Community Detection Based on Modularity and k-Plexes

Jinrong Zhu[a], Bilian Chen[a,*], Yifeng Zeng[b]

[a]*Department of Automation, Xiamen University, Xiamen 361005, China.*
[b]*School of Computing, Teesside University, UK.*

Abstract

Community identification is of great worth for analyzing the structure or characteristics of a complex network. Many community detection methods have been developed, such as modularity-based optimization models, which are widely used but significantly restricted in "resolution limit". In this paper, we propose a novel algorithm, called modularity optimization with k-plexes (MOKP), to solve this problem, and this algorithm can identify communities smaller than a scale. The proposed algorithm uses k-plexes to generate community seeds from the whole network and assigns the remaining nodes by modularity optimization. To save computational time, we further propose the improved MOKP algorithm (IMOKP) by reducing the scale of the network before community seeds generation and adjusting rules of nodes assignment. Extensive experimental results demonstrate our proposed algorithms perform better than several state-of-the-art algorithms in terms of accuracy of detected communities on various networks, and can effectively detect small communities in terms of a newly defined index, namely small community level, on multiple networks as well.

Keywords:
Community Detection, Modularity, k-Plex, Small Community

1. Introduction

Complex networks are widely used to model different types of relations and processes in physical, biological, economic, social and information systems. Given the complicated structure and large scale of networks, community detection, also known as graph clustering, is crucial for studying the organization of networks. Community detection is described as segmenting the graph into disjoint parts where nodes (or vertices) are dense in the same part but sparse between parts. Communities are groups of nodes, wherein the nodes within a group are more tightly connected to one another than to the rest of the network [34, 37]. Communities also indicate functional entities in a network. For example, in protein–protein interaction networks, communities represent groups of proteins with a specific function [6, 31]; in social networks, communities are groups of people with similar interests or features [20, 13]; in product co-purchasing networks (e.g., Amazon and eBay), communities correspond to the categories of products. Moreover, communities are useful in recommendation systems (e.g.,

*Corresponding author
Email addresses: jinrongzhu@stu.xmu.edu.cn (Jinrong Zhu), blchen@xmu.edu.cn (Bilian Chen), y.zeng@tees.ac.uk (Yifeng Zeng)

location, music and film recommendation), i.e., Feng et al. [9] adopted community detection technique to form collaborative recommendations since members in the same community share similar interests.

In recent decades, numerous studies have been conducted on identifying communities in networks, among which, two main approaches are significant and worth mentioning. The first one is graph partitioning (e.g., normalized cut [35] and spectral partitioning [22]), which aims to divide a network into several disjoint modules with the same size. The second one is the modularity-based algorithm [24], which uses an optimization function to define the community detection problem. Nevertheless, these two approaches have a common serious drawback, i.e., they both focus on finding communities that are larger than a certain size rather than small communities. However, small communities usually exist in real-world networks. Many other types of algorithms are also designed for community detection, e.g., ITDC [19], MNDP [14], DNR [44], LPA [28], two-layer RBM [1] and DIR [15]. However, these algorithms rarely refer to small communities and they are not the tailor for the "resolution limit" problem from the angle of modularity. Thus, we intend to design a modularity-based method to solve the problem in our work.

Furthermore, k-plex [32] can help detect both small and large communities of a network. Topologically, k-plexes (with a small k) are denser than the other parts of a network, and it is in line with the fact that nodes in a network are dense within communities and sparse in different communities. Note that a k-plex is a set of nodes in which each node is not adjacent to at most $k-1$ nodes and the nodes in it become much denser when k decreases. Hence, we can compute numerous different sizes of k-plexes for a given k and then regard them as candidate communities for further community detection techniques. Under this setting, several distinct small groups will be found in the network. Xiao et al. [40] proposed a fast algorithm to compute maximum k-plexes, which performs better than the other state-of-the-art methods in terms of efficiency. Hence, we will utilize the fast algorithm to detect k-plexes in our algorithm.

In this work, we propose a novel modularity optimization with k-plexes algorithm (MOKP) to detect both small and large communities of a network. The algorithm computes k-plexes of the network as community seeds, and then allocates the remaining nodes to some proper community seeds by modularity optimization algorithm. To speed up the computation, we further propose an improved modularity optimization with k-plexes algorithm (IMOKP). It improves the former algorithm from two sides: one is reducing the scale of the network by the technique of k-core before conducting community seeds creation, and the other one is adjusting the order of allocating the remaining nodes from their alphabetical order to descending order of node degree, and subsequently adding community labels to the node assignment process. We use three different metrics to evaluate the detected communities and further propose a new comprehensive metric to measure the quality of detected small communities. The extensive experimental results show that our algorithms perform better than several state-of-the-art algorithms on nine networks containing both real-world and synthetic networks. We also conduct experiments to illustrate the necessity of detecting communities and even small communities on a real network.

The rest of this paper is organized as follows. Section 2 briefly provides some related works. Section 3 presents some preliminaries of k-plexes computation and modularity optimization. Section 4 presents our two new algorithms, i.e., MOKP and IMOKP. Experimental

evaluations are performed in Section 5. Finally, we make a conclusion and give some suggestions on the future work in Section 6.

2. Related works

Community detection is widely studied and various methods have been proposed thus far. We mainly discuss disjoint community detection methods and k-plexes computational methods in this section.

2.1. Disjoint community detection methods

Simon [36] was the first to explore module structural characteristics in a complex system. Such characteristics are consistent with the nature of a community evolved from graph partition. Kernighan and Lin [16] designed a heuristic procedure based on gain function optimization, which is a local graph partition method, for dividing graphs. Thereafter, global graph partition methods (e.g., spectral clustering [17]) have been proposed, and it has been found that the global methods perform better than local ones when a graph is divided approximately into two parts. If the graph has more than two parts, we can use the partition algorithm repeatedly on subgraphs. Girvan and Newman [11] also proposed GN algorithm to iteratively split a network by removing edges with high betweenness centrality. These aforementioned algorithms require a partitioning number of vertices as termination criterion and it should be set in advance. However, it is not easy to initialize the partitioning number of vertices, which makes these algorithms impractical for real networks.

To address this issue, Newman and Griven [25] introduced a network division measure (called modularity) and used it as stopping criterion of their algorithm, resulting in an objective way for choosing the number of communities. However, the high complexity of computing modularity in this algorithm cannot be disregarded. Subsequently, Newman [23] proposed a fast algorithm in which every node is initially a community and then these communities try to merge together in order to optimize modularity. This new algorithm performs better than GN in terms of running time but worse in terms of accuracy. Hence, Clauset et al. [8] further improved it in terms of running time by using several shortcuts in the optimization problem and more sophisticated data structures. Later on, Blondel et al. [4] proposed a heuristic method based on modularity optimization. Despite its acknowledged success, Fortunato and Barthlemy [10] proved that modularity optimization may fail to detect the community whose size is less than a scale, which depends on the total number nodes of a network and the degree of the interconnectedness of communities.

Moreover, many other methods that do not suffer from the resolution limit problem have been proposed. Raghavan et al. [28] proposed a label propagation method. However, this method is nondeterministic, which indicates that different communities will be detected when the algorithm is operated repeatedly. Shao et al. [33] viewed a given network as an adaptive dynamic system to detect communities. In addition, some other methods, such as Infomap [30], SLPA [41], and DCLP [39], are also available. However, except for the algorithm proposed in [33], other algorithms provide few descriptions for small community detection. In this paper, we mainly focus on detecting communities based on the criterion of modularity, and the effectiveness of finding small communities by using our new methods is also analyzed.

2.2. Computing k-plexes methods

Clique, a subgraph such that every two vertices are adjacent, was first used by Luce and Perry [21] to model cliques of people who know each other in social networks. Clique is also a commonly used model for a community, which was studied in [5, 38]. To relax the familiarity restriction of a clique, Seidman and Foster [32] proposed a relaxation model, namely k-plex. Each node of a k-plex is not adjacent to at most $k-1$ nodes. Computing the maximum k-plex problem in a graph is crucial but challenging, partly due to the NP-completeness of obtaining all the maximum k-plexes. A maximum k-plex is the one cannot be induced by another k-plex. More recently, Xiao et al. [40] developed a fast algorithm for identifying the maximum k-plexes by exploring structural properties of the community detection problem, and this method performs better than other state-of-the-art methods in terms of efficiency.

