Local Overlapping Community Detection

LI NI, WENJIAN LUO, WENJIE ZHU, and BEI HUA, University of Science and Technology of China

Local community detection refers to finding the community that contains the given node based on local information, which becomes very meaningful when global information about the network is unavailable or expensive to acquire. Most studies on local community detection focus on finding non-overlapping communities. However, many real-world networks contain overlapping communities like social networks. Given an overlapping node that belongs to multiple communities, the problem is to find communities to which it belongs according to local information. We propose a framework for local overlapping community detection. The framework has three steps. First, find nodes in multiple communities to which the given node belongs. Second, select representative nodes from nodes obtained above, which tends to be in different communities. Third, discover the communities to which these representative nodes belong. In addition, to demonstrate the effectiveness of the framework, we implement six versions of this framework. Experimental results demonstrate that the six implementation versions outperform the other algorithms.

CCS Concepts: • **Mathematics of computing → Graph algorithms**; • **Information systems → Clustering**;

Additional Key Words and Phrases: Social network, community detection, local community detection, local overlapping community detection

ACM Reference format:
Li Ni, Wenjian Luo, Wenjie Zhu, and Bei Hua. 2019. Local Overlapping Community Detection. *ACM Trans. Knowl. Discov. Data* 14, 1, Article 3 (December 2019), 25 pages.
https://doi.org/10.1145/3361739

3

1 INTRODUCTION

Community detection often aims at dividing the entire network into several communities, ensuring that nodes in the same community are tightly connected, whereas nodes in different communities are sparsely connected. Many real-world networks contain community structure, such as protein–protein intersection networks and social networks [14, 21, 24, 36]. Analyzing the network structure and extracting information from it is an important research topic, which has attracted

This work is partially supported by National Natural Science Foundation of China under Grant No. 61573327.
Authors' addresses: L. Ni, W. Luo (corresponding author), W. Zhu, and B. Hua, School of Computer Science and Technology, Anhui Province Key Laboratory of Software Engineering in Computing and Communication, School of Computer Science and Technology, University of Science and Technology of China, Jin Zhai Road, Hefei, Anhui, 230026, China; emails: nlcs@mail.ustc.edu.cn, wjluo@ustc.edu.cn, pzwjay@mail.ustc.edu.cn, bhua@ustc.edu.cn.

1556-4681/2019/12-ART3 $15.00
https://doi.org/10.1145/3361739

much attention in the past twenty years [11, 29, 33, 34, 39, 40]. Fortunato gave a thorough exposition of community detection in [16]. Newman et al. [31] proposed the modularity to measure the quality of detected communities. Raghavan et al. proposed a simple label propagation algorithm without prior information about the communities [35].

Community detection algorithms often need global information of the network [28]. However, in real-world applications, global information of the network may be unavailable or expensive to acquire [9]. Local community detection [9, 10, 26] emerged from the view of local structure. Local community detection refers to finding the community containing the given node based on the local information rather than the global information. Some studies have focused on local community detection [1, 8, 23, 41, 46, 48]. Clauset [10] defined a local modularity, R, to measure the quality of the local community. Further, he proposed a method named R method to detect the local community. Luo et al. [26] put forward the local modularity M, which is defined as the ratio between the number of edges within the community to the number of edges that link the detected community to the remaining part of the network. Chen et al. [9] presented a local community method based on the local degree central node. Their method detects the community starting from the local degree central node associated with the given starting node. The work in [2] finds the community by expanding several shells from the starting node. The ith shell is the set of nodes from starting node at depth i and expands from the previous $(i-1)$th shell.

A similar problem to local community detection is community search. Community search aims to find a connected subgraph for a graph and a set of query nodes. Some studies on community search require global information [13, 20, 38], and some do not [12]. Sozio et al. [38] studied the problem of discovering a community with size restriction or without size restriction for a set of query nodes. The algorithms in [38] take the entire graph as input, and delete one node each time to get the final community. Cui et al. [13] described two algorithms for community search: one is a global search algorithm and the other is a local search algorithm.

As often occurs in the real world, a person belongs to multiple social circles in a social network, such as the friend circle, the family circle and the colleague circle [30]. In the network, nodes that belong to multiple communities are called overlapping nodes. Many real-world networks contain overlapping communities like social networks [3]. So far, local overlapping community detection has not been paid much attention. (α, γ)-*OCS* model [12] was designed for local overlapping community detection. (α, γ)-*OCS* model is based on the α-adjacency-γ-quasi-k-clique. The γ-quasi-k-clique is a graph that contains k nodes and at least $\gamma \frac{k(k-1)}{2}$ edges. The α-adjacency-γ-quasi-k-clique component is the union of all γ-quasi-k-cliques that share at least α nodes. However, parameters of (α, γ)-*OCS* model need to be set, which is difficult when the network is unknown. If the γ-quasi-k-clique is sparser than the network, the size of discovered communities may be larger than that of real communities. If the γ-quasi-k-clique is denser than the network, the size of discovered communities may be smaller than that of real communities, or no community can be found. The latter case means the node and its neighbors cannot form an γ-quasi-k-clique.

In this article, we propose a framework for local overlapping community detection. The main contributions are given as follows.

- —We propose a framework for local overlapping community detection. It contains three main steps. First, find nodes in multiple communities to which the given node belongs. Second, select representative nodes from nodes obtained above, which tends to be in different communities. Third, discover the communities to which these representative nodes belong.
- —We implement six versions of this framework. These implementations are tested on both synthetic and real-world networks. The experimental results show the six implementations of the framework outperform other approaches.

The rest of this article is organized as follows. In Section 2, some related methods are introduced. In Section 3, we introduce the problem of local overlapping community detection and some methods used in our framework. The framework is introduced in Section 4. Implementations of the framework are described in Section 5. In Section 6, experiments and comparisons are given. In Section 7, the number of nodes that no community is found by different algorithms is discussed. Further, the effect of the local community algorithm and the effect of parameter selection are also discussed. Section 8 provides the conclusion.

2 RELATED WORK

2.1 Overlapping Community Detection

Overlapping is one of the primary characteristics of social networks. Many real-world networks contain overlapping communities like social networks [3]. Many literatures have studied on overlapping community detection [4–6, 15, 32, 37, 42, 44, 47]. Bai et al. [3] developed an excellent overlapping community detection algorithm named *OCDDP* based on density peaks. They set distance matrix by a similarity based method first, and then select cores of communities. Finally, all other nodes are allocated to different communities. Gregory et al. [17] proposed *COPRA* based on the work in [35]. In *COPRA*, multiple communities information needs to be kept in the label and propagation steps. Xie et al. presented SLPA for overlapping community detection, which spreads labels according to dynamic interaction rules [43].

However, in real-world applications, the network may be incomplete or expensive to acquire the whole one. In this situation, local overlapping community detection becomes meaningful. Besides, comparing with overlapping community detection algorithms that focus on the global structure of the network, the computational cost of the local overlapping community detection algorithm is greatly reduced.

2.2 Local Community Detection

Since global information of the network may be unavailable or expensive to acquire [9], local community detection [9, 10, 26] emerged from the view of local structure. The task of local community detection is to find the community that contains the given node according to the local information rather than the global information. There have been some studies on local community detection [1, 8, 23, 41, 48].

Typical local modularities, R and M, were proposed by Clauset [10] and Luo et al. [26], respectively. Based on the modularities R and M, R method and the M method were correspondingly proposed. According to the characteristics of the formation of local communities, Luo et al. [28] proposed two local community detection algorithms, *DMF_M* and *DMF_R*.

For the higher order structures of networks and directed networks, Yin et al. [46] proposed the MAPPR algorithm based on network motifs. The goal is to find the community that gets minimum motif conductance and contains the given node. The motif conductance is a generalization of conductance. Chen et al. [9] presented a local community method based on the local degree central node. Their method detects the community starting from the local degree central node instead of the given starting node.

For better understanding the difference between local community detection and traditional community detection, here, we take community detection algorithm *CDFR* (Section 3.3) and local community detection method M (Section 3.2) as examples. In *CDFR* algorithm, the first step is to calculate the centrality and fuzzy relation of all nodes in the network G. All nodes in G are accessed, so the algorithm takes advantage of global information. The M method only accesses the nodes in C and N in Figure 1. That is, only local information is utilized.

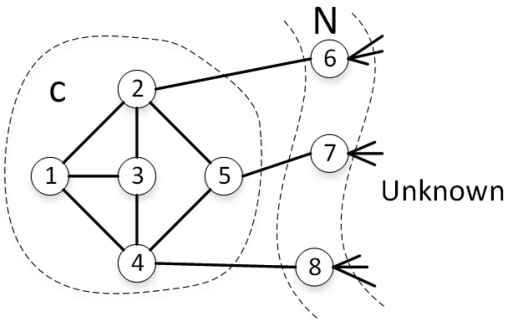


Fig. 1. The display of local community. C is the local community and N is the neighbor-node set of the local community C.

2.3 Local Overlapping Community Detection

So far, local overlapping community detection has not been paid much attention. In the following, we briefly review some local overlapping community detection methods. Yang et al. investigated two tasks: one finds a community to which the given node belongs, and the other finds all communities that contains the given node [45]. First, for each node, compute PageRank scores r_u with PageRank-Nibble random walk method and sort nodes in descending order according to r_u. Then, compute community score $f(S_k)$, where set S_k consists of top k nodes. The set S_k^* is the detected local community where $f(S_k^*)$ gets the first local minimum. Multiple communities that contain the given node correspond to multiple local minima. It can be seen that the discovered multiple communities are nested and the number of communities acquired needs to be set.

Kamuhanda et al. adopted Non-negative Matrix Factorization (NMF) algorithm to address multiple local community detection [22]. They first extract a subgraph using Breadth-First Search (BFS). Two or three levels of BFS are required for dense networks while more levels are required for sparse networks. Then they estimate the number of communities by NMF and use NMF to detect communities. Finally, final communities are obtained by adding neighbors of the initial community to the community. He et al. provided a method called *LOSP* to address local community detection [18]. Same as the work in [22], the first step of *LOSP* also extracts a subgraph from the network. Further, the local overlapping community detection algorithm is further proposed by using *LOSP* as a subroutine. Methods in [22] and [18] extract a subgraph from the network in advance, in which the number of layers of the subgraph needs to be set. Unlike these two algorithms, our work does not extract a subgraph from the network in advance.

