International Journal of Innovative
Computing, Information and Control
Volume **12**, Number **1**, February **2016**

ICIC International ©2016 ISSN 1349-4198
pp. **263–273**

MR-ECOCD: AN EDGE CLUSTERING ALGORITHM FOR OVERLAPPING COMMUNITY DETECTION ON LARGE-SCALE NETWORK USING MAPREDUCE

Haitao He[1,2], Peng Zhang[1,2,*], Jun Dong[1,2] and Jiadong Ren[1,2]

[1]College of Information Science and Engineering
Yanshan University
[2]The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province
No. 438, Hebei Ave., Qinhuangdao 066004, P. R. China
{ haitao; dongjun; jdren }@ysu.edu.cn; *Corresponding author: peng891025@163.com

Received June 2015; revised November 2015

Abstract. *Overlapping community detection is progressively becoming an important issue in complex networks. Many in-memory overlapping community detection algorithms have been proposed for graphs with thousands of nodes. However, analyzing massive graphs with millions of nodes is impossible for the traditional algorithm. In this paper, we propose MR-ECOCD, a novel distributed computation algorithm using MapReduce to detect overlapping communities efficiently on large-scale network. Firstly, the similarities of all adjacent edges are calculated by SimilarityMap algorithm to measure the distance of edges. Secondly, we define the direct edge communities (DEC) and mergeable direct edge communities (MDEC) based on edge density clustering method. Then, MarkMap algorithm and ClusteringReduce algorithm are designed to mark DEC and merge MDEC respectively for getting finally edge communities (FEC). Finally, we transform the FEC into node communities, and a node is an overlapping node in node communities if it belongs to different edges in different FEC. MR-ECOCD consists of four major stages, and all operations are executed in parallel using MapReduce. Extensive experiments show that our algorithm can effectively and fast detect overlapping communities.*
Keywords: Overlapping community detection, MapReduce, Large-scale networks, Edge clustering

1. **Introduction.** Many complex systems or structures can be represented by the complex networks in the real world. And overlapping community detection from network, such as social network [1], semantic social network [2], citation network [3], is progressively becoming an important issue. The discovery of network communities provides us a much better understanding about the structural topology of each community as well as its organization principles.

At present, some traditional algorithms [4] have been proposed to discover overlapping communities. Based on the assumption that a community consists of overlapping sets of fully connected subgraphs, clique percolation method (CPM) [5] detects communities by searching for adjacent cliques. Therefore, it is suitable for networks with dense connected parts. However, because of the iterative calculation in memory, it fails to terminate in large social networks. Local expansion methods [6-8] are based on growing a natural community or a partial community. Most of them rely on a local benefit function that characterizes the quality of a densely connected group of nodes. The label propagation method, in which nodes with same label form a community, has been extended to overlapping community detection by allowing a node to have multiple labels. In COPRA

[9], each node updates its belonging coefficients by averaging the coefficients from all its neighbors at each time step in a synchronous fashion.

The edge graph partitioning methods discover community structure by partitioning edges of original graph. And a node in the original graph is called overlapping if edges connected to it are put in more than one partition. Evans and Lambiotte [10] projected the network into a weighted line graph, whose nodes are the links of original graph. Then disjoint community detection algorithms can be applied. The node partition of a line graph leads toan edge partition of the original graph. Ahn et al. [11] proposed a hierarchical clustering method based on edge similarity. Given a pair of links e_{ik} and e_{jk} incident on a node k, a similarity can be computed via the Jaccard Index. Single-linkage hierarchical clustering is then used to build a link dendrogram. Cutting this dendrogram at some threshold yields link communities. Lim et al. [12] proposed a framework of the link-space transformation which transforms a given original graph into a link-space graph. Using the link-space transformation is not dependent on a specific community detection algorithm. Therefore, they adopt the algorithm SCAN since it identifies the hub or outlier nodes that do not belong to any community.

The previous algorithms focus on how to improve the accuracy of detecting overlapping communities. However, in big data era, large or super large scale network is becoming increasingly commonplace. As a result, the traditional methods are unable to meet the requirements of big data environment. Moreover, the research of overlapping community discovery algorithm on large scale network is very necessary.