In our perception, a k-plex or several k-plexes can be defined as a community (e.g., the CPM algorithm [26]). We take advantage of identifying many k-plexes with various sizes for a given k in a network, and focus on detecting communities by applying k-plex to a modularity-based optimization model, which may solve the resolution limit problem eventually.

3. Preliminaries

In this section, we first present the definition of k-plex proposed by Seidman and Foster [32] and how to compute the maximum k-plexes problem in a social network [40], which are both necessary parts for designing our algorithms, and then we introduce modularity proposed by Newman [24] which is a measure for the quality of detected communities and is used as an optimized objective function in our algorithms.

3.1. Maximum k-plexes computation

We introduce the definition of k-plex [32] in the following.

Definition 3.1 (k-plex). *A subgraph with n nodes is a k-plex if the number of neighbors for each node in the subgraph is more than $n-k$. Especially, k-plex is a clique if $k=1$.*

Computing maximum k-plexes problem is a hot topic in social networks, where a maximum k-plex represents any k-plex is not a subgraph of itself and hence its size is the largest. Naturally, it is NP-complete to compute the maximum k-plexes problem. Xiao et al. [40] gave a Branch-and-Search (BS) algorithm to quickly find the maximum k-plexes of a given graph G. Each detected maximum k-plex in their algorithm contains a constrained set F, where F serves an input and all the nodes in F are arbitrarily selected from the graph G before operating the BS algorithm. We represent this algorithm as $BS(I=(G;k;F);bound)$, where I denotes the F-constrained k-plex problem and *bound* is an integer served as an input parameter. It intends to find an F-constrained k-plex with a size of at least *bound*. The basic process of the BS algorithm is described as follows:

- create a candidate set C that contains vertices of the graph G excluding vertices of the constrained set F, and initialize a target set T as the constrained set F;
- reduce graph G by removing reducible vertices from C and removing vertices which are not satisfying *bound*;

- prune the search tree by branching on dominated vertices, F-vertices and U-vertices, where $U = V \setminus F$ and V is the set of vertices of G;
- recursively repeat the 2nd and 3rd procedures till the candidate set C is null.

When the algorithm stops, the target set T is the maximum F-constrained k-plexes, and the size of each detected k-plex automatically exceeds the integer *bound*. We also know that BS runs in $\sigma_k^N N^{O(1)}$ time, where $\sigma_k < 2$, and BS achieves the best performance among other state-of-the-art algorithms that run in $2^N N^{O(1)}$ time. In the following, we give an example to show the detailed process on computing maximum k-plex by the BS algorithm.

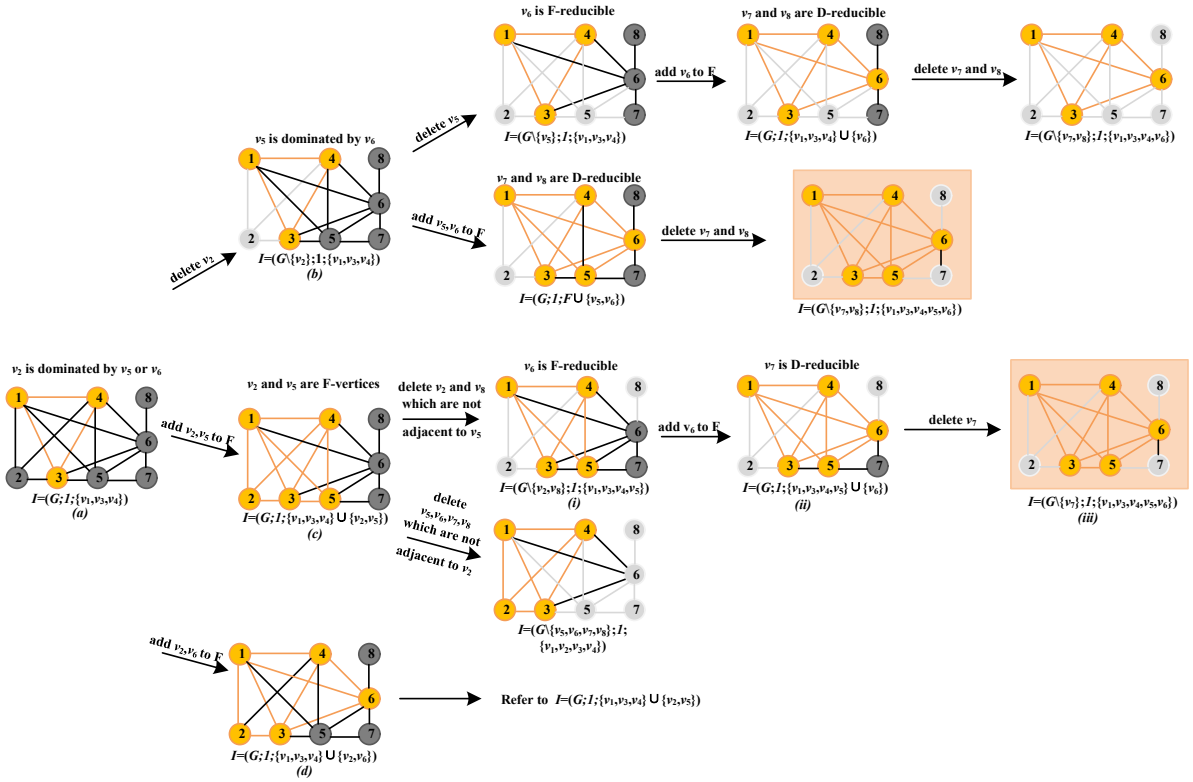


Fig. 1. Computing maximum k-plexes by using the BS algorithm.

Example 3.2. *Assuming that a network G has 8 nodes and 16 edges, as depicted in Fig. 1(a). We apply the BS algorithm to compute the maximum k-plex in the network G where k=1, F=$\{v_1, v_3, v_4\}$ and bound = 5. This process is represented as $BS(I = (G;1;\{v_1, v_3, v_4\});5)$. We first initialize $T = F = \{v_1, v_3, v_4\}$ and $C = G \setminus F = \{v_2, v_5, v_6, v_7, v_8\}$. We then reduce and prune the graph G by some rules. For node v_2 in the set C, v_2 is dominated by v_5 or v_6 since any neighbour of v_2 is a neighbour of v_5 or v_6. Thus, node v_2 is either deleted from G and C (see Fig. 1(b)) or added to T and F (see Fig. 1(c) and (d)). Next, we further operate the BS algorithm on these three k-plex problems to get the results respectively. Take branch (c) as an example, v_2 and v_5 are F-vertices in F because the degrees of v_2 and v_5 are both less than $V_F - k$ (where V_F denotes the number of nodes in F). Hence, we can prune it into two branches. One is to delete v_2 and nodes that are not adjacent to v_5 from C and G (i.e.,*

v_8), and the other one is to delete v_5 and nodes that are not adjacent to v_2 from C and G (i.e., v_6, v_7, v_8). In branch (i), v_6 is F-reducible because its degree is no less than $V_F - k$. Hence, we add v_6 to F and T, see Fig. 1 (ii). In branch (ii), v_7 is D-reducible because the remaining number of nodes in F is large than k after removing neighbors of v_7 from F and T. Furthermore, we delete v_7 from C and G, which results in Fig. 1(iii). At this point, C is null and $T = \{v_1, v_3, v_4, v_5, v_6\}$ is 1-plex. In other branches, we can get the results in a similar way. Finally, the best one is selected from these results as our maximum k-plex.

3.2. Modularity optimization

We use *modularity*, one measure of the structure of networks, to evaluate the strength of division of a network into modules (also called groups, clusters or communities). It is the fraction of the edges that fall within the given communities minus the expected fraction if edges are distributed at random. Note that a node can be connected to itself when edges are distributed randomly.