Hollocou et al. proposed a method named MULTICOM to discover multiple local communities [19]. MULTICOM finds multiple local communities by iteratively finding new seeds S_{new} (S_{new} is initialized with the initial seed set) and obtaining local communities based on S_{new}. It involves three main steps. The first step uses the local algorithm to discover a community for each node $s \in S_{new}$. The second step adopts the scoring function to map each node in the graph to a vector and obtained vectors are clustered by DBSCAN algorithm to get multiple clusters. The third step picks new seeds from obtained clusters.

The difference between the MULTICOM and our work includes the following three points. First, MULTICOM requires multiple iterations of two operations, i.e., picking new seeds and discovering communities for new seeds, while our algorithms do not require similar iterations. Second, MULTICOM requires two parameters, the number of communities and the parameter that controls the number of new seeds produced in each iteration. These two parameters are difficult to set

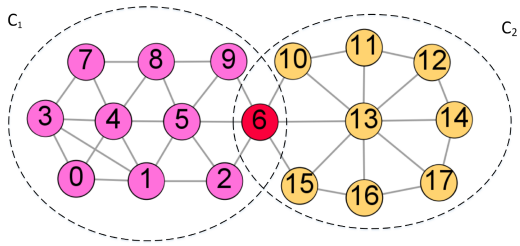


Fig. 2. An example of the small network. C_1 and C_2 are communities that marked with the dotted circle. Node 6 is an overlapping node which belongs to C_1 and C_2. The remaining nodes are non-overlapping nodes.

without prior knowledge. In our work, the parameter values are fixed. Third, compared with MULTICOM, the idea of our work is simpler. Most importantly, the experimental results in Section 6 demonstrate that our algorithms are much better than MULTICOM.

2.4 Community Search Problem

Community search aims to find a connected subgraph for a graph and a set of query nodes. It does not limit the need for global information. Some algorithms for community search require global information [13, 20, 38], and some do not [12].

Among the algorithms for community search, only [12] and [20] were designed for local overlapping community detection. Cui et al. [12] proposed a novel community model named (α, γ)-*OCS* model based on the α-adjacency-γ-quasi-k-clique. The γ-quasi-k-clique is a graph that contains k nodes and at least $\gamma \frac{k(k-1)}{2}$ edges. Two γ-quasi-k-cliques are α-adjacency if they share at least α nodes. The α-adjacency-γ-quasi-k-clique component is the union of all α-adjacency γ-quasi-k-cliques. Each α-adjacency-γ-quasi-k-clique component is regarded as a community. For a given node v_0, the task is to find all α-adjacency-γ-quasi-k-clique components containing v_0.

Based on the k-truss concept, Huang et al. [20] proposed k-truss model to detect overlapping communities to which the given node belongs. The k-truss is the largest subgraph with at least k-2 triangles containing each edge within the subgraph. Because k-truss model needs global information, our algorithms do not compare with it in the experiments.

3 PRELIMINARIES

We first introduce the problem of local overlapping community detection, then review two local community detection algorithms, *M* and *DMF_M*. Also, we introduce an efficient community detection algorithm named *CDFR*.

3.1 Problem Statement

For a graph G and a given node v_0, the objective of local overlapping community detection is to find all communities that contain v_0 according to local information. As shown in Figure 2, node 6 is an overlapping node that belongs to C_1 and C_2. For node 6, the task is to find communities C_1 and C_2 only with local information.

3.2 *M* and *DMF_M*

Here, we introduce two typical methods, *M* [26] and *DMF_M* [28]. *M* method is a local community detection algorithm based on typical local modularity *M*, proposed by Luo et al. [26]. *M* is defined as

$$M = \frac{e_{inner}}{e_{outer}},$$

ALGORITHM 1: M method

Input: v_0: the given node, G: the network
Output: C: the local community that v_0 belongs to

1: $C = \{v_0\}$
2: put the neighbors of v_0 into N
3: $M = 0$
4: **while** True **do**
5: $M_{max} = 0$
6: **for** each v in N **do**
7: compute M_v for $C \cup \{v\}$
8: **if** $M_v > M_{max}$ **then**
9: $M_{max} = M_v$
10: $vbest = v$
11: **end if**
12: **end for**
13: **if** $M_{max} \geq M$ **then**
14: $M = M_{max}$
15: $C = C \cup \{vbest\}$
16: $N = (N - \{vbest\}) \cup N_{vbest}$
17: **else**
18: break
19: **end if**
20: **end while**

where e_{inner} represents the number of edges within C and e_{outer} represents the number of edges between C and N. The meanings of C and N are shown in Figure 1.

M method is shown in Algorithm 1. First, initialize C, N, and M (Steps 1–3). Then, we select node $vbest$ that has maximum value of M_v (recorded as M_{max}) from N, where M_v is the M value if v is added to C (Steps 5–12). Next, add $vbest$ to C and update M and N if $M_{max} \geq M$ (Steps 13–16). There are two operations to update N. The first operation removes $vbest$ from N. The second operation adds nodes in N_{vbest} to N, where N_{vbest} is composed of $vbest$'s neighbors that are not in N and C.

According to the characteristics of the formation of local communities, two local community detection algorithms, *DMF_M* and *DMF_R*, were proposed in [28]. The work in [28] demonstrates that *DMF_M* method is better than *DMF_R* method.

We briefly review *DMF_M* shown in Algorithm 2. The process of community formation is divided into the following three stages: the initial stage, the middle stage, and the closing stage. In the initial stage (Steps 4–22), the number of external edges is greater than the number of internal edges, which means $M < 1$. At this stage, for each node v in N, put the node with $M_v - M \geq 0$ to $deltaMlargerzero$, where M_v is the M value if v is added to C. Here, the meanings of C and N are shown in Figure 1. Then, select node $vbest$ from $deltaMlargerzero$, and update C, N, and M. $vbest$ is the node that gets maximal value of $\omega_1(v_i)$, defined as

$$\omega_1(v_i) = \begin{cases} \frac{\max\limits_{v_j \in N_{ci}} \left(\frac{|N(v_i) \cap N(v_j)|+1}{|N(v_j)|} + 0.1 * \frac{|N_2(v_i) \cap N_2(v_j)|+1}{|N_2(v_j)|} \right)}{1.1} & \Delta M \geq 0 \\ 0, & \Delta M < 0, \end{cases}$$

where $N(v_i)$ is the set of neighbors of node v_i, $N_2(v_i) = \{x | y \in N(v_i), x \in N(y)\}$ and N_{ci} is the intersection of $N(v_i)$ and C.

ALGORITHM 2: *DMF_M*

Input: v_0: the given node, G: the network
Output: C: the local community that v_0 belongs to

```
1: C = {v0}
2: put the neighbors of v0 into N
3: M = 0
   //the initial stage
4: while True do
5:     Mmax = −1, deltaMlargerzero = ∅
6:     if M < 1 then
7:         for each v in N do
8:             add v to deltaMlargerzero when Mv − M ≥ 0
9:         end for
10:        if deltaMlargerzero! = ∅ then
11:            select node vbest from deltaMlargerzero
12:            Mmax = Mvbest
13:        else
14:            Mmax = −1
15:        end if
16:    end if
17:    if Mmax ≥ M then
18:        update C, M, N
19:    else if Mmax < M or M ≥ 1 then
20:        break
21:    end if
22: end while
   //the middle stage
23: steps 4-20 in Algorithm 1
   //the closing stage
24: add nodes that are closely related to nodes in C to C
```

As nodes added to the community, it enters the middle stage when $M \geq 1$ or no node with $M_v - M \geq 0$. The steps in middle stage (Step 23) are the same as Steps 4–20 in Algorithm 1. The task is to add the node that maximizes M to C. When the middle stage is over, the community is basically formed. At the closing stage, take node v_i from N and remove it from N. If $\omega_3(v_i)$ >0.5, add v_i to C and update N. Repeat the above steps until N is empty. By the way, $\omega_3(v_i)$ is defined as

$$\omega_3(v_i) = \max\left\{\max_{v_j \in N_{ci}} \frac{|N(v_i) \cap N(v_j)| + 1}{N(v_i)}, \frac{|N(v_i) \cap C|}{|N(v_i)|}\right\}.$$

3.3 *CDFR*

CDFR [27] is an efficient community detection algorithm. In *CDFR*, fuzzy relation is adopted to measure the followership or dependence relation between nodes. The *centrality* of the node represents the importance or the influence of the node. The idea of *CDFR* is that the node is in the same community as its Nearest node with Greater Centrality (abbreviated as *NGC* node) if the fuzzy relation from the node to its *NGC* is large enough. The node is a leader of one community when the fuzzy relation from it to its *NGC* node is small.

Next, we introduce the *centrality* and the fuzzy relation. The *centrality* of node v is defined as

$$centrality(v) = deg(v) + \sum_{u \in \Gamma(v)} \left(deg(u) + \sum_{w \in \Gamma(u)} deg(w) \right), \tag{1}$$

where $deg(u)$ is the degree of u and $\Gamma(v)$ represents the set of neighbors of v.

The fuzzy relation from v to the *NGC* node of v (i.e., $NGC(v)$) measures the followership or dependence relation between them. Suppose all paths from v to $NGC(v)$ consists of P, and one path in P can be expressed as $p = \{s_1, s_2, \ldots, s_k\}$, where s_i is the node in p, $s_1 = v$ and $s_k = NGC(v)$. The fuzzy relation from v to $NGC(v)$ is defined as

$$R(v, NGC(v)) = \max_{p \in P} \{\mu_p(s_1, s_k)\}, \tag{2}$$

and $\mu_p(s_1, s_k)$ is calculated as

$$\mu_p(s_1, s_k) = t(\mu_p(s_1, s_{k-1}), \mu_p(s_{k-1}, s_k)),$$

$$\mu_p(s_{k-1}, s_k) = \frac{1 + |\Gamma(s_{k-1}) \cap \Gamma(s_k)|}{|\Gamma(s_{k-1})|},$$

where $\Gamma(s)$ represents the set of neighbors of s; $t[.]$ means a kind of t-norms, which is calculated as follows:

$$t(x, y) = x * y.$$

Due to community centers with large *centrality* and small fuzzy relation, the decision graph is adopted to identify community centers. In decision graph, the horizontal axis represents *centrality*, and the vertical axis represents the fuzzy relation from a node to its *NGC* node. Thus, the community centers are in the lower region of the decision graph, and a gap is expected to exist between community centers and other nodes. To enlarge the gap between community centers and other nodes, *CDFR* makes further adjustments to the fuzzy relation by the rate of neighbors that pass through the node while finding the *NGC* nodes recursively.