Google released the MapReduce framework [13] in 2004. MapReduce is a programming model for processing large data sets with a parallel and distributed algorithm on a cluster. And MapReduce is used to speed up the execution in many fields including community detection. Moon et al. [14] proposed a parallel version of the GN algorithm under MapReduce model to support large-scale networks and suggested an approximation technique to further speed up community detection processes. Ovelgonne [15] presented a distributed ensemble learning algorithm for graph clustering that scales to networks with billions of edges using MapReduce. Su et al. [16] proposed a distributed computation method which combines MapReduce and the TTT algorithm to speed up the discovery of all maximal cliques in large-scale social networks.

The traditional community-detection algorithms can detect overlapping communities from small-scale network with thousands of vertices; however, these approaches are not suitable for processing networks with more than millions of nodes, because they are exclusively non-distributed, in-memory and singlethreaded algorithms.

The above issues motivate us to design, implement,and evaluate an efficient community-detection solution for large-scale networks with millions of nodes or bigger using MapReduce. In this paper, we define *direct edge community* and *mergeable direct edge community*, and then propose a novel distributed computation algorithm using MapReduce framework. Our algorithm consists of four major stages, and each stage is corresponding a MapReduce job which takes computing on the whole network and can be executed concurrently in a distributed environment.

The rest of the paper is organized as follows. Section 2 contains some basic definitions. Section 3 describes the algorithm MR-ECOCD. Section 4 is experiment analysis. Finally, Section 5 concludes this study.

2. **Definitions.**

Community detection can be transformed into a network clustering. In this paper, for discovering overlapping communities, we cluster networks by the edges like structure clustering algorithm SCAN [17] which clusters networks by the nodes. The community of edges is preferable to that of nodes from a theoretical point of view, because

the edge is likely to have a unique identity whereas the node tends to have multiple identities. And a node is an overlapping node in node communities if it belongs to different edges in different edge communities.

Edge Clustering: Let $G = (V, E)$ be a given network, where V is the set of vertices and E is the set of edges of G. Then, a collection P of subsets of E is said to be an *edge* clustering of G if the elements of P are pairwise disjointed and the union of the elements of P is equal to E.

The membership of a node is determined by those of its incident edges. Note that a node participates in multiple communities if the node has multiple incident edges with different types of relationship.

Neighbor Edges: Given an edge $e = (v, w)$, $e \in G$, the *neighbor edges* $N(e)$ of e is all the edges connected to the nodes v and w, except e. $N(e)$ can be described as:

$$N(e) = \{(v, i) \in E | i \in N(v)\} \cup \{(w, j) \in E | j \in N(w)\} - \{e\} \tag{1}$$

where $N(v)$ and $N(w)$ denote the neighbor nodes of v and w respectively.

ε-Neighbor Edges: Given a parameter ε, the *ε-neighbor edges* of e, denoted by $N_\varepsilon(e)$, refers to the set of edges in all neighbor edges of e whose similarity with e is greater than ε. $N_\varepsilon(e)$ can be described as:

$$N_\varepsilon(e) = \{l \in N(e) | \sigma(l, e) \geq \varepsilon\} \tag{2}$$

We adopt the edge similarity proposed by Ahn et al. [11] for $\sigma(e, l)$. The edge similarity is defined by Formula (3), then $\Gamma(v_i) = \{i' \in V | (i, i') \in E\} \cup \{i\}$ and $e = (k, i)$, $l = (k, j)$. Note that two edges of G have an edge similarity if and only if they share a common endpoint in G.

$$\sigma(e, l) = \frac{|\Gamma(v_i) \cap \Gamma(v_j)|}{|\Gamma(v_i) \cup \Gamma(v_j)|} \tag{3}$$

Core Edge: For given parameters ε and μ, edge e is a *core edge*, if and only if its *ε-neighbor edges* contains at least μ vertices. A *core edge* can be described as follows.

$$Core_{\varepsilon,\mu}(e) \equiv |N_\varepsilon(e)| \geq \mu \tag{4}$$

Direct Density Reachability: For a given *core edge* e, l is *direct density reachability* from e, if l belongs to e's *ε-neighbor edges*. A *direct density reachability* can be described as follows.

$$Direct_{\varepsilon,\mu}(e, l) \equiv Core_{\varepsilon,\mu}(e) \wedge l \in N_\varepsilon(e) \tag{5}$$

Based on the above definition, our definitions are given below for distributed parallel computing using MapReduce.

