machines in the cluster. Hence, the scalability of PMLPA is much better than single-machine system.

4. Parallel Multi-Label Propagation Based on Influence Model

4.1. *Design ideas*

Each vertex or label in PMLPA is treated equally. In fact, some vertices or labels are always more important than the others. For example, a recommendation from a famous scholar usually receives more attentions than the recommendations from an ordinary person. Therefore, we have the following property for improving PMLPA.

Property 2. The influence of neighboring vertices and their labels should be considered in multi-label propagation.

Description: First, by putting weight to each vertex, we can develop a scheme to evaluate the influence of vertices and prevent the weak labels held by a large number of common vertices from affecting the strong labels hold several important vertices. Hence, the algorithm can converge more rapidly and more stably. Second, by putting weights to each label, we extend the vertex influence to the evaluation of labels. Labels recommended by more influential vertices are more possible to be spread over the network.

In this section, we propose a novel influence model that integrates vertex influence with label influence. Vertex influence can be calculated in a way similar to Ref. 17.

$$VI(i) = \frac{W(i)}{\sqrt{1 + \sum_{j \in NB(i)} W^2(j)}}$$

$$W(i) = \sum_{j \in NB(i)} w(i,j) + \sum_{j \in NB(i)} \sum_{k \in NB(j)} w(j,k) \tag{2}$$

$$w(i,j) = \frac{|NB(i) \cap NB(j)|}{|NB(i) \cup NB(j)|}$$

In Eq. (2), $VI(i)$ and $W(i)$ represent the influence and weight of vertex i, separately. $w(i, j)$ represents the weight of edge e_{ij} and is calculated by the Jaccard index of i and j.

The parallelization of vertex influence calculation is composed of three critical steps.

(1) Parallel Jaccard index calculation

Parallel Jaccard index calculation requires more than one map/reduce job. Since Jaccard index depends on the information of two vertices' neighbor sets, a map task has to be used to send the neighbor set of a vertex to each of its neighbor. The input of the map task is (key = i, value = $NB(i)$). The output of the map task is (key = (i, j), value = $NB(i)$). The reduce task collects $NB(i)$ and $NB(j)$ corresponding to (i, j) and return their Jaccard index (key = (i, j), value = $w(i, j)$). Another reduce task is necessary to transform (key = (i, j), value = $w(i, j)$) into (key = i, value = set of $w(i, j)$) for each vertex i.

(2) Parallel vertex weight calculation

The calculation of $w(i, j)$ in Eq. (2) depends on the neighbor set $NB(i)$ and the neighbor set $NB(j)$ of each of i's neighbor j. Therefore, we use the following parallel operators to finish the job.

As shown in Fig. 2, the parallel vertex weight calculation contains four steps. First, the neighbor set $NB(i)$ and weight set $\{w(i, j)\}$ are joined together. Second, the sum s_i of the weights of i to all its neighbors is calculated by a map task. Third, s_i is sent to each of i's neighbor by another map task. Finally, $W(i)$ is calculated by collecting all the sums of the weights of i's neighbors and the neighbors of its neighbors through the reduce task.

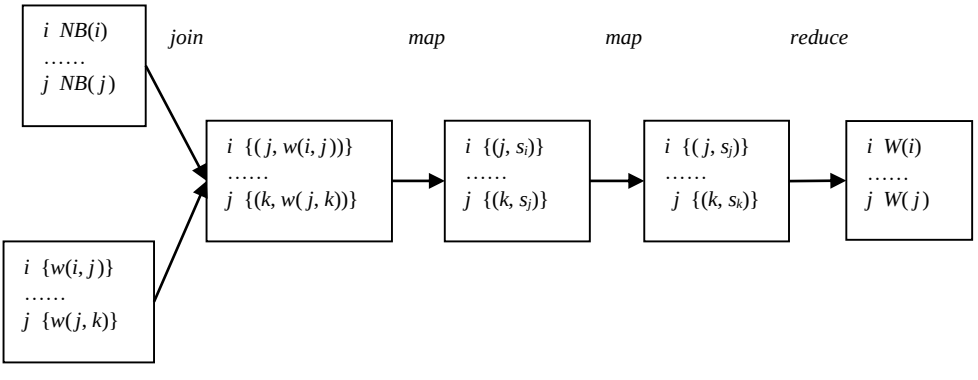

Fig. 2. Scheme for parallel vertex weight calculation.