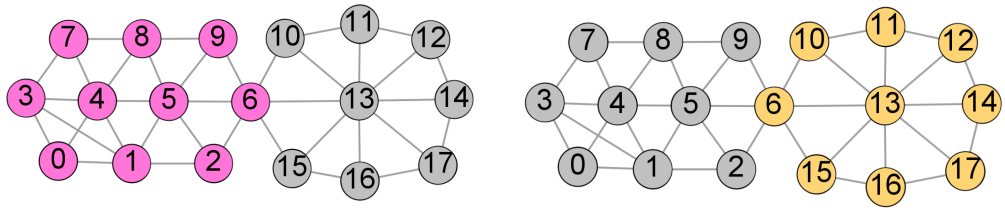
Based on the concepts explained above, the process of *CDFR* is given as follows. Calculate the *centrality* of each node first. Then, for each node v, find $NGC(v)$ and compute the fuzzy relation from v to $NGC(v)$. Next, draw the decision graph, and set the fuzzy relation threshold δ according to the gap in decision graph. Finally, assign each node v to the same community as $NGC(v)$ if the fuzzy relation is greater than or equal to δ. Otherwise v is considered a community center of a new community.

4 THE PROPOSED FRAMEWORK

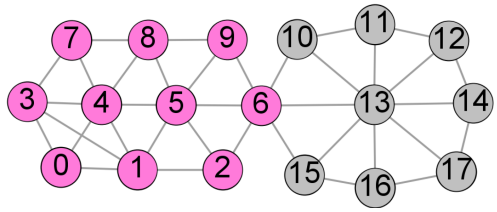
4.1 An Example

First, we analyze the accuracy of communities obtained by the local community detection algorithm for overlapping nodes and non-overlapping nodes. For the sake of understanding, we will give some examples. These examples are based on the network in Figure 2. Figure 3 shows the communities obtained by the M method with nodes 5, 6, and 13 in Figure 2 as the starting nodes, respectively.

The idea of some local community detection algorithms is finding nodes that are closely connected to nodes in the currently discovered community, and then adding these nodes to the community. Because non-overlapping nodes are only closely connected to nodes of the same community, and sparsely connected to nodes in other communities, the community obtained with the non-overlapping node as the starting node is relatively accurate. Taking node 5 in Figure 2 as an example, Figure 3(a) shows the community discovered by M method for node 5 is accurate.



(a) Local community found by M method for node 5. (b) Local community found by M method for node 13.



(c) Local community found by M method for node 6.

Fig. 3. Local communities found by M method with nodes 5, 6, and 13 as starting nodes, respectively. The found local communities marked with color.

The overlapping nodes are closely connected to the multiple communities in which they are located. Therefore, for overlapping nodes, the community detected using the local community detection algorithm may be the following two cases:

(1) One of the multiple communities to which the given overlapping node belongs is discovered. Taking node 6 in Figure 2 as an example, node 6 is an overlapping node that belongs to C_1 and C_2. Figure 3(c) shows the community discovered by M method for node 6, only C_1 obtained.
(2) The detected community contains nodes from multiple communities. Because overlapping nodes are tightly connected to nodes in multiple communities, it is understandable that the detected community contains nodes in multiple communities.

From the above analysis, we can see that the community detected by the local community detection algorithm with the overlapping node as the starting node is probably inaccurate and only one community can be obtained. The communities discovered by the local community detection algorithm for the non-overlapping nodes are relatively accurate. Assume that the given overlapping node is v_0. Inspired by this conclusion, we find a way for local overlapping community detection. That is, we first find the non-overlapping nodes (called representative nodes) in the same community as node v_0, and then use these nodes to discover the communities to which v_0 belongs.

We use an example to further illustrate our ideas.

Example 1. Consider the network shown in Figure 2. Suppose that we need to find local communities that contain node 6 and select the M method to discover the local community.

We first find nodes in the same communities as node 6, such as node 5 in C_1 and node 13 in C_2. Then, M method is performed with nodes 5 and 13 as the starting nodes, respectively. Figure 3(a) and 3(b) shows the community discovered by M method for node 5 is C_1 and the community discovered by M method for node 13 is C_2. The detected communities C_1 and C_2 are treated as

ALGORITHM 3: The framework for local overlapping community detection

Input: v_0: the given node
Output: LC: multiple communities that v_0 belongs to
1: $LC = \emptyset$
2: $Lcs = \emptyset$
3: construct *candidates* (Section 5.1)
4: select *seeds* from *candidates* (Section 5.2)
5: **for** each u in *seeds* **do**
6: $C_i = LCD(u)$ (Section 5.3)
7: **if** v_0 in C_i **then**
8: $LC.add(C_i)$
9: **end if**
10: $Lcs.add(C_i)$
11: **end for**
12: Post-process (Section 5.4)

communities to which node 6 belongs. In short, we can obtain C_1 and C_2 containing node 6 through non-overlapping nodes 5 and 13 that are in the same communities as node 6.

Nodes in the same community as v_0 are unknown. So, the first step is to find nodes in multiple communities to which v_0 belongs. Intuitively, nodes closely connected to v_0 or in the vicinity of v_0 may be in the same community as v_0. Therefore, nodes closely connected to v_0 or in the vicinity of v_0 make up the set of candidate nodes.

However, the set of candidate nodes may contain some boundary nodes. As pointed out in Refs. [28] and [9], the communities discovered with boundary nodes as the starting nodes may be inaccurate. Besides, ideally, representative nodes contain one node in each community that contains v_0. Therefore, we select community centers or nodes close to the community center from the set of candidate nodes and these selected nodes are called representative nodes. Finally, local community detection is performed with each representative node as the starting node.

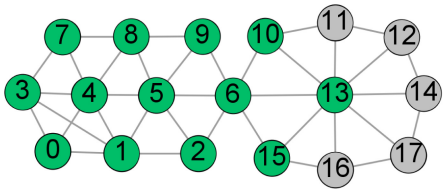
4.2 The Proposed Framework

Based on the analysis in Section 4.1, we propose the framework for local overlapping community detection, shown in Algorithm 3.

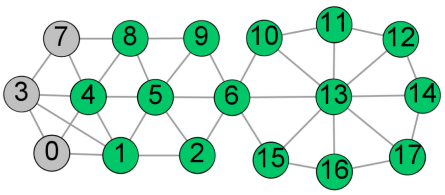
The process of framework is as follows. First, construct *candidates* that is the set of candidate nodes for representative nodes (Section 5.1). Then, select representative nodes (denoted by *seeds*) from *candidates* (Section 5.2). Next, discover the local community with each node in *seeds* as a starting node by the local community detection algorithm (Section 5.3), denoted by LCD in Algorithm 3. Afterwards, screen the communities by only adding communities that contain v_0 to LC (Steps 7–9 in Algorithm 3). Nodes that are not in the same community as v_0 may be mixed in *seeds*. This step is used to exclude communities with these nodes as the starting nodes. In addition, all acquired communities are added to Lcs for post-processing. Finally, we post-process the obtained communities (Section 5.4).

5 IMPLEMENTATIONS

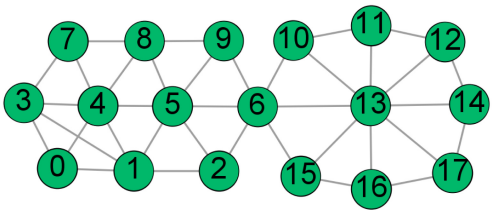
In this section, we implement the proposed framework. We first give three methods to construct *candidates*. Then, we introduce one method to select representative nodes. Next, we introduce two local community detection algorithms. Finally, we post-process communities obtained.



(a) *candidates* obtained by C_0N.



(b) *candidates* obtained by *neighK*.



(c) *candidates* obtained by C_0N+*nieghK*.

Fig. 4. The display of *candidates* for node 6.

5.1 Constructing *candidates*

As mentioned earlier, *candidates* is the set of candidate nodes, ideally including nodes in multiple communities to which the given node belongs. We provide three methods to construct *candidates*, namely C_0N, *nieghK*, and C_0N+*neighK*.

5.1.1 C_0N. The v_0 and nodes in the same community as v_0 are closely related. Here, we use local community detection algorithm to find these nodes.

Suppose C_0 is the community found by the local community detection algorithm (Section 5.3) for v_0, and $N(C_0)$ is the neighbor-node set of the local community C_0. C_0N method is to construct *candidates* by the union of C_0 and $N(C_0)$ (denoted by $C_0 \cup N(C_0)$). Assume that v_0 belongs to multiple communities C_i ($1 \leq i \leq k, |C_i| \geq 2$). Note that C_i has some nodes in $C_0 \cup N(C_0)$. Next, we will prove it.

Remark 1. *At least two nodes in C_i ($1 \leq i \leq k, |C_i| \geq 2$) are contained in $C_0 \cup N(C_0)$.*

Proof. For any C_i, $v_0 \in C_i$ is obviously established. Assume that all neighbors of v_0 do not belong to C_i. Then, C_i and v_0 are disconnected, which means that C_i does not contain v_0. This contradicts $v_0 \in C_i$. Therefore, at least one neighbor of v_0 belongs to C_i. Since v_0 and all its neighbors belong to $C_0 \cup N(C_0)$, $C_0 \cup N(C_0)$ contains at least two nodes in C_i.

The remark 1 states that *candidates* by C_0N contains at least two nodes in C_i. □

Example 2. Continue with the example 1. Figure 3(c) shows C_0 of node 6 by M method, which is marked in rose red. Figure 4(a) shows *candidates* by C_0N of node 6, which is marked in green.

5.1.2 *neighK*. Nodes in multiple communities to which v_0 belongs should be in the vicinity of v_0. The intuitive idea is that nodes in k-hop-neighborhood of v_0 make up *candidates*.

Ideally, *candidates* contains community centers or nodes close to the community centers that are in the same community as v_0. The shortest path length between the boundary node and the community center or node close to the community center is usually less than half of the diameter of the community. The diameter of community is the largest shortest path between nodes in the

community. Assume that k is half of the diameter of the community. Even if v_0 is a boundary node, k-hop-neighborhood of v_0 can also contain community centers or nodes close to the community centers.

Therefore, k is set to half of the diameter of the community. We consider the diameter of C_0 (denoted by $diameter(C_0)$) as the diameter of the community. Then, $k = \lceil 0.5 * diameter(C_0) \rceil$.

The value of k is at least 1, and all neighbors of v_0 are in *candidates* by *neighK*. So, *candidates* by *neighK* contains at least two nodes in C_i ($1 \leq i \leq k, |C_i| \geq 2$), where C_i is the community that contains v_0.