Definition 2.1. ***Direct Edge Community (DEC)*** *For a given core edge e, direct edge community is the set of all edges which are direct density reachability from e and e. A DEC can be described as follows.*

$$DEC = \{l | l \in N_\varepsilon(e)\} \cup \{e\} \tag{6}$$

Definition 2.2. ***Mergeable Edge Community (MEC)*** *For two given direct edge communities DEC_i and DEC_j, if there is an l that also belongs to DEC_i and DEC_j, then we call DEC_i and DEC_j are mutually mergeable edge community.*

Remark that the mergeable edge community is a symmetric relation and it is transitive.

Definition 2.3. ***Final Edge Community (FEC)*** *Final edge community is a set of mergeable edge communities with same edge.*

Final edge communities meet the following criteria:

- *If a final edge community is merged by the set* $S = \{DEC_1, DEC_2, \ldots, DEC_n\}$*, then there is an edge set* $E = \{l_1, l_2, \ldots, l_m\}$ *such that the set* S *can form a unique connected graph.*
- *There is an edge* l *such that* $l \in DEC_i$ *and* $l \in DEC_j$*, if* $DEC_i \in S$ *then* $DEC_j \in S$*, and vice versa.*

In fact, the *edge clustering* of a graph G is translated to calculate all *final edge communities*. Then translating the *final edge communities* to node communities is our ultimate goal.

3. **The Design and Analysis of MR-ECOCD.** The algorithm MR-ECOCD consists of four steps: searching neighbor nodes, calculating edge similarity, marking and clustering communities, transforming edge communities. These four steps correspond to four MapReduce jobs, and each job can run itself in a distributed environment. These jobs are executed sequentially, and the posterior job relies on the output of the prior job. The following is the outline of MR-ECOCD algorithm.

- Traverse the original network, search the neighbors of each node respectively and store the result in a distributed database.
- Calculate the similarity of all the adjacent edges, and the result is stored in distributed file system (DFS).
- According to the similarity of the previous step, get all the *direct edge communities*, and merge all the *mergeable edge communities* to get the *final edge communities*.
- Transform the *final edge communities* into the *final point communities*.

3.1. **Searching neighbor nodes.** The original network data is a set of edges, and each row represents an edge which consists of two nodes. In the Map stage, the Mapper collects all edges and transfers them to the key-value format. Then in the Reduce stage, the Reducer puts the same-key records together to represent the neighbors of the key node.

NeighborsMap: The Map stage of searching neighbor nodes

Input: $< k_1, v_1 >$
Output: $< k_2, v_2 >$
1: get v and w from v_1
2: $k_2 = v$, $v_2 = w$
3: WriteToDFS(k_2, v_2)

The input of the Map stage is the rows of the original file, k_1 is the row identifier and v_1 is the content of this row.

NeighborsReduce: The Reduce stage of searching neighbor nodes

Input: $< v, NList >$
Output: $< v, neighbors >$
1: for each w_i in *NList* do
2: *neighbors* append w_i
3: end for
4: insert $< v, neighbors >$ into distributed database

For each node, we get its neighbors from *NList* and append to the string variable *neighobrs*, then save this node and its neighbors into distributed database. We save neighbor nodes into distributed database, because in the next step the algorithm will query some node's neighbor nodes when calculating edge similarity, and distributed database is well suited for applications with a large number of query operations.

3.2. **Calculating edge similarity.** In this step, the algorithm traverses the information saved in distributed database, calculates all the similarities between adjacent edges in the Map stage, and then writes each edge, neighbor edges of that edge and the similarities between them to Distributed File System in the Reduce stage.