(3) Parallel vertex influence calculation

After joining $W(i)$ and $NB(i)$ for each vertex i, $W(i)$ is sent by a map task to each neighbor of i where the sum s_i of the squares of $W(i)$s is calculated. Then, a reduce task is enough to calculate each vertex's influence.

The label influence can be calculated with respect to vertex influence as follows.

$$LI(l) = \frac{\sum_{j \in VS_l} w(i,j)}{\sqrt{\sum_{j \in VS_l} w^2(i,j)}} + \frac{\sum_{j \in VS_l} VI(j)}{\sqrt{\sum_{j \in VS_l} VI^2(j)}}, \quad (3)$$

where $LI(l)$ represents the influence of label l of vertex i, and $VS(l)$ is the set of neighbors of i that contain l in its label set. The parallelization of label influence calculation is embedded into the parallel label updating step of PMLPA. The new parallel label updating step is illustrated in Fig. 3.

The differences between Fig. 3 and Fig. 1 are: first, four datasets including vertex influence and label influence are joined in the new scheme; second, $L(i)$ now represents $(l, LI(l))$ instead of (l, bc_l); third, the map task is now used to calculate label influence according to Eq. (3) where $s_w(l_i)$ and $s_v(l_i)$ represent the sum of $w^2(i, j)$ and the sum of $VI^2(j)$, separately.

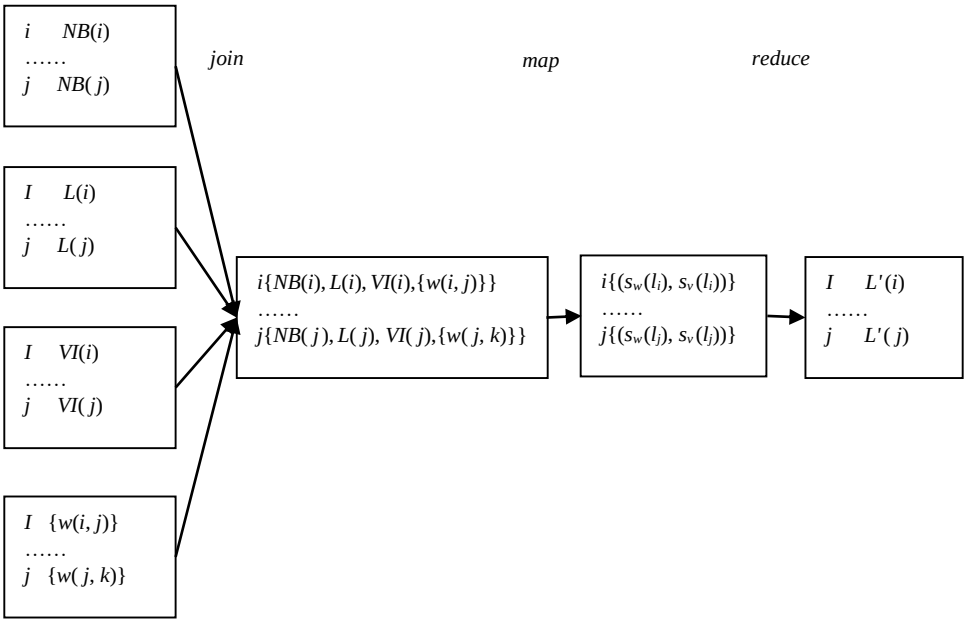

Fig. 3. Scheme for parallel label updating based on the new influence model.

4.2. *The PMLPA-IM algorithm*

The new algorithm that based on the new influence model is called PMLPA-IM (PMLPA with Influence Model), whose details are given in Algorithm 2.