Given an undirected and unweighted graph $G = (V, E)$ including a set of N nodes $V = \{v_i \,|\, i \in [1, 2, \ldots, N]\}$ connected by a set of M edges $E = \{e_{ij} \,|\, i, j \in [1, 2, \ldots, N]\}$, let $deg_G(v)$ ($deg(v)$ for simplicity) be the degree of v in the graph G, the modularity in general is computed as

$$Q(V) = \frac{1}{2M} \sum_{v_i, v_j \in V} \left(A_{ij} - \frac{deg(v_i) deg(v_j)}{2M} \right) \delta_{s_i, s_j}, \tag{1}$$

where $M = \frac{1}{2} \sum_i deg(v_i)$, $\mathbf{A} = [A_{ij}]_{N \times N}$ is the adjacency matrix of G and $A_{ij} = 1$ if $e_{ij} \in E$ and 0 otherwise. Moreover, $\frac{1}{2M} deg(v_i) deg(v_j)$ is the expected number of edges between nodes v_i and v_j when edges are distributed randomly. δ_{s_i, s_j} equals to 1 if s_i=s_j where s_i means v_i belongs to the community s, or 0 if $s_i \neq s_j$. Eq. (1) can be applicable to a network with two communities or more. The higher the modularity, the better the quality of community detection. Furthermore, let

$$B = A_{v_i v_j} - \frac{deg(v_i) deg(v_j)}{2M}, \tag{2}$$

which is the so-called *modularity matrix*. Given a network, modularity matrix is a constant value.

The modularity is always used in optimization methods for detecting communities in networks [4, 44]. However, as discussed in [10], modularity optimization methods have resolution limit in community detection, i.e., the size of every detected community is no less than $\sqrt{N}$ [2]. Therefore, it is unable to detect small communities, whose definition can be derived from [2].

Definition 3.3 (Small Community). *A group of a network G is called a small community if its number of nodes is less than $\sqrt{N}$, where N is the number of nodes in G.*

We will bend ourselves to solve this resolution limit problem while detecting small communities in Section 4.

4. Community detection algorithms

To better identify various sizes of communities, we propose a new method named *modularity optimization with k-plexes algorithm* (MOKP) to detect both large and small communities in a network. The proposed MOKP uses k-plexes to form community seeds of the network and subsequently applies the modularity optimization to find the final community seeds (i.e., communities) of the network. However, MOKP is a bit time-consuming, so we further develop another algorithm to improve its efficiency, namely *improved modularity optimization with k-plexes algorithm* (IMOKP).

4.1. MOKP algorithm

In this part, we present the MOKP algorithm that contains two main parts: one is *community seeds creation* enumerating all the community seeds in the graph, and the other one is *nodes assignment* putting the rest nodes of the graph into certain appropriate seeds.

4.1.1. Community seeds creation

To describe the creation of community seeds intuitively, we first give the definition of community seeds based on maximum k-plexes.

Definition 4.1. *Let $P(G) = \{P_1, P_2, \cdots, P_l\}$ be a set of maximum k-plexes in the graph G, $|P_m|$ is the number of nodes in the k-plex P_m and $|P_m| \geq bound$, $m = 1, 2, \ldots, l$. A subset $CS(G) \subseteq P(G)$ is called* ***community seeds****, if the element in $CS(G)$ simultaneously satisfies the following three conditions*

- *$cs_i \in P(G), i \in \{1, 2, \ldots, k\}$, $k \leq l$,*
- *$|cs_i| \geq z$, where z is an integer number such that $z \geq bound$, and*
- *every pair of k-plexes $\{cs_i, cs_j\}$ with $i \neq j$ has no common nodes.*

In our setting, z is a lower bound of the size of community seeds. It plays an important role in creating community seeds because the number of nodes in the community seeds is closely related to z. If the value of z is small, we can create many community seeds whose minimum size is small, and vice versa. Hence, it is important to explore the value of z properly. Note that we have two parameters z and *bound* in the definition, and z depends on *bound*. For implementation, we simply let z equal to *bound*, and we will discuss the impact of z on the results of community detection in Section 5.2.1.

To generate community seeds from the graph G, we first use k-plexes computing method BS [40] to find k-plexes in the network. The detailed procedures are presented in Algorithm 1. We first use the Bron-Kerbosch method (BK) [5] to find the constrained F which serves as the input of BS (line 3). We then use BS algorithm to detect the maximum k-plex ψ (we randomly choose one of them as ψ if there are many maximum k-plexes), and consider ψ as a community seed if $|\psi| \geq z$ (line 4). We may denote this k-plex as cs_1 and add it into the set of community seeds $CS = \{cs_1\}$ (line 5). After removing the nodes of CS from G (line 6), we use BK and BS again to find the maximum k-plex of the remaining network to get a community seed cs_2 and update $CS = \{cs_1, cs_2\}$. By recursively running the process, we can create community seeds $CS = \{cs_1, cs_2, \ldots, cs_m\}$, where $\{1, 2, \ldots, m\}$ are the community

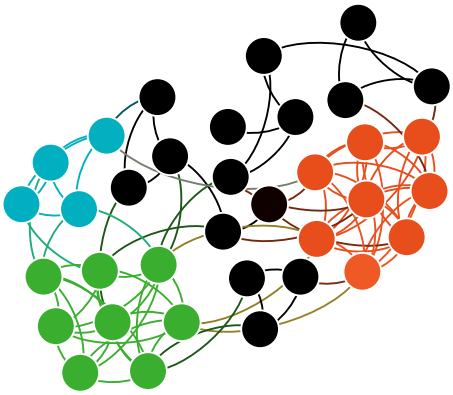


Fig. 2. Three different community seeds (nodes colored in red, green and blue, respectively) are generated after running Algorithm 1.

seed labels. Three different community seeds are intuitively shown in Fig. 2. Remark that we can also detect distinct small communities by adjusting k and z due to the benefit of k-plex method, which can help detect small communities as discussed before.

Algorithm 1 CommunitySeeds(G, z, k).

```
Initialize |ψ| = N, CS = φ;
while |ψ| ≥ z do
    use BK algorithm to find a set of nodes F where every two nodes are adjacent;
    ψ ← BS(I = (G, k, F), z);
    Update CS ← CS ∪ ψ;
    Update I ← I(G \ ψ, k, F);
end while
return CS.
```

4.1.2. Nodes assignment

After generating community seeds CS, we get a subgraph $\hat{G} = (\hat{V}, \hat{E})$, where $\hat{V}$ represents nodes of V that are not in the set CS and $\hat{E} = \{e_{ij} \,|\, i, j \in \hat{V}\}$. We aim to assign each node of $\hat{V}$ into an appropriate community seed through modularity optimization, namely the process *nodes assignment*.

Given a node $v_i \in \hat{V}$, we denote $Q_p(v_i)$ as the modularity of the network after assigning node v_i into a community seed $p \in \{1, 2, \cdots, m\}$, it computes

$$Q_p(v_i) = Q(cs_p \cup v_i) + Q(V \setminus (cs_p \cup v_i)), \tag{3}$$

where $Q(cs_p \cup v_i)$ and $Q(V \setminus (cs_p \cup v_i))$ can be derived from Eq. (1). Then, the modularity increment of node v_i is

$$\Delta Q_p(v_i) = Q_p(v_i) - Q(v_i), \tag{4}$$

where $Q(v_i) = \frac{1}{2M} \sum_{v_j \in V} \left(A_{ij} - \frac{deg(v_i)deg(v_j)}{2M} \right) \delta_{s_i, s_j}$ is the modularity before this assignment. Therefore, we have to compute the modularity increment $O(mN)$ times, which is not practi-

cal in reality. Since the structure of nodes and edges remains unchanged except the distribution of the communities in the nodes assignment process, the modularity matrix B remains a constant and δ_{s_i,s_j} varies between 0 and 1. Hence, we rewrite Eq. (4) as

$$\Delta Q_p(v_i) = \sum_{v_j \in cs_p} (A_{v_i v_j} - \frac{deg(v_i)deg(v_j)}{2M})\delta_{s_i,s_j}, \tag{5}$$

where $cs_p \subseteq CS$. For each node in $\hat{G}$, we calculate its modularity increment through only the community seeds rather than the whole network, as depicted in Eq. (5). The computation of the modularity increment now costs $O(m \max_i |cs_i|)$ times, where $|cs_i| \ll N$. Hence, we greatly improve the computational efficiency via Eq. (5). Whereafter, for the node v_i, the values $\Delta Q_1(v_i), \Delta Q_2(v_i), \cdots, \Delta Q_m(v_i)$ are generated and the maximum one is the optimal modularity increment of v_i. Suppose $\Delta Q_q^*(v_i)$ is the maximum one, and the corresponding optimal community seed is cs_q^*. Given a threshold ε, we put node v_i into the community seed q if $\Delta Q_q^*(v_i) \geq \varepsilon$, otherwise we create a new community seed and put v_i into it.