SimilarityMap: The Map stage of calculating edge similarity

Input: $< v_k, neighbors >$
Output: $< e, l - s >$
1: for each v_i in *neighbors* do
2: for each v_j in *neighbors* and $(j > i)$ do
3: $list_1 =$ getNeighbors(v_i) add v_i
4: $list_2 =$ getNeighbors(v_j) add v_j
5: $s =$ getSimilarity$(list_1, list_2)$
6: WriteToDFS$(e_{ik}, e_{jk} - s)$
7: WriteToDFS$(e_{jk}, e_{ik} - s)$
8: end for
9: end for

In the Map stage, for each node, such as v_k, the *neighbors* is a string variable like '$v_1, v_2, v_3, \ldots, v_n$', we calculate the similarity between each two edges that all of them have a common node v_k. The method getNeighbors in lines 3 and 4 of SimilarityMap is to get the neighbors of v_k from distributed database, and the method getSimilarity in line 5 uses the formula $\sigma(e_{ik}, e_{jk}) = \frac{|\Gamma(v_i) \cap \Gamma(v_j)|}{|\Gamma(v_i) \cup \Gamma(v_j)|}$ to get the similarity of e_{ik} and e_{jk}.

SimilarityReduce: The Reduce stage of calculating edge similarity

Input: $< e_{vw}, SList >$
Output: $< e_{vw}, n\_similarities >$
1: for each $e_{ij} - s$ in *SList* do
2: *n_similarities* append $e_{ij} - s$
3: end for
4: WriteToDFS$(e_{vw}, n_similaritys)$

In lines 1-3 of the algorithm *SimilarityReduce*, for each edge e_{vw}, the *SList* is a collection of its neighbors and similarities, the element of the *SList* likes $e_{ij} - s$ where e_{ij} is the neighbor of e_{vw} and s is the similarity of them, then *n_similaritys* is a string variable for saving result to DFS.

3.3. **Marking and clustering communities.** Based on the similarities of the adjacent edges, the algorithm *marking communities* gets all the *direct edge communities* in the Map stage, then gets all *mergeable edge communities* and merges them to get *final edge communities* in the Reduce stage.

For given parameters μ and ε, in lines 1-6 of the algorithm *MarkMap*, the variable *Num* is to count the number of e_{vw}'s neighbor edges where the similarity of them meets the condition $s > \varepsilon$, and *NList* is used to save the list of neighbors which meet that condition. In line 7, we judge whether a given edge e_{vw} is a core edge or not. If it is a core edge, then we can get a *direct edge community*, and in lines 8-13 we flag this *direct edge community* to *comid* which is a unique identification to other *direct edge communities*.

In the algorithm *ClusteringReduce*, *CList* is the collection of e_{vw}'s *direct edge communities* and they are mutually *mergeable edge communities* because they have the common edge e_{vw}, the variable *CMap* is used to save the map of *direct edge community* and *final edge community*. Based on the judgment of whether someone in *CList* has been mapped

MarkMap: The Map stage of marking and clustering communities

Input: $< e_{vw}, n\_similarities >$, μ, ε
Output: $< e_{vw}, TaskID\_i >$

```
1: for each (n, s) in n_similarities do
2:    if s > ε then
3:       Num++
4:       NList.add(n)
5:    end if
6: end for
7: if Num > μ then
8:    for each edge in NList do
9:       Flag edge's community to comid
10:      WriteToDFS(edge, comid)
11:   end for
12:   Flag e_vw's community to comid
13:   WriteToDFS(e_vw, comid)
14: end if
```

ClusteringReduce: The Reduce stage of marking and clustering communities

Input: $< e_{vw}, CList >$
Output: $< e_{vw}, endC >$, Final Edge Communities

```
1: for each c in CList do
2:    if CMap.contains(c) and flag == false then
3:       flag = true
4:       cnum = CMap.get(c)
5:    end if
6: end for
7: if flag = true then
8:    endC = cnum
9: else
10:   endC = newC
11: for each c in CList do
12:      CMap.put(c, endC)
13: end for
14: WriteToDFS(e_vw, endC)
15: Return Final Edge Communities
```

to *final edge community* in lines 1-6, we get the community identification of e_{vw} as *endC* in lines 7-10, and in lines 11-13, we map others of *CList* to *endC* and save to *CMap*. Finally, we get the final edge communities.

3.4. **Transforming edge communities.** This algorithm will transform the *final edge communities* into *node communities* which may include overlapping communities.

In the Map stage, for a given edge, for example e_{ij} and its community is c, we flag the community of node v_i and v_j to c. And for a given node, for example v_i, we can get all the communities *CList* of it in the Reduce stage.