4.3. *Complexity analysis*

The parallel Jaccard index calculation needs $O(nk^2/n_c)$ time where n is the number of vertices and k is the average vertex degree. The parallel vertex weight and vertex influence

Algorithm 2. PMLPA-IM

Input: network G=(V,E), where V is the vertex set and E is the edge set, maximum iterations maxIter
Output: Community set $\{C_i\}$

```
    //PARALLEL ADJACENCY LIST GENERATION
1:  map(key=line offset, value=line string)
2:   output (key=i, value=j);
3:   output (key=j, value=i);
4:  reduce (key=i, values={j, k, …})
5:   output (key=i, value=NB(i));
6:  calculate rough core set;
    //PARALLEL VERTEX LABEL INITIALIZATION
7:  map (key=i, value=NB(i))
8:   output (key=i, value=L(i));
    //PARALLEL JACCARD INDEX CALCULATION
9:  map (key=i, value=NB(i))
10:  output (key=(i,j), value=NB(i));
11: reduce (key=(i,j),values={NB(i),NB(j)})
12:  output (key=(i,j), value=w(i,j));
13: reduce (key=(i,j),values={w(i,j)})
14:  output (key=i, value={w(i,j)});
    //PARALLEL VERTEX WEIGHT CALCULATION
15: join (key=i, value=NB(i)), (key=i, value={w(i,j)})
16:  output (key=i, value={(j,w(i,j))});
17: map (key=i, value={(j,w(i,j))})
18:  output (key=i, value={(j,s_i)});
19: map (key=i, value={(j, s_i)})
20:  output (key=j, value={(i,s_i)});
21: reduce (key=i, values={(i,s_i)})
22:  output (key=i, value=w_i);
    //PARALLEL VERTEX INFLUENCE CALCULATION
23: join (key=i, value=NB(i)), (key=i, value={w_i})
24:  output (key=i, value={( NB(i),w_i)});
25: map (key=i, value={( NB(i),w_i)})
26:  output (key=i, value=s_i);
27: reduce (key=i, values={s_i})
     output (key=i, value=VI_i);
28: iter=1;
29: while iter <= maxIter
    //PARALLEL LABEL UPDATING
30:  join (key=i, value=NB(i)), (key=i, value=L(i))), (key=i,
    value=VI(i)), (key=i, value={(w(i,j)})
31:   output (key=i, value=(NB(i), L(i), VI(i),
        {w(i,j)}));
32:  map (key=i, value=(NB(i), L(i), VI(i), {w(i,j)}))
33:   output (key=i, value={(s_w(l_i), s_v(l_i))}
34:  reduce (key=i, values={(s_w(l_i), s_v(l_i))})
35:   output (key=i, value=L'(i));
36:  if termination conditions are satisfied then
37:   break;
38:  end if
39:  iter=iter+1;
40: end while
41: extract community set {C_i} from vertex labels;
42: remove communities contained by other communities from {C_i};
43: return {C_i};
```

calculation needs $O(m/n_c)$ time. The new parallel label updating procedure also needs $O(m/n_c)$ time. Therefore, the total time complexity of PMLPA-IM is $O(m/n_c)$. The space complexity of PMLPA-IM is the same as PMLPA.

5. Experiments

We compared PMLPA and PMLPA-IM with COPRA and CFinder[18] on both artificial and real networks to evaluate their performance. The algorithms are written in Java or Scala on Spark framework,[19,20] The artificial networks were generated by the benchmark network generator designed by Lancichinetti and Fortunato.[21] The real networks were obtained from the SNAP project website.[22] The details of the experimental networks are summarized in Table 1. Parameters *n*, *m*, *mu*, *k*, *maxk*, *minc*, *maxc*, *on*, and *om* represent the number of vertices and network edges, the mixing parameter required by the network generator, the minimum and maximum degrees of the vertices, the average and maximum community sizes, the number of overlapping vertices, and the number of memberships of the overlapping nodes, respectively.

Table 1. Details of the experimental networks.

Networks		Parameters
Artificial networks		n = 100 k ~ 500 k, *mu* = 0.3, k = 15, *maxk*=150, *minc* = 50, *maxc* = 100, *om* = 5, *on* = 0.1n
Real networks	Amazon	n = 334863, m = 925872
	DBLP	n = 317080, m = 1049866
	Youtube	n = 1134890, m = 2987624

The sizes of the artificial networks vary from 100 k (one thousand vertices) to 500 k (fifty thousand vertices). The ratio of the vertices belonging to multiple communities is 0.3 (30 percent). The average and the maximum vertex degrees are 15 and 150, separately. Community size varies from 50 to 100.