Example 3. Continue with the example 1. Figure 3(c) shows the diameter of C_0 is 3. $k = \lceil 0.5 * 3 \rceil = 2$. Figure 4(b) shows *candidates* by *neighK* of node 6, which is marked in green.

5.1.3 C_0N+*neighK*. For node 6, Figure 4(a) shows *candidates* by C_0N does not include nodes 11, 12, 14, 16, and 17 and Figure 4(b) shows *candidates* by *neighK* does not include nodes 0, 3, and 7. Nodes 11, 12, 14, 16, 17, 0, 3, and 7 are in two different communities that contains node 6. These nodes can also be added to *candidates*, so the union of *candidates* by C_0N and *candidates* by *neighK* is as a third method to obtain *candidates*, named C_0N+*neighK*.

Obviously, *candidates* by C_0N+*neighK* contains at least two nodes in C_i ($1 \leq i \leq k, |C_i| \geq 2$).

Example 4. Continue with the example 1. Figure 4(c) shows *candidates* by C_0N+*neighK* of node 6, which is marked in green.

5.2 Selecting Representative Nodes from *candidates*

Ideally, representative nodes contain one node in each community that contains the given node. We use the fuzzy relations between nodes and their *NGC* nodes to select these representative nodes. The *NGC* node and *centrality* of a node are introduced in Section 3.3 [27].

The fuzzy relation from node v to $NGC(v)$ (denoted as $R(v, NGC(v))$) indicates the likelihood that node v and $NGC(v)$ are in the same community. If $R(v, NGC(v)) \geq 0.5$, any adjacent two nodes s_i and s_{i+1} on the path from node v to $NGC(v)$ satisfy that the number of shared neighbors of nodes s_i and s_{i+1} is greater than or equal to half of the number of neighbors of s_i. That is, the connections between adjacent nodes are close.

Node v and $NGC(v)$ closely connected should be in the same community, so node v does not need to be a representative node. Therefore, nodes with fuzzy relations greater than or equal to 0.5 in *candidates* are not representative nodes. Here, nodes whose fuzzy relations (to their *NGC* nodes) are less than 0.5 are selected as the representative nodes.

The process of selecting representative nodes is given as follows. First, we calculate the *centrality* for each node in *candidates* by (1). Then, find *NGC* node in *candidates* and compute the corresponding fuzzy relation by (2) for each node in *candidates*. By the way, the adjustment method in [27] is adopted when calculating the fuzzy relation. The fuzzy relation of the node with the maximum *centrality* in *candidates* is set to 0 because this node has no *NGC* in candidates. Finally, select representative nodes with fuzzy relations (to their *NGC* nodes) less than 0.5.

Selecting representative nodes only use local information. The two key steps are computing the *centrality* of the node and finding *NGC* node for the node. The computation of *centrality* only requires local information. We find *NGC* node in *candidates* for each node instead of finding *NGC* node in the global network, so the process of finding *NGC* node is also local.

Example 5. Continue with the examples 2, 3, and 4. Select nodes with fuzzy relations less than 0.5. The same *seeds* are obtained from examples 2, 3, and 4. The *seeds* include nodes 5 and 13, which are in C_1 and C_2, respectively.

5.3 Local Community Detection

Two places use local community detection algorithms. The first one is used to construct *candidates* in step 3 in Algorithm 3, and the second is step 6 in Algorithm 3. We adopt *DMF_M* and *M* methods introduced in Section 3.2 to discover the local community.

Example 6. Continue with the example 5. Discover the local community for each node in *seeds*. The communities obtained by *M* method for nodes 13 and 5 are {13, 16, 15, 17, 14, 12, 11, 10, 6} and {5, 2, 1, 4, 0, 3, 7, 8, 9, 6}, respectively. These two communities are exactly the same as the real communities.

5.4 Post-process

The following two steps are for the acquired communities in *LC*:

(1) *LC* is empty. The communities discovered may not contain v_0 even if nodes in *seeds* are in multiple communities to which v_0 belongs. In this situation, v_0 may be an overlapping node with routing characteristics. Assume that $Lcs = \{C_1, \ldots, C_{|Lcs|}\}$ and $maxNeighNum$=$\max_{1 \le i \le |Lcs|} |N(v_0) \cap C_i|$, where $N(v_0)$ represents the set of neighbors of v_0 and $|N(v_0) \cap C_i|$ represents the size of intersection of $N(v_0)$ and C_i. If $maxNeighNum > 0$ and $\frac{|N(v_0) \cap C_i|}{maxNeighNum} \ge 0.5$, $C_i \cup \{v_0\}$ is added to *LC*. If *LC* is still an empty set after this operation, C_0 is regard as the community to which v_0 belongs and add C_0 to *LC*.

(2) *LC* contains duplicated communities. When multiple nodes in one community are in *seeds*, this community may be found multiple times by different nodes. In this situation, remove the duplicated communities.

5.5 Six Algorithms

We implement six versions of framework, and name them by the method for local community detection and the method of constructing *candidates*, like the following.

*LOCD*1: *DMF_M*+C_0N
*LOCD*2: *DMF_M*+*neighK*
*LOCD*3: *DMF_M*+C_0N+*neighK*
*LOCD*4: *M*+C_0N
*LOCD*5: *M*+*neighK*
*LOCD*6: *M*+C_0N+*neighK*

All the above algorithms use fuzzy relations from nodes to their *NGC* nodes to select representative nodes from *candidates*.

6 EXPERIMENTS

In this section, *LOCD*1, *LOCD*2, *LOCD*3, *LOCD*4, *LOCD*5, and *LOCD*6 are tested on both synthetic and real-world networks. We first introduce parameters settings of our algorithms and comparison algorithms. Then, we introduce the datasets and evaluations. Next, we validate the effectiveness of our algorithms by experiments.

6.1 Parameters Settings

Our algorithms are compared with MULTICOM [19] and (α, γ)-*OCS* model ($\alpha = k - 1$) [12]. The code of MULTICOM is obtained from the github website (https://github.com/ahollocou/multicom). The parameters of MULTICOM is set to the default parameters in the code. For (α, γ)-*OCS* model, we implement its process referring to [12]. The parameters of *OCS* model follow the experimental

Table 1. Parameters for the LFR Synthetic Networks

Parameter	Description	Value
n	Number of nodes	1,000
mu	Mixing parameter	{0.1, 0.3, 0.4}
k	Average degree	10
$kmax$	Maximum degree	50
$\|C\|_{min}$	Minimum for community sizes	20
$\|C\|_{max}$	Maximum for community sizes	100
t_1	Minus exponent for the degree sequence	2
t_2	Minus exponent for the community size distribution	1
om	Number of memberships of the overlapping nodes	$\{2, 3, \ldots, 8\}$
on	Number of overlapping nodes	100

setting in [12]. Specifically, $k = 3, \gamma = 1$; $k = 4, \gamma = 0.8$; $k = 4, \gamma = 0.9$, and algorithms using these parameters are expressed as (2, 1)-*OCS*, (3, 0.8)-*OCS* and (3, 0.9)-*OCS*, respectively. For convenience, (2, 1)-*OCS*, (3, 0.8)-*OCS*, and (3, 0.9)-*OCS* are called *OCS* algorithms.

6.2 Datasets

Synthetic networks are generated by the LFR benchmark [25] to evaluate the performance of local overlapping community detection algorithm. The network topology is controlled by several parameters, such as mixing parameter *mu* affecting community structure. The meaning of the parameters and parameters settings used in the experiments are shown in Table 1. Specifically, we use the LFR benchmark to generate LFR networks with mixing parameter $mu = 0.1$, $mu = 0.3$, and $mu = 0.4$, respectively. For each *mu*, networks with *om* from 2 to 8 are generated, so 21 networks are generated in total.

The Amazon dataset [45], a real-word network with ground truth, is adopted in the experiments, which is available from Stanford Network Analysis Project. The Amazon dataset contains 75,149 communities, 334,863 nodes, and 925,872 edges. Nodes represent products and an edge exists between two nodes if two corresponding products are often purchased together by customers.

6.3 Evaluations

In order to test the performance of the algorithm, *recall*, *precision*, and $fscore$ are adopted [7, 9]. Suppose that $C = \{c_1, c_2, \ldots, c_I\}$ is the set of found communities and $G = \{g_1, g_2, \ldots, g_J\}$ is the set of ground truth communities. The number of found communities and ground truth communities may be more than one, so *recall* and *precision* are slightly changed to apply to multiple communities. Since the corresponding relation between the found communities and the ground truth communities is unknown, we choose the best match for each community when calculating the *recall* and *precision*. The specific definitions of *recall*, *precision*, and $fscore$ are as follows:

$$precision = \frac{1}{I}\sum_{i=1}^{I}\frac{|c_i \cap g_j|}{|c_i|}, \quad g_j \ is \ the \ best \ match \ for \ c_i, \tag{3}$$

where g_j is the best match for c_i meaning $\forall t \in \{1, 2, \ldots, J\}$, $\frac{|c_i \cap g_j|}{|c_i|} \geq \frac{|c_i \cap g_t|}{|c_i|}$.

If the found community c_i does not correspond to any ground truth community, $\frac{|c_i \cap g_j|}{|c_i|}$ is low, so the value of *precision* will be pulled down.