MR-ECOCD consists of four major stages, and each stage is corresponding to a MapReduce job, and each job can be executed concurrently in a distributed environment on the

TransformMap: The Map stage of transforming edge communities

Input: $< e_{ij}, c >$
Output: $< v_i, c > < v_j, c >$
1: Flag v_i's community to c
2: Flag v_j's community to c
3: WriteToDFS(v_i, c)
4: WriteToDFS(v_j, c)

TransformReduce: The Reduce stage of transforming edge communities

Input: $< v_i, CList >$
Output: $< v_i, c >$
1: for each c in *CList* do
2: WriteToDFS(v_i, c)
3: end for

whole network so that our algorithm does not need in-memory loops or recursive operation which is not suitable for processing large-scale networks. In addition, MR-ECOCD is a distributed algorithm, and it can be run on Hadoop platform which is deployed on a cluster. Therefore, it is easily extensible for different size of the network. In fact, it can process networks with more than millions of nodes if we have big enough cluster.

4. **Experiments.** In this section, we examine the performance of our proposed algorithm MR-ECOCD. We compared the performance of four algorithms including our algorithm. The three existing algorithms have been recognized as the state-of-the-art overlapping community detection algorithms.

(1) MR-ECOCD: our proposed algorithm
(2) the CMP (Clique Percolation Method) [5]
(3) the link-partition method [10]
(4) the COPRA (Community Overlap Propagation Algorithm) [9]

4.1. **Data generation.** In order to demonstrate the performance of our proposed algorithms, we investigated the results on synthetic networks generated by the LFR benchmark [18].

The parameters we use for the LFR benchmark are described in Table 1. In our experiments, the synthetic network is built with varying the parameter values of N, $< k >$, mu, on and om. We set the value of $\max k$ to $2 < k >$, and default values are used for others.

Table 1. Parameters for the LFR benchmark

Parameter	Description
N	number of nodes
$< k >$	average degree
$\max k$	maximum degree
mu	mixing parameter
on	number of overlapping nodes
om	number of memberships of the overlapping nodes

4.2. **Evaluation metrics.** For the traditional graph partitioning, the *Normalized Mutual Information (NMI)* has been most widely used to measure the quality of partition when the ground-truth is known. Recently, Lancichinetti et al. [6] proposed an extended version of the *NMI*, which can be used for the case when a node may belong to more than

one cluster. In recent years, *NMI* has become the most widely-used community clustering quality standard. Note that the NMI is based on the information theory that compares the similarity between the memberships of two groups. Here, one group indicates the results of community detection, and the other group the ground-truth communities. Thus, the *NMI* naturally evaluates the quality of community detection. It ranges from 0 to 1 by normalization, and a higher value represents a better quality.

For each node i in cover C', its community membership can be expressed as a binary vector of length $|C'|$ (i.e., the number of clusters in C'). $(x_i)_k = 1$ if node i belongs to the k^{th} cluster C'_k; $(x_i)_k = 0$ otherwise. The k^{th} entry of this vector can be viewed as a random variable X_k, whose probability distribution is given by $P(X_k = 1) = n_k/n$, $P(X_k = 0) = 1 - P(X_k = 1)$, where $n_k = |C'_k|$ is the number of nodes in the cluster C'_k and n is the total number of nodes. The same holds for the random variable Y_l associated with the l^{th} cluster in cover C''. Both the empirical marginal probability distribution $P(X_k)$ and the joint probability distribution $P(X_k, Y_l)$ are used to further define entropy $H(X)$ and $H(X_k, Y_l)$.

The conditional entropy of a cluster X_k given Y_l is defined as $(X_k|Y_l) = H(X_k, Y_l) - H(X)$. The entropy of X_k with respect to the entire vector Y is based on the best matching between X_k and any component of Y given by

$$H(X_k|Y) = \min_{l \in \{1,2,\cdots,|C''|\}} H(X_k|Y_l).$$

The normalized conditional entropy of a cover X with respect to Y is

$$H(X|Y) = \frac{1}{|C'|} \sum_k \frac{H(X_k|Y)}{H(X_k)}.$$

In the same way, one can define $H(Y|X)$. Finally the *NMI* for two covers C' and C'' is given by

$$NMI(X|Y) = 1 - [H(X|Y) + H(Y|X)]/2.$$

4.3. **Clustering quality.** After experiments, we find the result is best in our method when the parameters $\mu = 2$ and $\varepsilon = 0.14$. For the other algorithms, the impact of each parameter is investigated with other parameters fixed at the typical values suggested by Xie et al. [4]. We explain our results with the four aspects below.