The accuracy of the algorithms is measured by the Normalized Mutual Information (NMI)[23] if the community structures of a network are known or overlapping modularity Q_{ov} [4,24] if this information is absent. The equations for NMI and Q_{ov} are as follows.

$$NMI(X|Y) = 1 - [H(X|Y) + H(Y|X)]/2$$
$$H(X) = -\sum_{i=1}^{n} p(x_i \log x_i) \tag{4}$$
$$H(X) = \sum_{i,j} p(x_i, y_j) \log \frac{p(y_j)}{p(x_i, y_j)}$$

where X and Y denote the sets of true communities and discovered communities, respectively. x_i and y_i represent a vertex in X and Y, respectively. $H(X)$ is the entropy of X, and $H(X|Y)$ is the conditional entropy of X given Y.

$$Q_{ov} = \frac{1}{2m}\sum_{c=1}^{n_c}\sum_{i,j\in C_i}\frac{1}{O_i O_j}\left[\frac{1}{m} - \frac{k_i k_j}{2m}\right], \tag{5}$$

where m, n_c, C_i, and k_i are the edge number, the number of communities, community C_i, and the degree of vertex i, respectively.

All the experiments are performed on a cluster composed of eight machines with the same configuration: 2.0 GHz 2 cores CPU, 16GB RAM, Hadoop 2.4.0 and Spark 1.5.0.

5.1. *Experimental results on artificial networks*

(1) Effect of Network Size

Artificial networks whose sizes vary from 100 k to 500 k were used to evaluate the accuracy and run time of the algorithms. The experimental results are illustrated in Figs. 4 and 5.

As shown in Fig. 4, the accuracy of PMLPA and PMLPA-IM are consistently better than COPRA and CFinder. PMLPA-IM performs slightly better than PMLPA because it considers the influence of vertices and labels in labels updating as suggested by Property 2. Since the parameters of all the networks are the same except for their sizes, the NMI values of the algorithms on the networks are close. The only exception is CFinder. The dependence on the detection of the k-cliques weakens its performance.

Figure 5 shows that PMLPA and PMLPA-IM run consistently faster than the other algorithms. According to Property 1, PMLPA and PMLPA-IM are both implemented on the MapReduce mode and efficient parallel operators. Therefore, they exhibit excellent scalability. PMLPA-IM runs more slowly than PMLPA because it needs to spend extra time to calculate the influence of vertices and labels according to Property 2. Influence calculation may be time-consuming if a large number of vertices in the network have high degrees.

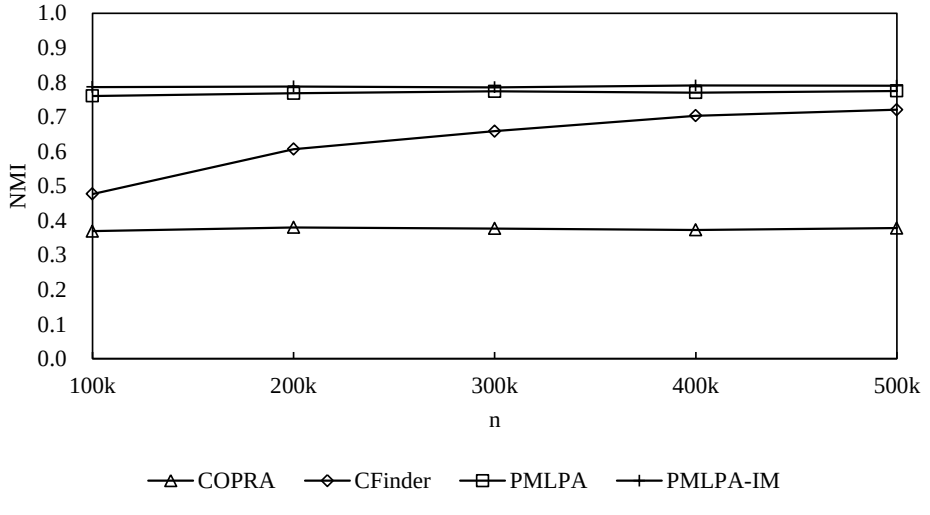

Fig. 4. NMI of the algorithms on artificial networks with varying sizes.