We simply describe this process in Algorithm 2. We first input m community seeds CS generated by Algorithm 1, select one node v_i in $\hat{V}$ by the alphabetical order, and next, compute the maximum modularity increment and get the corresponding community seed cs_q^* (line 2). We put v_i into cs_q^* if the modularity increment is larger than ε (lines 3-4). In contrary, we generate a new community seed cs_{m+1} to put v_i into it (line 6), and update the set of community seeds $CS = \{cs_1, cs_2, \cdots, cs_m, cs_{m+1}\}$ for assigning other remaining nodes of $\hat{V}$ (line 7). We repeatedly utilize Algorithm 2 to assign all the nodes in $\hat{V}$ into some proper community seeds, and finally we find the community result (the set CS) of the whole network.

Algorithm 2 FindCommunity(v_i, CS, ε).

1: Initialize $m = |CS|$;
2: $cs_q^* \leftarrow \arg\max_{cs_p \in CS} \sum_{v_j \in cs_p} (A_{v_i v_j} - \frac{deg(v_i)deg(v_j)}{2M})\delta_{s_i,s_j}$;
3: **if** $\Delta Q_q^*(v_i) \geq \varepsilon$ **then**
4: Update $cs_q^* \leftarrow cs_q^* \cup v_i$;
5: **else**
6: $cs_{m+1} \leftarrow v_i$;
7: Update $CS \leftarrow CS \cup cs_{m+1}$;
8: **end if**
9: **return** CS.

We summarize MOKP in Algorithm 3. First, the empty set CS is initialized to store the community seeds (line 1). Next, we generate m community seeds by Algorithm 1 (line 2). Finally, we remove CS from V (line 3) and put the remaining nodes $\hat{V}$ into some proper community seeds by Algorithm 2 (lines 4-6). Note that the parameter k is highly related to datasets, which will be discussed in Section 5.2.2.

4.2. IMOKP algorithm

Operating the MOKP algorithm is a challenging task in terms of its efficiency and complexity, which reflects in community seeds creation and nodes assignment as well. To accel-

Algorithm 3 MOKP algorithm.

Input: an unweighted and undirected graph $G(V, E)$, the smallest size z of the community seeds, the modularity increment threshold ε and parameter k;
Output: the detected communities CS;

```
1: Initialize CS = φ;
2: CS ← CommunitySeeds(G, z, k);
3: V̂ ← V \ CS;
4: for each node v_i in V̂ do
5:     CS ← FindCommunity(v_i, CS, ε);
6: end for
7: return CS.
```

erate the computing speed of MOKP, we improve MOKP from two orientations, one is *scale reduction* and the other one is *nodes assignment adjustment*.

Algorithm 4 KCoreDetect(G, z, k).

```
1: for each node v with deg(v) ≤ (z − k) do
2:     V ← V \ v;
3:     for node u ∈ N_G(v) do
4:         G(V, E) ← G(V, E \ (u, v));
5:         deg(u) ← deg_G(u)
6:     end for
7: end for
8: return G.
```

Scale reduction is proposed to reduce the scale of a network since the computational time of the MOKP algorithm increases exponentially as its scale increases. Hence, we try to reduce the scale of the network before the process of detecting k-plexes. Note that it is rational to detect k-plexes from $(z-k)$-core graph because any k-plex with size bigger than z is an induced graph of the $(z-k)$-core graph of a given network. There are some $(z-k)$-core graph detection methods, e.g., Batagelj and Zaversnik [3] and Xin [27]. We resort to the latter method to detect the $(z-k)$-core graph due to its simplicity and efficiency. Since the complexity of computing $(z-k)$-core graph is linear, our improved method could save much computational time. The details are presented in Algorithm 4. Let $N_G(v)$ be the set of neighbors of v in a graph G. When there are nodes whose degree is less than $z-k$ (line 1), we remove such nodes from V (line 2), the corresponding edges and then update the degrees of these nodes' neighbors (lines 3-6).

To further accelerate MOKP, nodes assignment adjustment is proposed to adjust rules of nodes assignment in two ways. On one hand, we directly label a node as the community seed c when operating Algorithm 2 under the situation that most of its neighbours are in the same community seed c. The underlying reason is that nodes in a network are deeply attracted by their neighbors. Hence, the computation of modularity increment can be largely reduced if we have already know enough labels of nodes. On the other hand, we modify the assignment order from alphabetical order into descending order of degree. The detailed

Algorithm 5 AssignmentAjustment($\hat{G}$, CS, ε).

1: **while** $\hat{G}$ **do**
2: $v_i = \arg\max_v deg_{\hat{G}}(v)$
3: $L \leftarrow int[N] = 0$;
4: **for** $u \in N_G(v_i)$ **do**
5: $L[l(u)] \leftarrow L[l(u)] + 1$;
6: **end for**
7: $L[r] \leftarrow max(L)$;
8: **if** $L[r] \geq p\,|N_G(v_i)|$ **then**
9: $l(v_i) \leftarrow r$;
10: Update $cs_r \longleftarrow cs_r \cup v$ and $CS \longleftarrow CS \cup cs_r$;
11: **else**
12: $CS \leftarrow FindCommunity(v_i, CS, \varepsilon)$;
13: **end if**
14: $\hat{G}(\hat{V}, \hat{E}) \leftarrow \hat{G}(\hat{V} \setminus v_i, \hat{E} \setminus (u, v_i))$ where $u \in N_{\hat{G}}(v_i)$
15: **end while**
16: **return** CS.

procedures are described in Algorithm 5. For nodes in $\hat{V}$, we select a single node v_i with the current largest degree (line 2). Let $l(u) \in \{1, 2, \cdots, m\}$ be the label (also represents node's label) of a community seed that node v_u belongs to, and $L[q] \subseteq L$ denotes the number of nodes whose labels are q. We first initialize L to a list with length N of which each item is 0 (line 3), compute the community label set L for node v_i by counting the number of each community label in its neighbours (lines 4-6) and find the maximum number $L[r]$ of L (line 7). If $L[r] \geq p\,|N_G(v_i)|$, where p is a threshold percentage, we put node v_i into the community seed r and update the community seeds set CS (lines 8-10). Otherwise, we label this node's community using Algorithm 2 (line 12). After removing the node v_i and its corresponding edges from $\hat{G}$ (line 14), we run these steps again to assign the remaining nodes in $\hat{G}$. The good side of operating these steps several times is that nodes with smaller degree could naturally be divided into the existing communities without computing modularity increment.

To summarize the previous discussions, we present the IMOKP algorithm in Algorithm 6. We first initialize a label $l(n)$ for each node n (line 1). Next, we detect the $(z-k)$-core graph of the given network (line 2). Based on the structure of the graph, we create the community seeds (line 3). For each community seed, we randomly select a node of it, and use this node's label to update the labels of the other nodes in order to uniform the whole labels in this community seed (lines 4-9). Hence, the label of the selected node becomes the label of the community seed. Finally, we assign the remaining nodes of $\hat{G}$ by Algorithm 5 (lines 10-11).

5. Empirical study

In this section, we conduct some experiments to compare the effectiveness of the MOKP algorithm with several state-of-the-art methods on five different networks, and evaluate the

Algorithm 6 IMOKP algorithm.

Input: an unweighted and undirected graph $G(V, E)$, the smallest size z of the community seed, the modularity increment threshold ε, a threshold percentage p and parameter k.