- Network scale: In order to validate that MR-ECOCD is suitable for network community mining of various scale, we choose a series of networks whose nodes are increased from $1k$ to $17k$, and MR-ECOCD performs well when the nodes are increasing in Figure 1 ($mu = 0.1$, $on/N = 0.3$, $< k >= 10$, $om = 2$ and N from $1k$ to $17k$).
- Degree of overlapping: To generate highly overlapping nodes, we increased the value of om from 2 to 5. It is very natural that the accuracy goes down as increasing of degree of overlapping, but MR-ECOCD goes down slowly in Figure 2 ($N = 5000$, $mu = 0.1$, $on/N = 0.3$, $< k >= 10$, and om from 2 to 5).
- Fraction of overlapping nodes: To generate high fraction of overlapping nodes, we increase on/N from 0 to 0.5. In Figure 3 ($N = 5000$, $mu = 0.1$, $om = 2$, $< k >= 5$, and on/N from 0 to 0.5), overlapping community structure is identified successfully by MR-ECOCD when there exist many overlapping nodes, e.g., 40% or 50%.
- Community density: The mixing parameter mu is the ratio of the number of inter-community links to the total number of links, and thus, it can be considered as the internal density of a community. In Figure 4 ($N = 5000$, $om = 2$, $on/N = 0.3$, $< k >= 10$, and mu from 0.1 to 0.3), the results show that our algorithm can discover communities of various internal density.