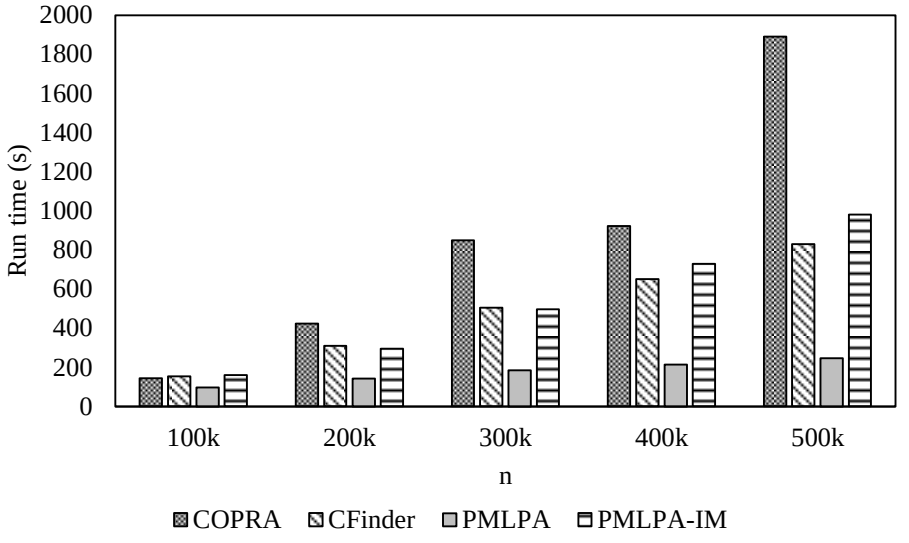

Fig. 5. Run time of the algorithms on artificial networks with varying sizes.

(2) Effect of Mixing Degree

Identifying highly overlapping communities in large networks is an important capability for community discovery algorithms. Hence, experiments were conducted on artificial networks with varying *mu* values to evaluate the algorithms. We used the 100 k network in the experiments. The *mu* values are varied and other parameters are identical to Table 1. The experimental results are given in Fig. 6.

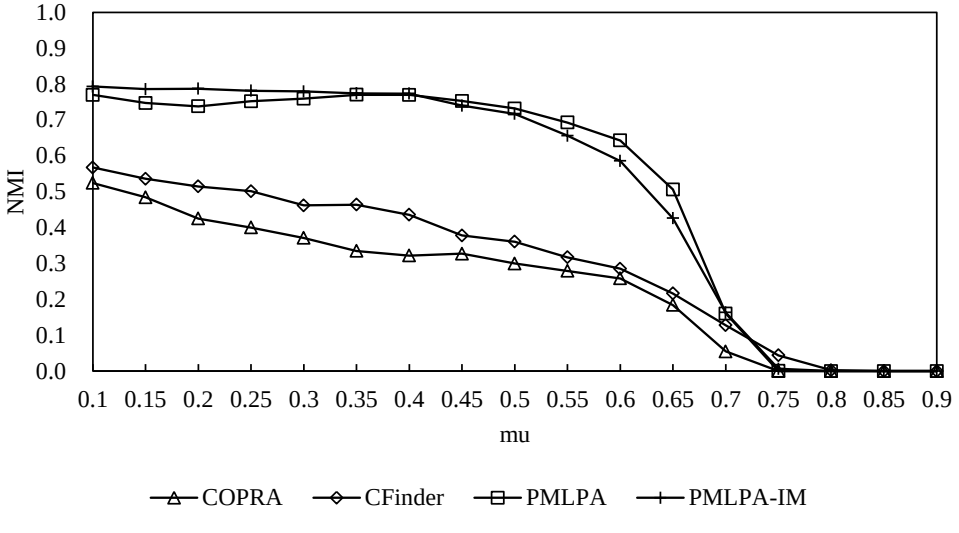

Fig. 6. NMI of the algorithms on artificial networks with varying *mu* values.

Figure 6 shows that the new updating strategy developed according to Property 1 and 2 greatly help improve the accuracy of PMLPA and PMLPA-IM. They outperform COPRA and CFinder more than 20% when the value of *mu* is smaller than 0.7. It is always more difficult to detect communities when more communities are mixed up. Therefore, all the algorithms fail when the value of *mu* is greater than 0.7. It is interesting to notice that PMLPA-IM performs better than PMLPA when the value of *mu* is smaller than 0.4 and worse than PMLPA when the value of *mu* is greater than 0.4. The phenomenon reveals the fact that the influence model may not always beneficial. It is possible to contribute negatively to community discovery when the number of overlapping communities reaches a certain value.