Table 2. The Experimental Results on LFR Datasets with $mu = 0.1$

om		*LOCD1*	*LOCD2*	*LOCD3*	*LOCD4*	*LOCD5*	*LOCD6*	*(2,1)-OCS*	*(3,0.8)-OCS*	*(3,0.9)-OCS*	MULTICOM
2	*precision*	0.7466	0.7711	0.7457	0.9705	0.9680	**0.9774**	0.7493	0.6778	0.6740	0.1495
	recall	0.9363	0.9376	**0.9392**	0.9049	0.9078	0.9104	0.7188	0.7553	0.3072	0.6460
	fscore	0.8118	0.8275	0.8128	0.9299	0.9301	**0.9368**	0.7007	0.6707	0.4116	0.2380
3	*precision*	0.8367	0.8356	0.8302	0.9250	0.9342	**0.9352**	0.4995	0.4239	0.6337	0.1565
	recall	0.7812	**0.7867**	0.7859	0.7342	0.7427	0.7483	0.6502	0.6824	0.2746	0.5625
	fscore	0.7916	0.7944	0.7910	0.8072	0.8174	**0.8220**	0.4867	0.4376	0.3708	0.2373
4	*precision*	0.8979	0.9192	0.8983	0.9534	0.9597	**0.9644**	0.6435	0.6058	0.6595	0.1566
	recall	0.6915	0.7024	**0.7033**	0.6849	0.6960	0.6979	0.6091	0.6331	0.2735	0.5214
	fscore	0.7673	0.7847	0.7757	0.7890	0.7998	**0.8038**	0.5660	0.5493	0.3764	0.2329
5	*precision*	0.9415	0.9515	0.9407	0.9517	0.9626	**0.9674**	0.4943	0.4386	0.7056	0.1757
	recall	0.6274	**0.6397**	0.6385	0.6188	0.6321	0.6345	0.5864	0.6191	0.2831	0.4866
	fscore	0.7451	0.7590	0.7539	0.7409	0.7560	**0.7607**	0.4625	0.4476	0.3939	0.2478
6	*precision*	0.9682	0.9695	**0.9713**	0.9495	0.9567	0.9673	0.4851	0.4424	0.6959	0.1729
	recall	0.5765	0.5977	0.5959	0.5802	0.5968	**0.5994**	0.5351	0.5527	0.2897	0.4595
	fscore	0.7123	0.7335	0.7327	0.7105	0.7279	**0.7344**	0.4246	0.4060	0.3986	0.2417
7	*precision*	0.9446	0.9479	0.9410	0.9691	0.9648	**0.9747**	0.6382	0.6060	0.7350	0.1586
	recall	0.5478	**0.5650**	0.5638	0.5521	0.5626	0.5648	0.5287	0.5544	0.3006	0.4482
	fscore	0.6830	0.7019	0.6983	0.6940	0.7031	**0.7085**	0.5275	0.5155	0.4197	0.2284
8	*precision*	0.9685	0.9711	0.9640	0.9685	0.9622	**0.9741**	0.7900	0.7774	0.7323	0.1636
	recall	0.5074	0.5248	0.5164	0.5107	0.5242	**0.5251**	0.4450	0.4653	0.2778	0.4243
	fscore	0.6555	0.6764	0.6647	0.6606	0.6719	**0.6765**	0.5462	0.5584	0.3945	0.2261

Boldface values indicate the best values.

$$recall = \frac{1}{J}\sum_{j=1}^{J}\frac{|c_i \cap g_j|}{|g_j|}, \quad c_i \text{ is the best match for } g_j, \tag{4}$$

where c_i is the best match for g_j meaning $\forall t \in \{1, 2, \ldots, I\}$, $\frac{|c_i \cap g_j|}{|g_j|} \geq \frac{|c_t \cap g_j|}{|g_j|}$.

If the ground truth community g_j does not correspond to any found community, $\frac{|c_i \cap g_j|}{|g_j|}$ is low, so the value of *recall* will be pulled down. *fscore* is a comprehensive consideration of precision and recall, defined as,

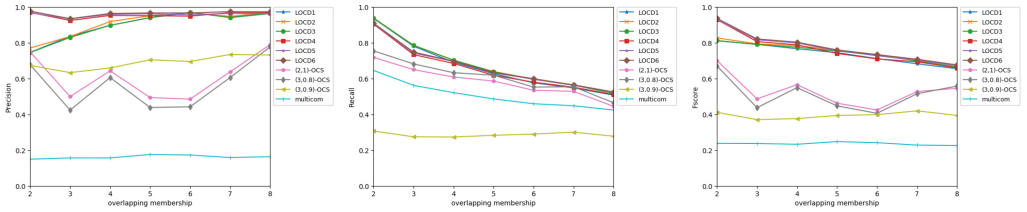
$$fscore = \frac{2 * precision * recall}{precision + recall}. \tag{5}$$

6.4 Experimental Results of Synthetic Networks

In the LFR datasets, each node is selected as the starting node for local overlapping community detection, and then the average values of *recall*, *precision*, and *fscore* are calculated for all nodes. The results are shown in Tables 2–4. To intuitively display the results in Tables 2–4, we draw the line chart shown in Figures 5–7, respectively.

6.4.1 Results of LFR Datasets with mu = 0.1. Table 2 summarizes the average *recall*, *precision*, and *fscore* of our algorithms, *OCS* algorithms and MULTICOM. We draw the following conclusions from Table 2.

(1) *om*=2: *LOCD*6 gets the highest values of *precision* and *fscore*. *LOCD*3 gets the highest value of *recall*.



(a) *precision* (b) *recall* (c) $fscore$

Fig. 5. Comparison of the average *precision*, *recall*, and $fscore$ on LFR datasets with *mu*=0.1.

(2) *om*=3: *LOCD*6 obtains the best values on *precision* and $fscore$. *LOCD*2 obtains the best value of *recall*. Our algorithms outperform *OCS* algorithms and MULTICOM on the *recall*, *precision*, and $fscore$.

(3) *om*=4: *LOCD*3 obtains the best performance on *recall*. *LOCD*6 obtains the best values on *precision* and $fscore$. The $fscore$ values obtained by our algorithms are all greater than 0.76, the $fscore$ values obtained by *OCS* algorithms are less than 0.6, and $fscore$ value obtained by MULTICOM is 0.2329.

(4) *om*=5: *LOCD*2 gets the highest value of *recall*. Our algorithms perform better than the *OCS* algorithms on the *precision* and $fscore$, and better than MULTICOM on all indicators.

(5) *om*=6: *LOCD*6 gets the highest values of *recall* and $fscore$ and *LOCD*3 gets the highest value of *precision*.

(6) *om*=7, 8: *LOCD*6 gets the best values of $fscore$ and *precision*. The $fscore$ values of our algorithms are all greater than 0.65, the $fscore$ values of *OCS* algorithms are less than 0.56, and the $fscore$ value of MULTICOM is below 0.23.

Figure 5(a) shows the average *precision* obtained by our algorithms, *OCS* algorithms, and MULTICOM. Obviously, *precision* values of the *OCS* algorithms and MULTICOM are lower than that of our algorithms except for $om = 2$. As *om* changes from 2 to 8, *precision* values of *LOCD*4, *LOCD*5, and *LOCD*6 are relatively stable. When *om*=6, *precision* values of *LOCD*4, *LOCD*5, and *LOCD*6 are less than that of *LOCD*1, *LOCD*2, and *LOCD*3. When *om*=2, 3, 4, 5, and 7, *precision* values of *LOCD*4, *LOCD*5, and *LOCD*6 are greater than that of *LOCD*1, *LOCD*2, and *LOCD*3.

Figure 5(b) shows the average *recall* obtained by our algorithms, *OCS* algorithms and MULTICOM. The *recall* values of (3, 0.9)-*OCS* and MULTICOM are significantly lower than that of other algorithms. When *om*=2, 3, 4, 6, and 8, *recall* values of (3, 0.8)-*OCS* and (2, 1)-*OCS* are lower than that of our algorithms.

Figure 5(c) shows the average $fscore$ obtained by our algorithms, *OCS* algorithms and MULTICOM. The performance of our algorithms is significantly better than that of the *OCS* algorithms.

In our algorithms, for the *recall*, *precision* and $fscore$, *LOCD*6 gets the highest value 15 times, and both *LOCD*2 and *LOCD*3 get the highest value three times. From this point of view, *LOCD*6 algorithm has the best performance and the performance of *LOCD*2 and *LOCD*3 is the next.

6.4.2 Results of LFR Datasets with mu = 0.3. Table 3 shows the average values of *precision*, *recall* and $fscore$ obtained by our algorithms, *OCS* algorithms and MULTICOM. We draw the following conclusions from Table 3.

(1) When *om*=3 and 4, *LOCD*5 gets the best value of *precision*.

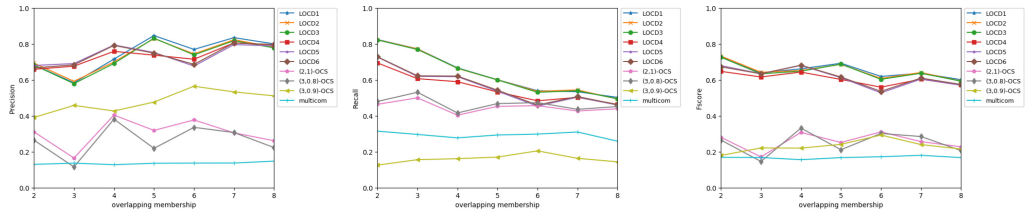
(2) When *om*=2, 3, and 7, *LOCD*2 gets the best value of $fscore$.

(3) When *om*= 5, 6, and 8, *LOCD*1 gets the highest values of $fscore$, *recall*, and *precision* and when *om*=2, 3, and 4, *LOCD*1 gets the highest value of *recall*.

Table 3. The Experimental Results on LFR Datasets with *mu* = 0.3

om		*LOCD1*	*LOCD2*	*LOCD3*	*LOCD4*	*LOCD5*	*LOCD6*	*(2,1)-OCS*	*(3,0.8)-OCS*	*(3,0.9)-OCS*	MULTICOM
	precision	0.6838	**0.6936**	0.6826	0.6591	0.6806	0.6670	0.3108	0.2645	0.3909	0.1297
2	*recall*	**0.8249**	0.8233	0.8227	0.6948	0.7272	0.7289	0.4643	0.4803	0.1257	0.3147
	fscore	0.7276	**0.7329**	0.7265	0.6476	0.6798	0.6731	0.2804	0.2652	0.1795	0.1689
	precision	0.5840	0.5921	0.5795	0.6767	**0.6912**	0.6842	0.1650	0.1154	0.4581	0.1365
3	*recall*	**0.7733**	0.7728	0.7694	0.6071	0.6198	0.6230	0.5003	0.5313	0.1557	0.2965
	fscore	0.6390	**0.6427**	0.6339	0.6165	0.6348	0.6340	0.1712	0.1459	0.2213	0.1679
	precision	0.7163	0.7000	0.6918	0.7592	**0.7946**	0.7921	0.4042	0.3815	0.4271	0.1280
4	*recall*	**0.6672**	0.6645	0.6658	0.5903	0.6181	0.6214	0.4037	0.4162	0.1615	0.2773
	fscore	0.6618	0.6523	0.6487	0.6437	0.6807	**0.6820**	0.3078	0.3317	0.2202	0.1557
	precision	**0.8469**	0.8306	0.8322	0.7381	0.7530	0.7494	0.3188	0.2189	0.4766	0.1358
5	*recall*	**0.6009**	0.6003	0.6002	0.5331	0.5373	0.5430	0.4528	0.4683	0.1701	0.2934
	fscore	**0.6930**	0.6870	0.6879	0.6026	0.6124	0.6160	0.2517	0.2113	0.2413	0.1673
	precision	**0.7700**	0.7449	0.7397	0.7151	0.6772	0.6866	0.3771	0.3359	0.5649	0.1369
6	*recall*	**0.5398**	0.5363	0.5312	0.4838	0.4541	0.4600	0.4566	0.4737	0.2050	0.2983
	fscore	**0.6195**	0.6084	0.6035	0.5596	0.5281	0.5363	0.3106	0.3019	0.2930	0.1722
	precision	**0.8354**	0.8240	0.8185	0.8061	0.7970	0.8060	0.3054	0.3070	0.5332	0.1371
7	*recall*	0.5343	**0.5445**	0.5395	0.5027	0.5040	0.5075	0.4270	0.4363	0.1634	0.3100
	fscore	0.6372	**0.6414**	0.6371	0.6040	0.6047	0.6103	0.2563	0.2848	0.2400	0.1802
	precision	**0.8008**	0.7824	0.7788	0.7957	0.7895	0.7970	0.2612	0.2233	0.5114	0.1477
8	*recall*	**0.5028**	0.4934	0.4950	0.4639	0.4600	0.4625	0.4394	0.4519	0.1438	0.2589
	fscore	**0.6014**	0.5936	0.5931	0.5717	0.5703	0.5747	0.2275	0.2065	0.2152	0.1674

Boldface values indicate the best values.