Output: the detected communities CS.

Initialize $CS = \phi$, $l(n) = n, n = 1, 2, \cdots, N$;
$G_{core} \leftarrow$ KCoreDetect(G, z, k);
$CS \leftarrow$ CommunitySeeds(G_{core}, z, k);
for each seed φ in CS **do**
  randomly select a node v of φ;
  for each node j in φ **do**
    $l(j) \leftarrow l(v)$;
  end for
end for
$\hat{G} \leftarrow G \setminus CS$;
$CS \leftarrow$ AssignmentAjustment$(\hat{G}, CS, \varepsilon)$;
return CS.

accuracy of small communities detected by MOKP using our proposed metric on two real-world networks. We further demonstrate the efficiency and effectiveness of the IMOKP algorithm on seven various networks. Finally, we illustrate the necessity of detecting communities and even small communities on a real-world network.

We implement our algorithms in the platform of MATLAB R2016b and PYTHON 2.7, and all the computations are conducted in an Intel Core CPU 3.40GHz 16GB RAM computer. We set the parameter ε to be 0.015 for the MOKP and IMOKP algorithms, and p to be 0.5 by majority rule for the IMOKP algorithm.

5.1. Datasets and metrics

We choose five real-world networks[1] to measure the performance of the proposed MOKP, i.e., Zarachy [46], Football [11], Email-Eu-Core [45], com-DBLP [42], com-Live Journal [42]. For convenience sake, we sometimes abbreviate com-DBLP and com-Live Journal to DBLP and Live Journal, respectively. We select four more synthetic networks [18] to compare the performance of our algorithms, which are constructed by the Lancichinetti-Fortunato-Radicchi (LFR) method[2]. These nine networks provide ground-truth community labels that can be used to assess the accuracy of detected communities. Table 1 summarizes these networks. The total number of nodes, edges and ground-truth communities (C^*) for each network are presented. In Table 1, d is the averaged degree, λ denotes the exponent for the degree distribution, β is the exponent for the community size distribution and μ represents mixing parameter. Due to the large scale of DBLP and Live Journal, we randomly select a set of nodes and edges to form a new sub-network to do experiments. We further test the

[1] http://snap.stanford.edu/
[2] http://santo.fortunato.googlepages.com/benchmark.tgz

performance of the MOKP algorithm on the DM-LC network[3,4] [29, 7] to assess the necessity of detecting communities and even small communities. It is a protein-protein interaction network with 658 nodes and 1,100 edges, where a node represents a gene. However, this network does not have the ground-truth community labels.

Table 1 Real-World networks and LFR networks.

Datasets	**Nodes**	**Edges**	**C***	**Description**
Zarachy	34	78	2	a social network which reflects the friendship between members from a karate club at a US university in the 1970s
Football	115	613	16	a network of American football games between Division IA colleges during regular season Fall 2000
Email-Eu-Core	1005	16706	41	a network generated from the email communication data of a large European research institution
DBLP (Small)	815	2045	27	a collaboration network derived from the DBLP Computer Science Bibliography
Live Journal	1018	7149	36	a friendship social network is constructed from a free on-line blogging community
DBLP (Large)	10029	29400	326	a collaboration network derived from the DBLP Computer Science Bibliography
LFR-1	1000	10232	37	$d = 10$, $\lambda = 1$, $\beta = 1$, $\mu = 0.5$
LFR-2	2000	20500	90	$d = 20$, $\lambda = 1$, $\beta = 1$, $\mu = 0.5$
LFR-3	1000	20770	41	$d = 10$, $\lambda = 1$, $\beta = 1$, $\mu = 1$
LFR-4	2000	40322	87	$d = 20$, $\lambda = 1$, $\beta = 1$, $\mu = 1$

We adopt three widely used metrics to quantify the accuracy of detected communities C. They are normalized mutual information (NMI) [20], F-score similarity and Jaccard similarity [43], respectively. These three metrics vary from 0 to 1, and 1 indicates the perfect quality of the detected communities.

- NMI

$$\mathrm{NMI}(C^*, C) = \frac{1}{\max(H(C^*), H(C))} \sum_{C_i^*, C_j} p(C_i^*, C_j) \log \frac{p(C_i^*, C_j)}{p(C_i^*)p(C_j)},$$

  where $H(C)$ is the mutual information of C.

- F-score similarity (Jaccard similarity)

$$\mathrm{F}(C^*, C) = \sum_{C_i^* \in C^*} \frac{\max_{C_j \in C} \delta(C^*, C)}{2|C|} + \sum_{C_j \in C} \frac{\max_{C_i^* \in C^*} \delta(C^*, C)}{2|C^*|},$$

[3]https://www.inetbio.org/wormnet/
[4]http://networkrepository.com/bio-DM-LC.php

where $\delta(C^*, C)$ measures the similarity between C^* and C, which is set to be F-score similarity (Jaccard similarity).

It is essential to evaluate the quality of detected small communities. Fortunato and Barthlemy [10] and Ailon et al. [2] mentioned small communities in their papers, while they did not give methods to evaluate the small communities. Shao et al. [33] evaluated small communities using three normal metrics, i.e., NMI, adjusted rand index (ARI) and cluster purity. However, these metrics are not the tailor for small communities. And there are less work about small communities besides the above. Thus, to measure the quality of detected small communities on the behalf of ground-truth S^*, we design a comprehensive index, *small community level* (SCL), inspired by Jaccard similarity [12]. Suppose that S denotes the detected small communities, let S_i be the i-th small community of S and S_i^* be the i-th small community of S^*, the small community level is defined as

$$\mathrm{SCL}(S^*) = \frac{1}{\Omega^*} \sum_i \mathrm{SCL}(S_i^*),$$

$$\mathrm{SCL}(S_i^*) = \frac{1}{\sqrt{|\Omega^* - \Omega|}} \sum_j \frac{|S_i^* \cap S_j|}{|S_i^* \cup S_j|} \frac{|S_j|}{\sum_j |S_j|},$$

where Ω^* and Ω are the total number of small communities in S^* and S respectively, and symbol $|\cdot|$ denotes the number of nodes in a community. Given the small community S_i^*, $\sum_j \frac{|S_i^* \cap S_j|}{|S_i^* \cup S_j|}$ is the total similarity between S_i^* and the detected small community S_j, for all j, each of which computes their Jaccard similarity. Moreover, $\sum_j \frac{|S_j|}{\sum_j |S_j|}$ denotes the total percentage of every detected small community among them and $\frac{1}{\sqrt{|\Omega^* - \Omega|}}$ is the punishment. The larger the value of SCL, the better the quality of the detected small communities. Note that there may exist the case that some small communities are detected together as one single community, then the value SCL should not be large, and therefore the punishment term is needed to control its value. The larger value of the punishment means the number of detected small communities is closer to the ground-truth. The punishment achieves 1 if Ω^* equals to Ω.

5.2. Parameter sensitivity analysis

Our algorithms require the smallest size z of community seeds and parameter k as part of the inputs. We explore the sensitivity of these two parameters in the MOKP algorithm on five real-world networks in terms of NMI, F-score and Jaccard similarity.