(3) Effect of Cluster Scale

Experiments were conducted on the artificial networks with varying number of machines to evaluate the effect of cluster scale on the performance of PMLPA-IM. The results are given in Fig. 7.

Each curve in Fig. 7 represents the run time of the algorithm on different size of cluster for an artificial network. It is easy to find that the run time decreases with the increase of the number of machines. The more machines are involved in the computation, the faster the algorithm will run. However, the speedup of the algorithm on different network is different. On small networks like the 100 k network, the reduction in run time is not noticeable. On large networks like the 500 k network, on the contrary, the reduction in run time is significant. It may be explained by the difference in effective workload. The effective workload is the ratio of run time spent on executing algorithm steps (called work

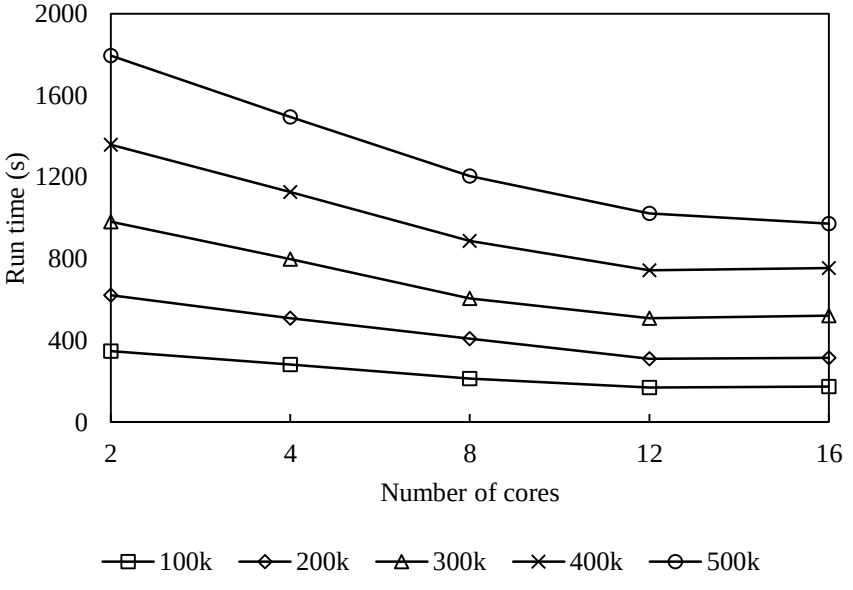

Fig. 7. Run time of PMLPA-IM on artificial networks with varying sizes.

time) to the total run time. When the network is not very large, the time spent on cluster administration and network communication (called extra time) may be comparable to work time. In this case, increasing the number of machines in the cluster does not lead to the expected speedup effect. When the network is large enough, work time will be far greater than extra time. Therefore, the benefit brought by expanding the cluster becomes remarkable.

5.2. *Experimental results on real networks*

Networks in real world usually contain more complex community structures than the artificial networks. Hence, experiments were conducted on the real networks to evaluate the performance of the algorithms. The results are illustrated in Figs. 8 and 9.

As shown in Fig. 8, the accuracy of PMLPA and PMLPA-IM measured by Q_{ov} is superior to other algorithms on all the real networks. As stated previously, it largely attributes to the efficient label updating strategy designed according to Property 1 and 2. PMLPA-IM performs a little better than PMLPA because it additionally considers the influence of vertices and labels. CFinder runs too slowly on the Youtube network. Therefore, its Q_{ov} value on the network is absent from Fig. 8.

Since real networks may contain complex community structures that greatly affect the time for community discovery, the vertical axis scale of Fig. 9 is set to the power of 10. As shown in Fig. 9, PMLPA and PMLPA-IM algorithms run faster than the other algorithms. PMLPA runs slightly faster than PMLPA-IM because it spends more time on the calculation of the influence of vertices and labels. We can find by comparing Fig. 9 with Fig. 5 that the difference between PMLPA and PMLPA-IM on run time on the real networks is not as remarkable as that on the artificial networks. Real networks are usually