(a) *precision* (b) *recall* (c) *fscore*

Fig. 6. Comparison of the average *precision*, *recall* and *fscore* on LFR datasets with *mu*=0.3.

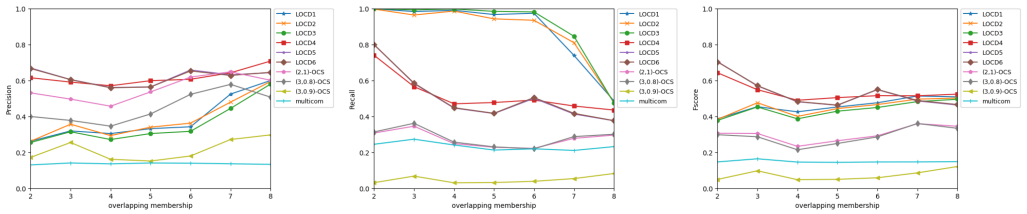
Figure 6(a)–(c) shows the average *precision*, *recall*, and *fscore*, respectively. Figure 6(a) shows the performance of *LOCD*2 and *LOCD*3 is almost the same. Figure 6(b) shows *LOCD*1, *LOCD*2, and *LOCD*3 outperform other algorithms on *recall*. Figure 6(c) shows the *fscore* values of *LOCD*1, *LOCD*2, and *LOCD*3 are greater than that of *LOCD*4, *LOCD*5, and *LOCD*6 except for *om*=3 and 4. Figure 6(a) and (c) shows *precision* and *fscore* values obtained by the *OCS* algorithms and MULTICOM are significantly lower than that of our algorithms.

For the *recall*, *precision*, and *fscore*, *LOCD*1 gets the highest value 13 times, and *LOCD*2 gets the highest value five times. From this point of view, *LOCD*1 algorithm gets the best performance, followed by *LOCD*2.

Table 4. The Experimental Results on LFR Datasets with *mu* = 0.4

om		*LOCD1*	*LOCD2*	*LOCD3*	*LOCD4*	*LOCD5*	*LOCD6*	*(2,1)-OCS*	*(3,0.8)-OCS*	*(3,0.9)-OCS*	MULTICOM
	precision	0.2621	0.2620	0.2550	0.6154	**0.6699**	0.6671	0.5312	0.3997	0.1720	0.1302
2	*recall*	**1.0000**	0.9978	**1.0000**	0.7410	0.8009	0.7997	0.3076	0.3137	0.0312	0.2451
	fscore	0.3866	0.3836	0.3777	0.6434	**0.7055**	0.7039	0.3064	0.2984	0.0491	0.1468
	precision	0.3192	0.3567	0.3147	0.5914	**0.6053**	0.6049	0.4965	0.3779	0.2550	0.1410
3	*recall*	0.9838	0.9648	**0.9934**	0.5660	0.5834	0.5858	0.3457	0.3619	0.0678	0.2734
	fscore	0.4544	0.4762	0.4529	0.5489	0.5700	**0.5715**	0.3053	0.2870	0.0976	0.1642
	precision	0.3047	0.2935	0.2716	**0.5709**	0.5614	0.5597	0.4572	0.3460	0.1616	0.1361
4	*recall*	0.9908	0.9867	**0.9983**	0.4704	0.4467	0.4486	0.2488	0.2570	0.0307	0.2412
	fscore	0.4257	0.4003	0.3869	**0.4902**	0.4817	0.4828	0.2336	0.2145	0.0480	0.1460
	precision	0.3321	0.3403	0.3048	**0.5991**	0.5643	0.5641	0.5367	0.4137	0.1518	0.1409
5	*recall*	0.9675	0.9434	**0.9851**	0.4775	0.4160	0.4175	0.2284	0.2306	0.0321	0.2128
	fscore	0.4527	0.4439	0.4293	**0.5048**	0.4621	0.4633	0.2646	0.2496	0.0499	0.1445
	precision	0.3425	0.3627	0.3172	0.6068	**0.6587**	0.6535	0.6193	0.5231	0.1800	0.1398
6	*recall*	0.9745	0.9355	**0.9821**	0.4912	0.5009	0.5053	0.2224	0.2199	0.0388	0.2197
	fscore	0.4762	0.4662	0.4499	0.5157	0.5492	**0.5502**	0.2922	0.2862	0.0586	0.1466
	precision	0.5246	0.4798	0.4449	0.6431	0.6320	0.6283	**0.6483**	0.5786	0.2717	0.1365
7	*recall*	0.7398	0.8105	**0.8465**	0.4581	0.4133	0.4170	0.2781	0.2878	0.0541	0.2108
	fscore	0.5136	0.4964	0.4826	**0.5150**	0.4862	0.4889	0.3599	0.3607	0.0860	0.1469
	precision	0.6000	0.5946	0.5802	**0.7083**	0.6437	0.6457	0.6014	0.5075	0.2974	0.1333
8	*recall*	**0.4863**	0.4778	0.4750	0.4356	0.3760	0.3767	0.2962	0.3010	0.0824	0.2320
	fscore	0.5089	0.5024	0.4958	**0.5240**	0.4635	0.4663	0.3455	0.3341	0.1212	0.1482

Boldface values indicate the best values.



(a) *precision* (b) *recall* (c) *fscore*

Fig. 7. Comparison of the average *precision*, *recall*, and *fscore* on LFR datasets with *mu*=0.4.

6.4.3 Results of LFR Datasets with mu = 0.4. Table 4 summarizes the average *recall*, *precision*, and *fscore* of our algorithms, *OCS* algorithms and MULTICOM. We draw the following conclusions from Table 4.

(1) *om*=2: *LOCD*1 and *LOCD*3 get the maximum value of 1 on *recall*, but these two algorithms have very low values on *precision*.

(2) *om*=4, 5: *LOCD*4 gets the highest values of *precision* and *fscore* and *LOCD*3 gets the highest value of *recall*.

(3) *om*=7: (2,1)-*OCS* gets the best value of *precision*, followed by *LOCD*4.

Figure 7(a)–(c) shows the average *precision*, *recall* and *fscore*, respectively. Figure 7(a) shows MULTICOM is the worst and (3,0.9)-*OCS* is the second worst on *precision*. Overall, the *precision* values of *LOCD*4, *LOCD*5, and *LOCD*6 are better than that of the rest algorithms. Figure 7(b)

Table 5. The Experimental Results of the Amazon Dataset

		LOCD1	*LOCD2*	*LOCD3*	*LOCD4*	*LOCD5*	*LOCD6*	*(2,1)-OCS*	*(3,0.8)-OCS*	*(3,0.9)-OCS*	MULTICOM
all	*precision*	0.9636	**0.9669**	0.9631	0.9613	0.9640	0.9618	0.9281	0.8798	0.7558	0.4811
	recall	0.5135	0.5128	0.5145	0.4495	0.4465	0.4504	0.4468	0.4398	0.3031	**0.5411**
	fscore	0.6220	0.6227	**0.6229**	0.5605	0.5583	0.5618	0.5585	0.5451	0.3994	0.3952
top 5000	*precision*	0.9065	**0.9168**	0.9061	0.8923	0.9027	0.8935	0.9018	0.8534	0.7444	0.1861
	recall	0.7732	0.7722	0.7748	0.6787	0.6752	0.6800	0.6805	0.6612	0.4572	**0.8027**
	fscore	0.7816	**0.7863**	0.7825	0.7096	0.7111	0.7112	0.7247	0.6987	0.5244	0.2727

Boldface values indicate the best values.

shows our algorithms are significantly better than *OCS* algorithms and MULTICOM in terms of *recall*. The *recall* values of *LOCD*1, *LOCD*2, and *LOCD*3 are stable when *om* changes from 1 to 6, and rapidly decrease when *om* changes from 6 to 8. Figure 7(c) shows that the $fscore$ values of our algorithms are significantly better than that of *OCS* algorithms and MULTICOM. The $fscore$ values of *LOCD*1, *LOCD*2, and *LOCD*3 increase as *om* increases.

6.5 Experimental Results of a Real-World Network

We apply our algorithms on the Amazon dataset to further test our methods. To measure the average performance of algorithms, we select some nodes from the top 5,000 communities as the starting nodes. Specifically, according to the number of the top 5,000 communities that nodes belong to, all the nodes in the top 5,000 communities are divided into several parts. For example, nodes belonging to one community form part 1, and nodes belonging to two communities constitute part 2. Then, we evenly select 10% nodes from all parts and nodes in each part are sorted by the degree of node. Assume that the number of nodes in the part is l. The indexes of the selected nodes are $\lfloor i * (l-1)/(\lceil 0.1 * l \rceil - 1) \rfloor$, i=0, 1, 2, ..., $\lceil 0.1 * l \rceil - 1$. Each selected node is as the starting node for local overlapping community detection, and then the averages of *recall*, *precision*, and $fscore$ are calculated.

Stanford Network Analysis Project provides all the communities in the network, and also provides the top 5,000 communities with highest quality. We use these two types of communities as the ground truth communities, respectively. Table 5 shows the experimental results.