5.2.1. Sensitivity analysis on z

To analyze the effect of z, we set $k = 1$ since each community seed in this circumstance will be a clique which is intuitively similar to the distinct small community. As discussed in Section 4.1.1, the value of z has an impact on community seeds creation so as to affect the detected communities. Consequently, we vary z within 10 (there is no apparent improvement in resolution limit if z is larger in our experiments) to report its effect on the performance of MOKP. The results are presented in Fig. 3. From Fig. 3, we know that the performance of MOKP is sensitive to z as measured by any metric. No matter which metric is used, MOKP

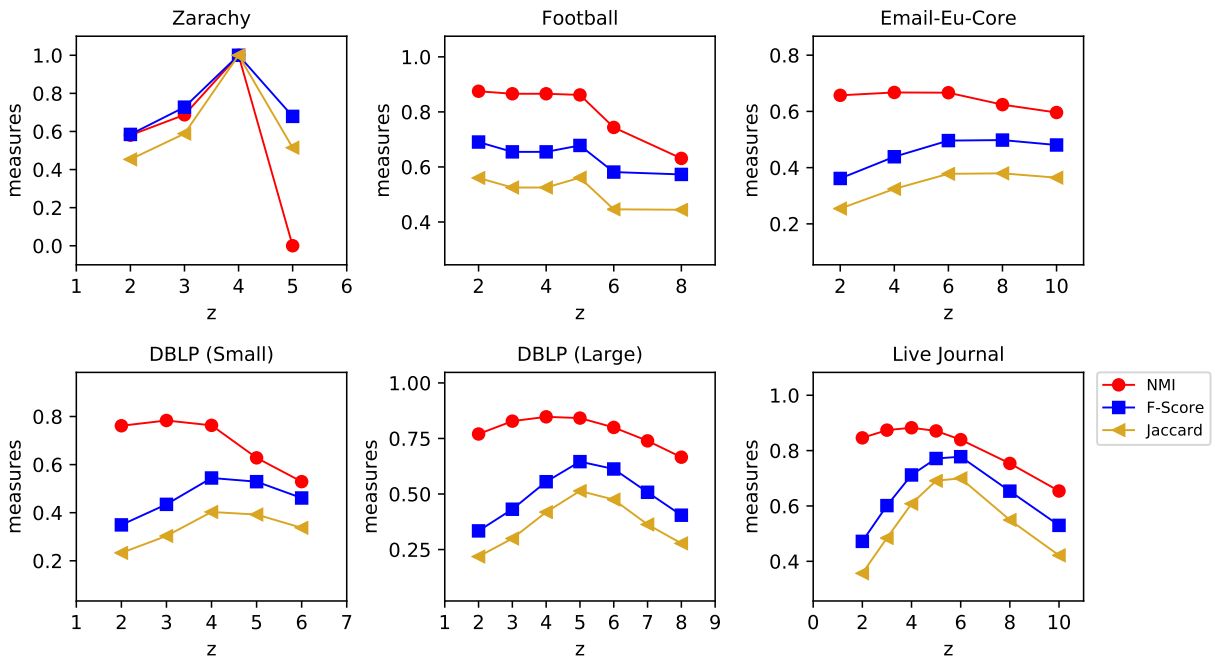


Fig. 3. Performance of MOKP with various z and $k = 1$.

achieves its best performance all on the same z on the Zarachy and Football networks, i.e., z equals to 4 and 2, respectively. However, this is not the case on the other three networks, i.e., MOKP performs best on the same z via F-score and Jaccard similarity while it performs best on another z via the metric NMI. Hence, we adopt the majority voting method to select the value z as 6, 4, 5 and 6 on the Email-Eu-Core, DBLP (Small), DBLP (Large) and Live Journal networks, respectively.

5.2.2. Sensitivity analysis on k

In this part, we use the values of z that determined above and analyze the effect of k on the detected communities. We run the MOKP algorithm on the Football, Email-Eu-Core, DBLP and Live Journal networks, varying k from 1 to 7. Here we do not test the algorithm on the Zarachy network because the values of the three metrics have achieved their peak (i.e., 1) when $k = 1$ and the size of the Zarachy network is too small to do this experiment. As shown in Fig. 4, the performance of MOKP is relatively insensitive to k on the Email-Eu-Core, DBLP (Small), DBLP (Large) and Live Journal networks, and no matter which metric is, the best performance is reached when $k = 6, 4, 5, 1$ respectively. However, the effect of k is obvious when running MOKP on the Football network, and thus we can choose $k = 4$ on this network. In the following tests, we will choose the values of z and k that MOKP performs best on these five networks, respectively.

5.3. Performance of MOKP

We compare the MOKP algorithm to the Spectral algorithm (SP) [24], the Fast algorithm (FN) [23], the Label Propagation method (LPA) [28] and the FastUnfolding algorithm (Louvain) [4] on the five real-world networks. The comparison results are summarized in Table 2, where the best ones are marked in bold. Table 2 shows that MOKP outperforms

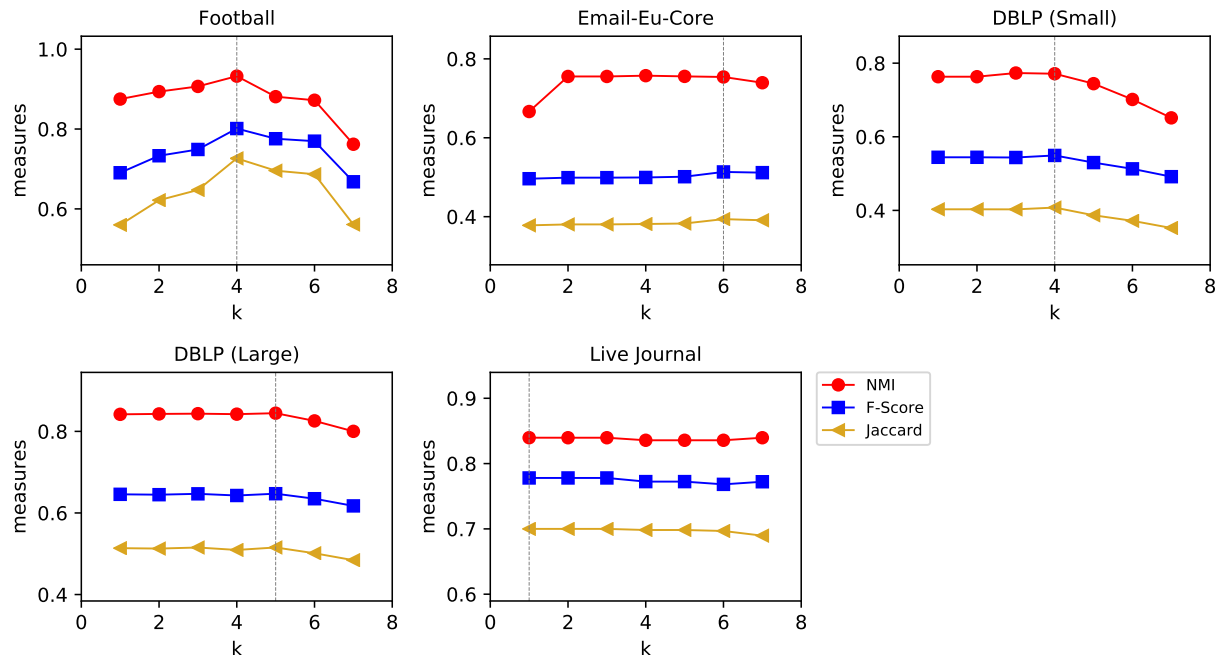


Fig. 4. Performance of MOKP with fixed z and various k.

Table 2 Accuracy of detected communities on real-world networks.

Method	Zarachy			Football			Email-Eu-Core		
	NMI	F-score	Jaccard	NMI	F-score	Jaccard	NMI	F-score	Jaccard
SP	0.2258	0.4480	0.3062	0.7145	0.2434	0.1407	0.5760	0.1537	0.0864
FN	0.2322	0.3587	0.2356	0.6530	0.4428	0.3158	0.6036	0.2272	0.1537
LPA	0.4002	0.5930	0.4593	0.8588	0.7524	0.6835	0.5561	0.3627	0.2510
Louvain	0.7435	0.7430	0.6441	0.8693	0.7424	0.6621	0.5569	0.2034	0.1416
MOKP	**1.0000**	**1.0000**	**1.0000**	**0.9324**	**0.8012**	**0.7265**	**0.7541**	**0.5134**	**0.3938**
Method	**com-DBLP (Small)**			**com-DBLP (Large)**			**com-Live Journal**		
	NMI	F-score	Jaccard	NMI	F-score	Jaccard	NMI	F-score	Jaccard
SP	0.7233	0.2908	0.1861	0.8101	0.2775	0.1746	0.7768	0.3505	0.2388
FN	0.8294	0.5187	0.3907	0.8418	0.4869	0.3662	0.9372	0.6449	0.5827
LPA	0.8286	0.5024	0.4003	**0.8796**	0.5412	0.4169	0.9066	0.5793	0.5000
Louvain	**0.8480**	0.5254	0.3899	0.8003	0.5326	0.4314	**0.9478**	0.7093	0.6912
MOKP	0.7713	**0.5493**	**0.4077**	0.8446	**0.6472**	**0.5154**	0.8396	**0.7780**	**0.7000**

most of the other methods except the worse accuracy in terms of NMI on the DBLP and Live Journal networks. Specifically, the MOKP algorithm absolutely overwhelms other four algorithms on the Zarachy network, since the values of NMI, F-score and Jaccard similarity are all achieve 1 using our algorithm while they are less than 0.8 using others. To visually view part of the results, we show the communities detected by our MOKP algorithm on the Zarachy and Football networks in Figs. 5 and 6, respectively. In these figures, different numbers indicate different ground-truth communities, and different colors represent different detected communities. We can see that the two communities on the Zarachy network are completely detected and most of the detected communities on the American Football network match with the ground-truth very well.