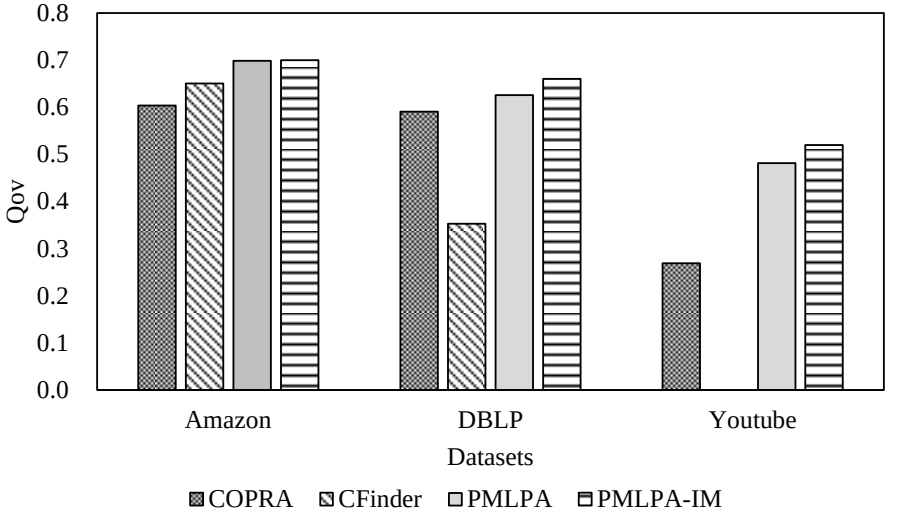

Fig. 8. NMI of the algorithms on the real networks.

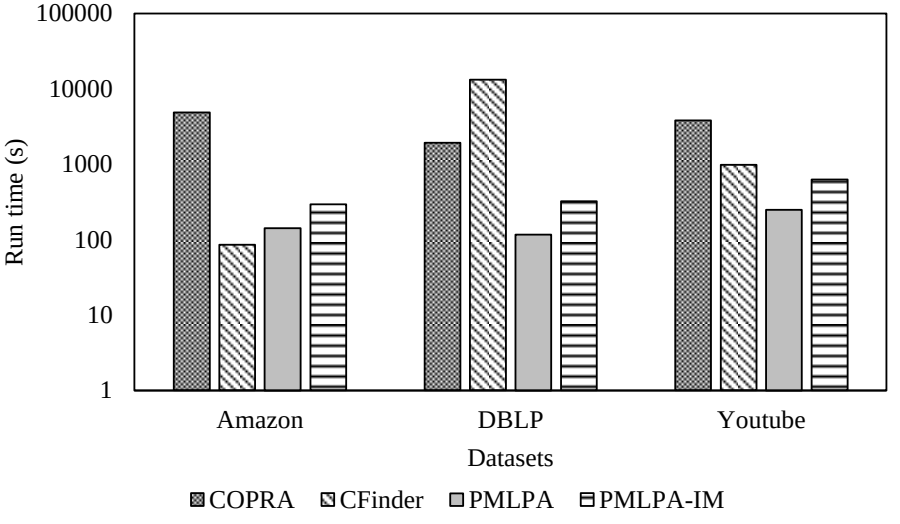

Fig. 9. Run time of the algorithms on real networks.

sparse and contain only a few vertices with high degrees. Therefore, it does not require too much time to calculate the influence of vertices and labels.

6. Conclusion

In this paper, a new Multi-Label Propagation Algorithm based on parallel computation model and its improvement that considers the influence of vertices and labels are proposed. First, the design ideas behind the algorithms are discussed in detail. Second, the pseudo-codes of the algorithms based on the parallel operators are presented with the complexity analysis. Finally, the experiments on artificial and real networks demonstrate that the proposed algorithms are capable of processing large networks efficiently with high accuracy. In future, we plan to further improve our algorithms by incorporating more efficient label updating strategies and more powerful parallel operators.

Acknowledgments

This work is partly supported by the National Natural Science Foundations of China under Grant Nos. 61300104 and 61672159, the Program for New Century Excellent Talents in Fujian Province University under Grant No. JA13021, the Key Project of Industry-Academic Cooperation of Fujian Province under Grant No. 2014H6014, the Fujian Natural Science Fund for Distinguished Young Scholar under Grant No. 2014J06017, the Fujian Collaborative Innovation Center for Big Data Application in Governments, and the Technology Innovation Platform Projects of Fujian Province under Grant Nos. 2014H2005 and 2009J1007, and the Natural Science Foundations of Fujian Province under Grant Nos. 2013J01230 and 2017J01752.