The upper half of Table 5 shows the results of experiments with all communities as ground truth communities. *LOCD*3 gets the best value of $fscore$, *LOCD*2 achieves the best value of *precision*, and MULTICOM gets the best value of *recall*. *precision*, *recall*, and $fscore$ of *LOCD*1, *LOCD*2, and *LOCD*3 are higher than that of the *OCS* algorithms. *LOCD*1, *LOCD*2, and *LOCD*3 are better than MULTICOM on *precision* and $fscore$, and worse than MULTICOM on *recall*.

The lower half of Table 5 shows the results of experiments with top 5,000 communities as ground truth communities. *LOCD*2 obtains the best performance on *precision* and $fscore$. Obviously, *LOCD*1, *LOCD*2, and *LOCD*3 outperform *OCS* algorithms on the *recall*, *precision*, and $fscore$, and outperform MULTICOM on *precision* and $fscore$.

In our algorithms, for *recall*, *precision*, and $fscore$, *LOCD*2 gets the highest value three times and *LOCD*3 gets the highest value once. From this point of view, *LOCD*2 achieves the best result.

We comprehensively compare the results of the algorithms on all the LFR datasets and the Amazon dataset. For each LFR dataset with *mu*=0.1 and each indicator, all algorithms are ranked according to the value of the algorithm on the indicator. The smaller the ranking of the algorithm means the better its performance. After that, calculate the sum of the ranking of each algorithm on all indicators and LFR datasets with *mu*=0.1. Further, re-ranking all algorithms based on the sum of the rankings calculated above. These operations are also performed on LFR datasets with *mu*=0.3 and *mu*=0.4, as well as the Amazon dataset.

Table 6. The Rankings of Algorithms on All the LFR Datasets and Amazon Datasets

	LOCD1	*LOCD2*	*LOCD3*	*LOCD4*	*LOCD5*	*LOCD6*	*(2,1)-OCS*	*(3,0.8)-OCS*	*(3,0.9)-OCS*	MULTICOM
mu=0.1	106/6	57/2	84/4	94/5	69/3	35/1	158/7	162/8	186/9	203/10
mu=0.3	36/1	51/2	71/3	106/6	96/5	84/4	164/7	168/8	178/9	201/10
mu=0.4	82/4	94/5	104/6	62/1	73/3	68/2	129/7	143/8	203/10	196/9
Amazon	17/3	13/1	14/2	38/7	35/6	31/4	34/5	50/9	56/10	42/8
sum of new rankings	14	10	15	19	17	11	26	33	38	37

Table 6 shows the sum of the rankings of all algorithms on the LFR datasets and the Amazon dataset and the new rankings (based on the sum of the rankings). For each algorithm, the sum of new rankings is calculated, shown in the last row of Table 6. The smaller the sum of new rankings means the better the performance of the algorithm. So overall, *LOCD2* has the best performance.

7 DISCUSSION

In this section, the number of nodes that no community is found is discussed. Further, the effect of the local community detection algorithm and the effect of parameters settings are also discussed.

7.1 Number of Nodes that No Community is Found

As already mentioned in Section 1, the *OCS* algorithms cannot find the community for some nodes. Table 7 shows the number of nodes that no community is found by *OCS* algorithms on datasets used in the experiments.

For the LFR dataset with *mu*=0.1 and *om*=2, the number of nodes that no community is found by (2, 1)-*OCS*, (3, 0.8)-*OCS* and (3, 0.9)-*OCS* is 41, 45, and 326, respectively. The number of nodes that no community is found by (3, 0.9)-*OCS* is greater than that of (3, 0.8)-*OCS*. The number of nodes that no community is found by (3, 0.8)-*OCS* is greater than that of (2, 1)-*OCS*. Similar results on other LFR datasets are obtained and omitted to save space. From the phenomenon, we can see that parameters greatly affect on the performance of the *OCS* algorithms.

For Amazon dataset, the number of nodes that no community is found by (2, 1)-*OCS*, (3, 0.8)-*OCS*, and (3, 0.9)-*OCS* is 91, 172, and 402, respectively.

Our algorithms will find at least one community for each node. Compared with *OCS* algorithms, our algorithms has better robustness. In MULTICOM, the number of found communities is a predetermined parameter. For a given node, a specified number of communities are discovered.

7.2 Effect of the Local Community Detection Methods

The ideal local community detection algorithm for the proposed framework satisfies the following two expectations: (1) For the overlapping nodes v_0 belonging to communities $C_1, C_2, \ldots, C_T$, the community C_0 found by local community detection algorithm contains some nodes in the communities $C_1, C_2, \ldots, C_T$. (2) For the non-overlapping node v, the local community detection algorithm can accurately discover the community that contains node v.

The better the algorithm performs in these two aspects, the better its final performance. We take the *DMF_M* and *M* methods as examples. In most cases, the performance of *DMF_M* is better than that of the *M* method in accurately finding the community for the non-overlapping node [28]. Therefore, in Section 6, in most cases, the performance of *LOCD*1, *LOCD*2, and *LOCD*3 algorithms using *DMF_M* method is better than that of the *LOCD*4 and *LOCD*5 algorithms using *M* method.

Table 7. Number of Nodes that No Community is Found by *OCS* Algorithms

Dataset		(2,1)-OCS	(3,0.8)-OCS	(3,0.9)-OCS
LFR *mu*=0.1	*om*=2	41	45	326
	om=3	61	70	366
	om=4	74	75	340
	om=5	74	81	294
	om=6	91	104	303
	om=7	66	74	265
	om=8	91	102	267
LFR *mu*=0.3	*om*=2	190	217	607
	om=3	166	180	538
	om=4	198	245	572
	om=5	154	189	508
	om=6	137	153	431
	om=7	119	158	465
	om=8	131	150	485
LFR *mu*=0.4	*om*=2	228	359	826
	om=3	191	285	745
	om=4	260	355	836
	om=5	265	376	845
	om=6	243	354	820
	om=7	203	275	726
	om=8	166	245	701
Amazon		91	172	402

Meanwhile, *LOCD*6 using *M* method is not bad because both C_0N and *neighK* are used to construct *candidates*.

7.3 Effect of *k*

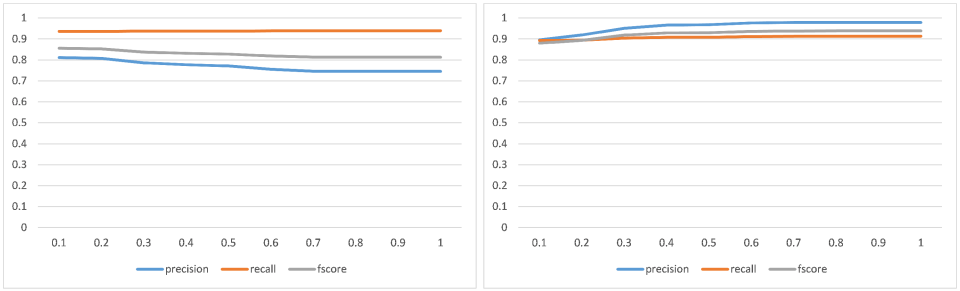
We use the performance of *LOCD*2 and *LOCD*5 on the LFR dataset with *mu* = 0.1 and *om*=2 to illustrate the effect of *k* on the final results and set the threshold of fuzzy relation to 0.5. The *candidates* in the *LOCD*2 and *LOCD*5 algorithms is obtained by *nieghK*. As *k* increases, the number of nodes in *candidates* gradually increases.

Figure 8 shows the performance of the *LOCD*2 and *LOCD*5 under different *k*. Figure 8(a) shows that the *recall* value of the *LOCD*2 algorithm is stable, and the *precision* has a slight downward trend.

Figure 8(b) shows the performance of *LOCD*5 becomes better when α increases from 0.1 to 0.3, and the performance of *LOCD*5 is stable when $\alpha > 0.3$. This phenomenon shows that as *k* increases, the community structure discovered by the *LOCD*5 is more accurate.

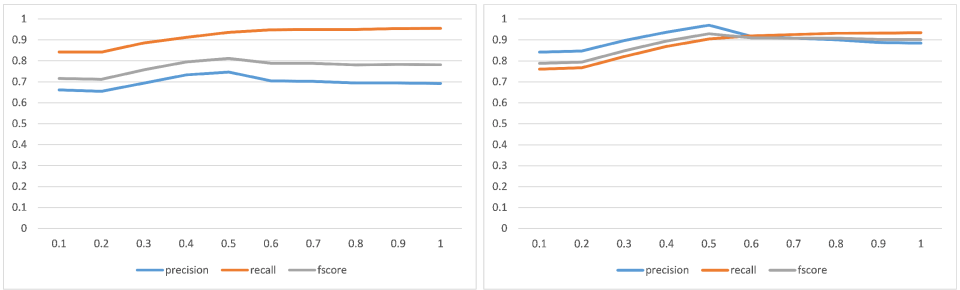
7.4 Effect of the Threshold of the Fuzzy Relation

The threshold of the fuzzy relation controls the number of representative nodes. The larger its value, the more representative nodes are selected. Here, we use the performance of *LOCD*1 and *LOCD*4 on the LFR dataset with *mu* = 0.1 and *om*=2 to investigate the effect of the fuzzy relation threshold on the final performance.



(a) *LOCD*2 (b) *LOCD*5

Fig. 8. The performance of *LOCD*2 and *LOCD*5 with looping α from 0.1 to 1, with a step size of 0.1. ($k = \lceil \alpha * diameter(C_0) \rceil$.)



(a) *LOCD*1 (b) *LOCD*4

Fig. 9. The performance of *LOCD*1 and *LOCD*4 with looping the threshold of fuzzy relation from 0.1 to 1, with a step size of 0.1.

Figure 9 shows the performance of *LOCD*1 and *LOCD*4 under different thresholds of the fuzzy relation. From Figure 9, we can reach the following conclusions.

(1) The threshold changes from 0.1 to 0.5. The *recall*, *precision*, and *fscore* of *LOCD*1 and *LOCD*4 are first stable and then gradually increase. The number of selected representative nodes gradually increases as the threshold increases. As the number of representative nodes increases, the likelihood that representative nodes can cover communities to which the given node belongs also rises. Therefore, the performance of *LOCD*1 and *LOCD*4 becomes better as the threshold increases.

(2) The threshold changes from 0.5 to 0.6. The *recall* values of *LOCD*1 and *LOCD*4 rise, while the *precision* and *fscore* of *LOCD*1 and *LOCD*4 decrease. The reason for the rise of *recall* is the same as the reason for the increase of *recall* when the threshold is from 0.1 to 0.5.