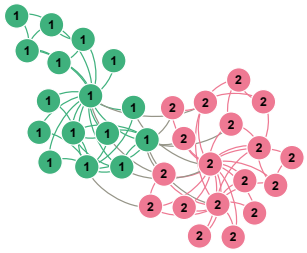


Fig. 5. Detected communities on the Zarachy network.

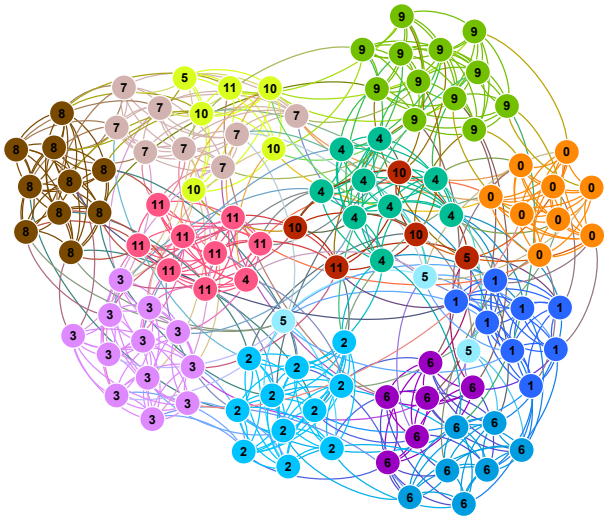
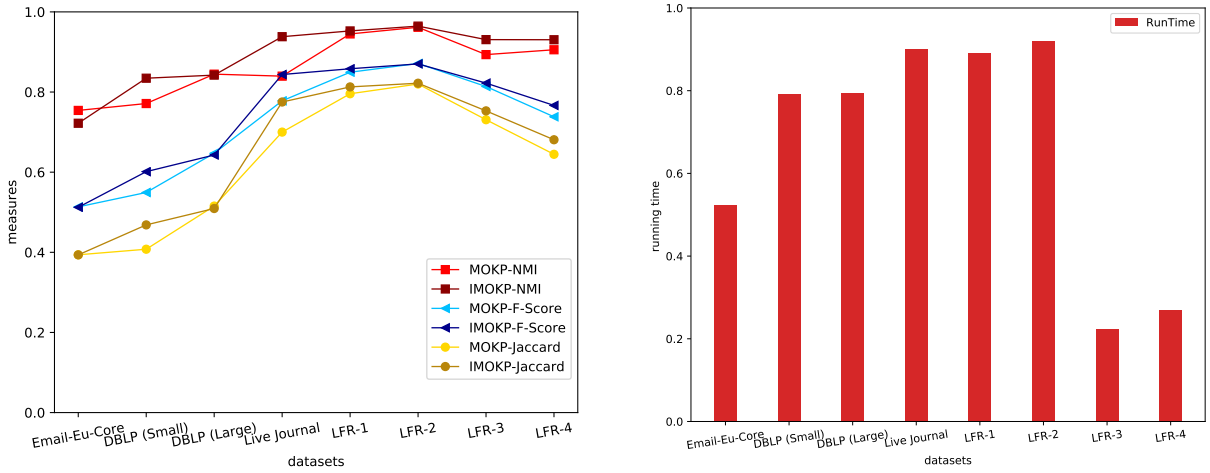


Fig. 6. Detected communities on the American Football network.

5.4. Performance of IMOKP

We further use three real-world networks (Email-Eu-Core, DBLP and Live Journal) and four synthetic networks to evaluate the efficiency of the IMOKP algorithm. Here we set z and k to be the values that MOKP performs best for the three real-world network discussed in Section 5.2. While for the four synthetic networks, we set z to be 4, 4, 6, 5 and k to be 4, 4, 6, 5, respectively. The way of determining these values is the same as that in Section 5.2, which we will not discuss in detail here. The comparison results between MOKP and IMOKP are shown in Fig. 7. For the accuracy, Fig. 7(a) shows that IMOKP almost performs better than MOKP measured by any metric on all the datasets, and it is worth mentioning that the NMI of IMOKP reaches 0.9381 while it is 0.8396 for MOKP on Live Journal, the F-score of IMOKP is 0.6013 and that of MOKP is 0.5493 on DBLP. For the running time, the ratio of saved computational effort is presented in Fig. 7(b), which is the difference of the running time between IMOKP and MOKP divides the running time of MOKP, ranging from 0 to 1. Fig. 7(b) indicates that the IMOKP algorithm significantly reduces community detection time, e.g., up to 22% improvement over MOKP on LFR-3 and up to 92% improvement on LFR-2.



(a) Accuracy (b) Reduction of running time

Fig. 7. Comparison between MOKP and IMOKP in terms of accuracy and running time.

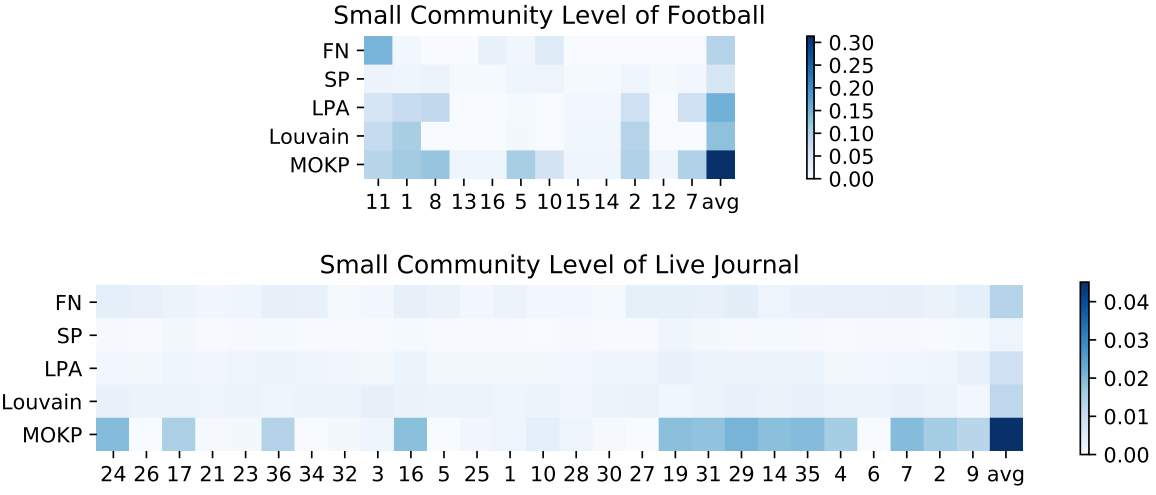


Fig. 8. Comparison of small community level of detected communities.

5.5. Small community analysis

We use our newly defined index SCL to evaluate the detected small communities. Taking two networks (Football and Live Journal) for example, Fig. 8 demonstrates the small community level of each small community in detail, where the darker the color, the larger the value of SCL. Horizontal axis in this figure means the labels of small communities. In particular, the symbol "avg" denotes the average value of SCL, which is equal to SCL(S^*). We simultaneously multiply their corresponding averages by 5 for the five algorithms for distinct display. As observed in Fig. 8, the proposed MOKP outperforms other four algorithms, which indicates the rationality of small communities detected by MOKP.