References

1. M. Girvan and M. E. J. Newman, Community structure in social and biological networks, *Proc. of the National Academy of Sciences* **99** (2002) 7821–7826.
2. J. R. Tyler and D. M. Wilkinson, B. A. Huberman, E-mail as spectroscopy: Automated discovery of community structure within organizations, *The Information Society* **21** (2005) 143–153.
3. F. Radicchi, C. Castellano, F. Cecconi, V. Loreto and D. Parisi, Defining and identifying communities in networks, *Proc. of the National Academy of Sciences of the United States of America* **101** (2004) 2658–2663.
4. M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks, *Physical Review E* **69** (2004) 026113.
5. M. E. J. Newman, Fast algorithm for detecting community structure in networks, *Physical Review E* **69** (2004) 066133.
6. V. D. Blondel, J. L. Guillaume, R. Lambiotte and E. Lefebvre, Fast unfolding of communities in large networks, *Journal of Statistical Mechanics: Theory and Experiment* **2008** (2008) P10008.
7. U. N. Raghavan, R. Albert and S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, *Physical Review E* **76** (2007) 036106.
8. S. Gregory, Finding overlapping communities in networks by label propagation, *New Journal of Physics* **12** (2010) 103018.
9. Z. H. Wu, Y. F. Lin, S. Gregory, H. Y. Wan and S. F. Tian, Balanced multi-label propagation for overlapping community detection in social networks, *Journal of Computer Science and Technology* **27** (2012) 468–479.
10. J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, *Communications of the ACM* **51** (2008) 107–113.
11. S. Fortunato and M. Barthelemy, Resolution limit in community detection, *Proc. of the National Academy of Sciences* **104** (2007) 36–41.
12. B. H. Good, Y. A. Montjoye and A. Clauset, Performance of modularity maximization in practical contexts, *Physical Review E* **81** (2010) 046106.
13. W. Zhao, V. Martha and X. Xu, PSCAN: A parallel structural clustering algorithm for big networks in MapReduce, in *Proc. 2013 IEEE 27th Int. Conf. on Advanced Information Networking and Applications* (*AINA*) (Barcelona, Spain, 2013), pp. 862–869.
14. R. R. Li, W. Z. Guo, K. Guo and Q. R. Qiu, Parallel multi-label propagation for overlapping community detection in large-scale networks, in *Proc. Multi-disciplinary Trends in Artificial Intelligence* (Fuzhou, China, 2015), pp. 351–362.
15. I. X. Y. Leung, P. Hui, P. Lio and J. Crowcroft, Towards real-time community detection in large networks, *Physical Review E* **79** (2009) 066107.
16. J. B. Huang, H. L. Sun, D. Bortner and Y. G. Liu, Mining hierarchical community structure within networks from density-connected traveling orders, *Journal of Software* **22** (2011) 951–961.
17. X. S. Yi, Y. T. Han and X. W. Wang, Algorithm based on vertex influence for detecting local community structure, *Journal of Chinese Computer Systems* **34** (2013) 1975–1979.
18. G. Palla, I. Derenyi, I. Farkas and T. Vicsek, Uncovering the overlapping community structure of complex networks in nature and society, *Nature* **435** (2005) 814–818.
19. Apache software foundation, Apache Spark (2017), http://spark.apache.org.
20. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley M. J. Franklin, S. Shenker and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in *9th USENIX Conf. on Networked Systems Design and Implementation* (Lombard, United States, 2012), pp. 2-2.

21. A. Lancichinetti and S. Fortunato, Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities, *Physical Review E* **80** (2009) 016118.
22. J. Leskovec and A. Krevl, SNAP datasets: Large network dataset collection (2017), https://snap.stanford.edu/data.
23. A. Lancichinetti, S. Fortunato and J. Kertesz: Detecting the overlapping and hierarchical community structure in complex networks, *New Journal of Physics* **11** (2009) 033015.
24. H. Shen, X. Cheng, K. Cai and M. B. Hu, Detect overlapping and hierarchical community structure in networks, *Physica A: Statistical Mechanics and Its Applications* **388** (2009) 1706–1712.