The reason for the decrease of *precision* and *fscore* is explained as follows. The fuzzy relations of nodes calculated using the method in Ref. [27] have the following characteristics.

(a) The number of nodes with fuzzy relations less than 0.5 is much smaller than the number of nodes with fuzzy relations greater than or equal to 0.5.

(b) Nodes with fuzzy relations greater than or equal to 0.5 are not considered as community centers. Nodes with fuzzy relations less than 0.5 may be community centers, and smaller fuzzy relations of nodes indicates that they are more likely to be community centers.

According to the above characteristics of fuzzy relations of nodes, when the threshold changes from 0.5 to 0.6, the number of selected representative nodes increases rapidly and some of the newly added nodes are boundary nodes. The communities detected with these boundary nodes

as the starting node may be inaccurate, resulting in the decrease of *precision*. The *fscore* value is also reduced because of the decrease of *precision*.

(3) The threshold varies from 0.6 to 1.0. The *recall* values of *LOCD*1 and *LOCD*4 have a slight upward trend, while the *precision* and *fscore* values of *LOCD*1 and *LOCD*4 have a slow downward trend. The reason for the former case is similar to the reason for the increase of *recall* when the threshold changes from 0.1 to 0.5. The reason for the latter case is similar to the reason for the decrease of *precision* and *fscore* when the threshold changes from 0.5 to 0.6.

From the above analysis, we can understand that the reason why *LOCD*1 and *LOCD*4 achieve the optimal value when the threshold is 0.5. This is also the reason why the threshold is set to 0.5.

8 CONCLUSION

Local overlapping community detection is, only by local information, to find the communities containing the given overlapping node. It is very useful when global information about the network is unavailable or expensive to acquire. In this article, we propose a framework to discover local overlapping communities that contain the given node. It contains three main steps. First, we find the set of candidate nodes. Second, we select representative nodes from the set of candidate nodes. Third, we discover the communities to which these representative nodes belong. In addition, to demonstrate the effectiveness of the framework, we implement six versions of the proposed framework. In our implementations, overall, *LOCD*2 is the best one and *LOCD*6 performs the second. The experimental results show the implementations of the framework outperform the other approaches on synthetic and real-world datasets.

REFERENCES

[1] James P. Bagrow. 2008. Evaluating local community methods in networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 5 (2008), P05001.

[2] James P. Bagrow and Erik M. Bollt. 2005. Local method for detecting communities. *Physical Review E* 72, 4 (2005), 046108.

[3] Xueying Bai, Peilin Yang, and Xiaohu Shi. 2017. An overlapping community detection algorithm based on density peaks. *Neurocomputing* 226 (2017), 7–15.

[4] Abhijnan Chakraborty, Saptarshi Ghosh, and Niloy Ganguly. 2012. Detecting overlapping communities in folksonomies. In *Proceedings of the 23rd ACM Conference on Hypertext and Social Media*. ACM, Milwaukee, WI, 213–218.

[5] Tanmoy Chakraborty. 2015. Leveraging disjoint communities for detecting overlapping community structure. *Journal of Statistical Mechanics: Theory and Experiment* 2015, 5 (2015), P05017.

[6] Tanmoy Chakraborty, Suhansanu Kumar, Niloy Ganguly, Animesh Mukherjee, and Sanjukta Bhowmick. 2016. GenPerm: A unified method for detecting non-overlapping and overlapping communities. *IEEE Transactions on Knowledge and Data Engineering* 28, 8 (2016), 2101–2114.

[7] Jiyang Chen, Osmar R. Zaïane, and Randy Goebel. 2009. Local community identification in social networks. In *Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining*. IEEE Computer Society, Athens, Greece, 237–242.

[8] Qiong Chen and Ting-Ting Wu. 2010. A method for local community detection by finding maximal-degree nodes. In *Proceedings of the 2010 International Conference on Machine Learning and Cybernetics*, Vol. 1. IEEE, Qingdao, China, 8–13.

[9] Qiong Chen, Ting-Ting Wu, and Ming Fang. 2013. Detecting local community structures in complex networks based on local degree central nodes. *Physica A: Statistical Mechanics and Its Applications* 392, 3 (2013), 529–537.

[10] Aaron Clauset. 2005. Finding local community structure in networks. *Physical Review E* 72, 2 (2005), 026132.

[11] Michele Coscia, Fosca Giannotti, and Dino Pedreschi. 2011. A classification for community discovery methods in complex networks. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 4, 5 (2011), 512–546.

[12] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, 277–288.

[13] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, Snowbird, UT, 991–1002.

[14] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. 2007. Extraction and classification of dense communities in the web. In *Proceedings of the 16th International Conference on World Wide Web*. ACM, Banff, Alberta, Canada, 461–470.
[15] T. S. Evans and R. Lambiotte. 2010. Line graphs of weighted networks for overlapping communities. *The European Physical Journal B* 77, 2 (2010), 265–272.
[16] Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486, 3–5 (2010), 75–174.
[17] Steve Gregory. 2010. Finding overlapping communities in networks by label propagation. *New Journal of Physics* 12, 10 (2010), 103018.
[18] Kun He, Yiwei Sun, David Bindel, John E. Hopcroft, and Yixuan Li. 2015. Detecting overlapping communities from local spectral subspaces. In *Proceedings of the 2015 IEEE International Conference on Data Mining*. IEEE Computer Society, Atlantic City, NJ, 769–774.
[19] Alexandre Hollocou, Thomas Bonald, and Marc Lelarge. 2017. Multiple local community detection. *SIGMETRICS Performance Evaluation Review* 45, 3 (2017), 76–83.
[20] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, Snowbird, UT, 1311–1322.
[21] Jingchun Chen and Bo Yuan. 2006. Detecting functional modules in the yeast protein–protein interaction network. *Bioinformatics* 22, 18 (2006), 2283–2290.
[22] Dany Kamuhanda and Kun He. 2018. A nonnegative matrix factorization approach for multiple local community detection. In *Proceedings of the IEEE/ACM 2018 International Conference on Advances in Social Networks Analysis and Mining*. IEEE Computer Society, Barcelona, Spain, 642–649.
[23] Kyle Kloster and David F. Gleich. 2014. Heat kernel based community detection. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, 1386–1395.
[24] Ann E. Krause, Kenneth A. Frank, Doran M. Mason, Robert E. Ulanowicz, and William W. Taylor. 2003. Compartments revealed in food-web structure. *Nature* 426, 6964 (2003), 282.
[25] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Physical Review E* 78, 4 (2008), 046110.
[26] Feng Luo, James Zijun Wang, and Eric Promislow. 2008. Exploring local community structures in large networks. *Web Intelligence and Agent Systems* 6, 4 (2008), 387–400.
[27] Wenjian Luo, Zhenglong Yan, Chenyang Bu, and Daofu Zhang. 2017. Community detection by fuzzy relations. *IEEE Transactions on Emerging Topics in Computing*. DOI: 10.1109/TETC.2017.2751101
[28] Wenjian Luo, Daofu Zhang, Hao Jiang, Li Ni, and Yamin Hu. 2018. Local community detection with the dynamic membership function. *IEEE Transactions on Fuzzy Systems* 26, 5 (2018), 3136–3150.
[29] Stefanie Muff, Francesco Rao, and Amedeo Caflisch. 2005. Local modularity measure for network clusterizations. *Physical Review E* 72, 5 (2005), 056107.
[30] M. E. J. Newman. 2010. *Networks: An Introduction*. Oxford University Press.
[31] M. E. J. Newman and M. Girvan. 2004. Finding and evaluating community structure in networks. *Physical Review E* 69, 2 (2004), 026113.
[32] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (2005), 814–818.
[33] Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali, and Ploutarchos Spyridonos. 2012. Community detection in social media - Performance and application considerations. *Data Mining and Knowledge Discovery* 24, 3 (2012), 515–554.
[34] Michel Plantié and Michel Crampes. 2013. Survey on social community detection. In *Social Media Retrieval*. Springer, 65–85.
[35] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E* 76, 3 (2007), 036106.
[36] Alexander W. Rives and Timothy Galitski. 2003. Modular organization of cellular networks. *Proceedings of the National Academy of Sciences* 100, 3 (2003), 1128–1133.
[37] Huawei Shen, Xueqi Cheng, Kai Cai, and Mao-Bin Hu. 2009. Detect overlapping and hierarchical community structure in networks. *Physica A: Statistical Mechanics and its Applications* 388, 8 (2009), 1706–1712.
[38] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Washington, DC, 939–948.
[39] Wenjian Luo Wenhao Gao and Chenyang Bu. 2017. Adapting the TopLeaders algorithm for dynamic social networks. *The Journal of Supercomputing* (2017), 1–23.
[40] Joyce Jiyoung Whang, David F. Gleich, and Inderjit S. Dhillon. 2016. Overlapping community detection using neighborhood-inflated seed expansion. *IEEE Transactions on Knowledge and Data Engineering* 28, 5 (2016), 1272–1284.

[41] Yingjun Wu, Han Huang, Zhifeng Hao, and Feng Chen. 2012. Local community detection using link similarity. *Journal of Computer Science and Technology* 27, 6 (2012), 1261–1268.

[42] Jierui Xie, Stephen Kelley, and Boleslaw K. Szymanski. 2013. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Computing Surveys* 45, 4 (2013), 43:1–43:35.

[43] Jierui Xie and Boleslaw K. Szymanski. 2012. Towards linear time overlapping community detection in social networks. In *Proceedings of the 16th Pacific-Asia Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, Kuala Lumpur, Malaysia, 25–36.

[44] Jierui Xie, Boleslaw K. Szymanski, and Xiaoming Liu. 2011. SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining Workshops*. IEEE Computer Society, Vancouver, BC, Canada, 344–349.

[45] Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. In *Proceedings of the 12th IEEE International Conference on Data Mining*. IEEE Computer Society, Brussels, Belgium, 745–754.

[46] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. 2017. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Halifax, NS, Canada, 555–564.

[47] Shihua Zhang, Rui-Sheng Wang, and Xiang-Sun Zhang. 2007. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and Its Applications* 374, 1 (2007), 483–490.

[48] Xianchao Zhang, Liang Wang, Yueting Li, and Wenxin Liang. 2011. Extracting local community structure from local cores. In *Proceedings of the 16th International Conference on Database Systems for Advanced Applications*. Springer, Hong Kong, China, 287–298.

Received January 2019; revised July 2019; accepted September 2019