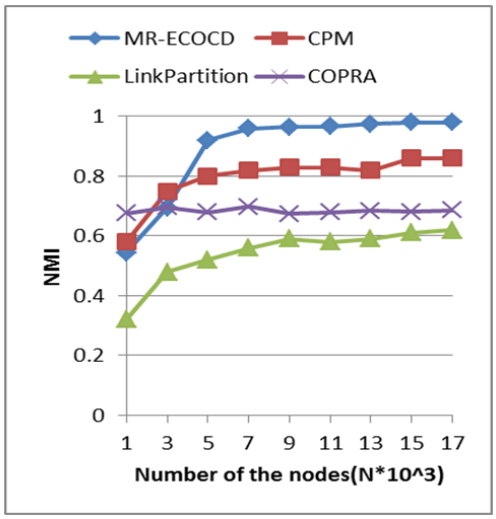

FIGURE 1. Effects of the number of node

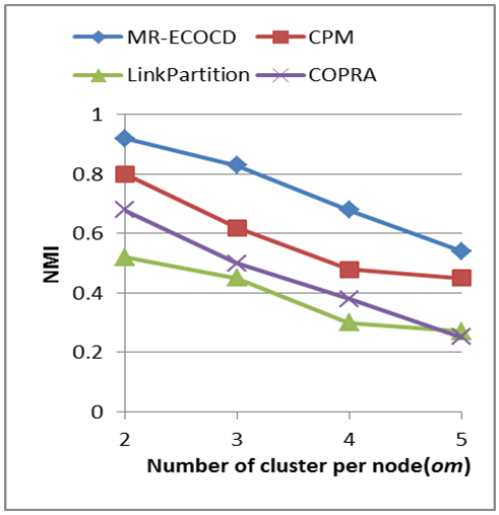

FIGURE 2. Effects of the degree of overlapping

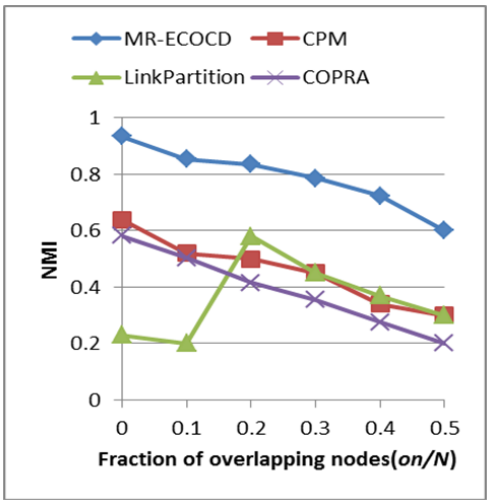

FIGURE 3. Effects of the fraction of overlapping nodes

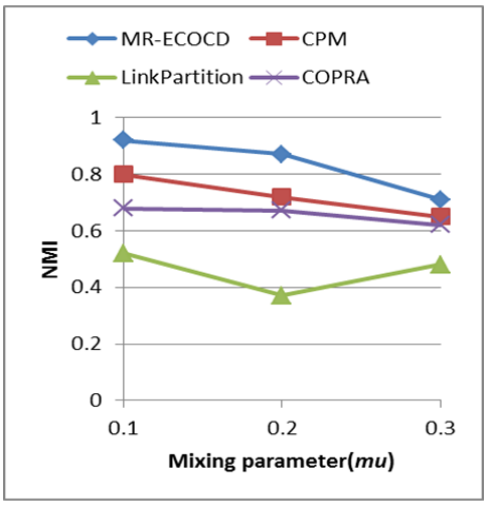

FIGURE 4. Effects of the community density

4.4. **Running time.** We test execution times of algorithm MR-ECOCD, CPM, LinkPartition and COPRA in networks whose nodes are more than $100k$. Running times of the four algorithms are depicted in Figure 5. In our experiments, when the number of nodes is greater than $100k$, the CPM is not able to finish the execution and get the final result because of the memory overflow error, so it is not shown in Figure 5. In addition to MR-ECOCD and CPM, the other algorithms are not able to finish the execution also when the node number is large enough. And it is because of the memory overflow error also. In the experiment, there are three machines running on the clustering environment for community detection for MR-ECOCD. The distributed computing architecture is apparently effective, the execution time grows slowly while the size of dataset grows from 100K to 1000K records, as shown in Figure 5. However, it is not better than some algorithms such as COPRA when the node number is small in Figure 5.

We can get the following conclusions from the analysis of the results. Using one machine to execute the traditional classic algorithms is time-consuming and requires large memory

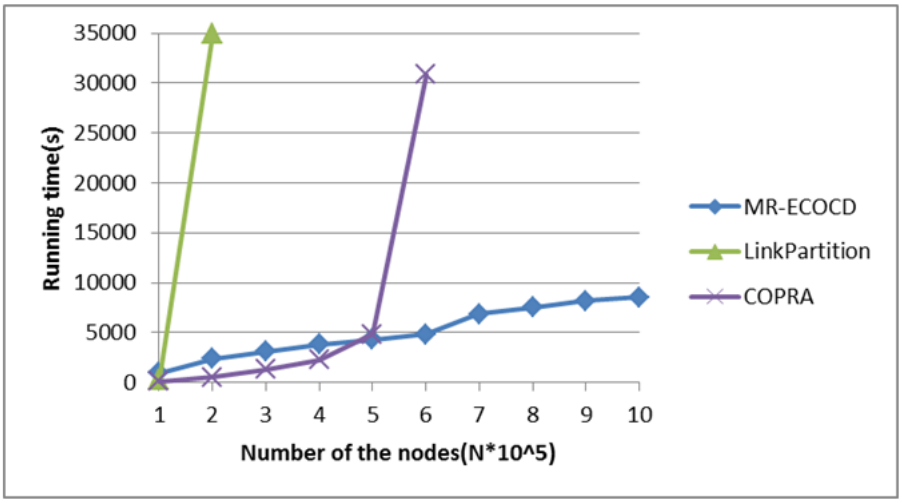

FIGURE 5. Comparison of execution time

space. As dataset sizes increase, execution time increases sharply as well. Based on the distributed computing architecture the result is effective because of the parallel and distributed execution on the clustering environment, but it is not effective when the data is small because of the configuration of environment and the large I/O operation of intermediate data on the disk.

5. **Conclusions.** In this paper, for discovering overlapping communities efficiently on large-scale network, we define *direct edge community* (DEC) and *mergeable direct edge community* (MDEC) based on edge density clustering method, and then a novel distributed computation algorithm using MapReduce framework is proposed. Our algorithm consists of four major stages: searching neighbor nodes, calculating edge similarity, marking and clustering communities, transforming edge communities. These four steps correspond to four MapReduce jobs, each job can take computing on the whole network and can be executed concurrently in a distributed environment. We implemented MR-EOCOD on Hadoop, and conducted comparative experiments on the LFR dataset. The results sufficiently show that our algorithm can not only detect the overlapping community from large-scale complex networks accurately, but also complete it within reasonable time. Future research should continue to look for methods that further reduce computation time while attaining higher quality in community structure discovery.

Acknowledgment. This work is supported by the National Natural Science Foundation of China under Grant No. 61170190, No. 61472341 and the Natural Science Foundation of Hebei Province China under Grant No. F2013203324, No. F2014203152 and No. F2015203326. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] Y. Gu, B. Zhang, G. Zou et al., Overlapping community detection in social network based on microblog user model, *2014 IEEE International Conference on Data Science and Advanced Analytics*, Shanghai, China, pp.333-339, 2014.

[2] X. Yu, J. Yang and Z. Q. Xie, A semantic overlapping community detection algorithm based on field sampling, *Expert Systems with Applications*, vol.42, no.1, pp.366-375, 2015.

[3] T. Chakraborty and A. Chakraborty, OverCite: Finding overlapping communities in citation network, *Proc. of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, Niagara Falls, ON, Canada, pp.1124-1131, 2013.

[4] J. Xie, S. Kelley and B. K. Szymanski, Overlapping community detection in networks: The state-of-the-art and comparative study, *ACM Computing Surveys*, vol.45, no.4, 2013.

[5] I. Derényi, G. Palla and T.Vicsek, Clique percolation in random networks, *Physical Review Letters*, vol.94, no.16, pp.1-4, 2005.

[6] A. Lancichinetti, S. Fortunato and J. Kertész, Detecting the overlapping and hierarchical community structure in complex networks, *New Journal of Physics*, vol.11, p.03315, 2009.

[7] D. Jin, B. Yang, C. Baquero et al., A Markov random walk under constraint for discovering overlapping communities in complex networks, *Journal of Statistical Mechanics Theory & Experiment*, vol.2011, no.1, pp.75-98, 2011.

[8] E. L. Martelot and C. Hankin, Fast multi-scale detection of overlapping communities using local criteria, *Computing*, vol.96, no.11, pp.1011-1027, 2014.

[9] S. Gregory, Finding overlapping communities in networks by label propagation, *New Journal of Physics*, vol.12, no.10, pp.2011-2024, 2010.

[10] T. S. Evans and R. Lambiotte, Line graphs, link partitions, and overlapping communities, *Physical Review E Statistical Nonlinear & Soft Matter Physics*, vol.80, no.1, pp.145-148, 2009.

[11] Y. Y. Ahn, J. P. Bagrow and S. Lehmann, Link communities reveal multiscale complexity in networks, *Nature*, vol.466, no.7307, pp.761-764, 2010.

[12] S. Lim, S. Ryu, S. Kwon et al., LinkSCAN*: Overlapping community detection using the link-space transformation, *Proc. of International Conference on Data Engineering*, Chicago, IL, The United States, pp.292-303, 2014.

[13] D. Jeffrey and G. Sanjay, MapReduce: Simplified data processing on large clusters, *Communications of the ACM*, vol.51, no.1, pp.107-113, 2004.

[14] S. Moon, J. G. Lee and M. Kang, Scalable community detection from networks by computing edge betweenness on MapReduce, *2014 International Conference on Big Data and Smart Computing*, pp.145-148, 2014.

[15] M. Ovelgonne, Distributed community detection in web-scale networks, *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp.66-73, 2013.

[16] Y. J. Su, W. L. Hsu and J. C. Wun, Overlapping community detection with a maximal clique enumeration method in MapReduce, *Advances in Intelligent Systems & Computing*, pp.367-376, 2014.

[17] X. Xu, N. Yuruk, Z. Feng et al., Scan: A structural clustering algorithm for networks, *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.824-833, 2007.

[18] A. Lancichinetti and S. Fortunato, Community detection algorithms: A comparative analysis, *Physical Review E*, vol.80, no.5, pp.2142-2152, 2009.