5.6. Analysis of the content of communities

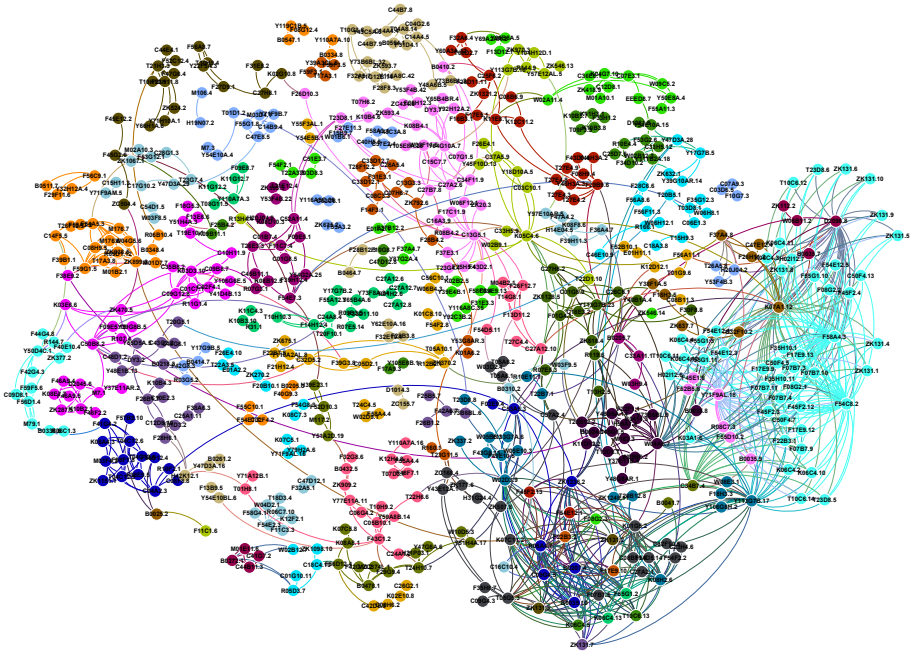
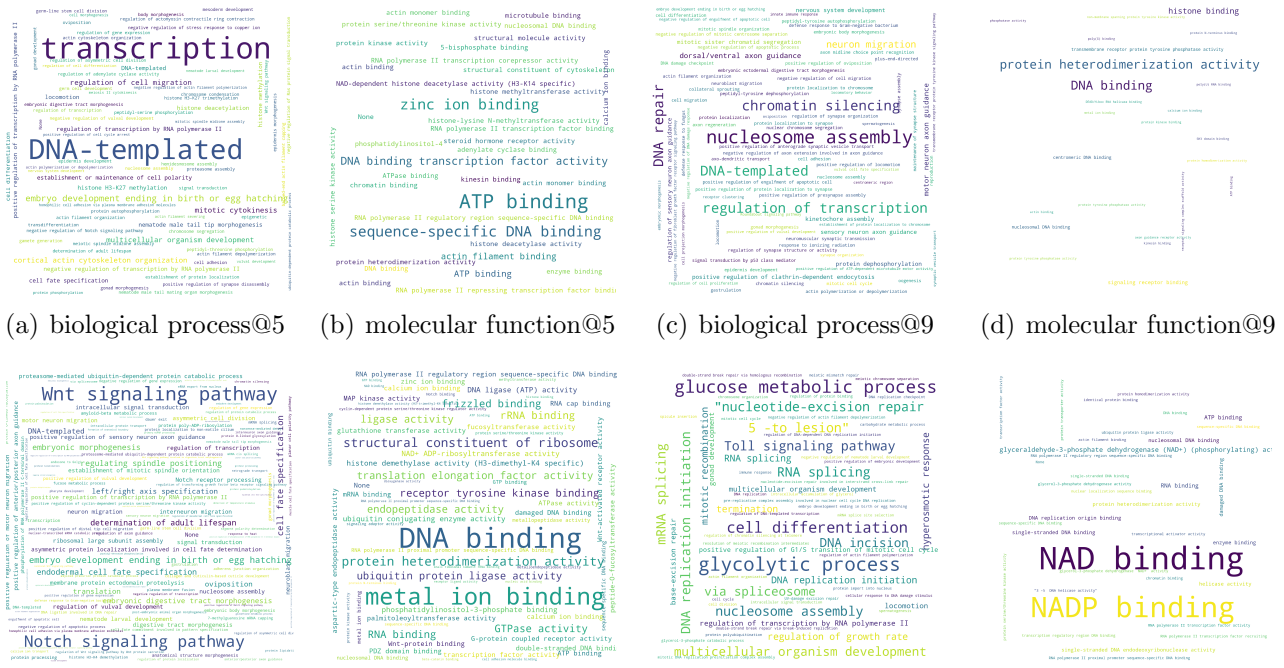


Fig. 9. Detected communities of DM-LC network. Labels indicate related gene names.

We tend to illustrate the necessity of detecting communities and even small communities from two parts. One is to check whether the characteristics of each node in the same community are similar. We know that members from the same community tend to exhibit the same characteristics in the real world, hence nodes with similar characteristics could constitute a community. The other one is to analyze whether the content of each small community is unique, in order to assure the necessity of small communities in the real world.

Each protein associated with a gene has two functions, i.e., molecular function and biological process, which refer to attributes in word clouds. The size of an attribute indicates the number of times it appears in a community, which may imply the characteristics of the community. We present all the detected communities in Fig. 9 and analyze four of them by word clouds in Fig. 10. Fig. 9 demonstrates that the DM-LC network detects 70 communities, among which the largest community contains 60 nodes and the smallest one contains 2 nodes. This phenomenon indicates that our algorithm can detect both large and small



(a) biological process@5 (b) molecular function@5 (c) biological process@9 (d) molecular function@9

(e) biological process@10 (f) molecular function@10 (g) biological process@19 (h) molecular function@19

Fig. 10. Word clouds of protein functions of detected communities, including biological process@n and molecular function@n, where n is the community label.

communities. We can also intuitively see from Fig. 9 that the nodes in the same community are dense while sparse in different communities. In Fig. 10(a), we observe that attributes *DNA-templated* and *transcription* are symbols of biological process@5, which is consistent with the fact that *Transcription* takes DNA as a template. Besides, Fig. 10(b) shows that most genes have attributes *ATP binding*, *zinc ion binding* and *sequence-specific DNA binding*, which indicates the feature *binding* of molecular function for the community 5. Similarly, in Fig. 10(c-h), the community 9, 10 and 19 show the features of *nucleosome*, *signaling pathway* and *carbohydrate metabolism* in biological process respectively, and also demonstrate the features of *protein*, *binding* and *NAD* in molecular function respectively. Communities 5 and 19 (marked in medium blue and dark green respectively in Fig. 9) are small communities, where the community 19 is related to the attribute NAD. NAD is of great importance in the respiration and photosynthesis, one of which is essential for any creature. Most of NADs are merely found in this community, which indicates the importance of detecting small communities.

6. Conclusion and future work

In this paper, we propose the MOKP algorithm, an algorithm combining the k-plex and modularity optimization, to solve the resolution limit problem in modularity optimization methods. It detects communities by creating the community seeds and then assigning the remaining nodes according to the modularity. With the benefit of the technique of k-plex, MOKP is able to detect both large and small communities. We further propose the IMOKP

algorithm to improve the MOKP algorithm, conducting scale reduction to the community seeds creation and adjusting rules of nodes assignment. Three common metrics are used to quantify the accuracy of detected communities and one newly defined index SCL is used to evaluate the quality of detected small communities on the behalf of ground-truth communities. Results from extensive experiments reveal best performance of the MOKP algorithm compared with several state-of-the-art algorithms on various networks. The performance of IMOKP is almost better than MOKP in term of accuracy and it greatly saves much computational time, both testing on seven different networks. We also analyze the necessity of detecting communities and even small communities on a protein-protein interaction network. In the future, we may develop more effective techniques to compute the k-plex, which is an important part of our algorithms. Moreover, we may give a mechanism to select an appropriate k for computing each k-plex since it directly affects each community seed.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grants No. 61772442, 61836005 and 11671335).

References

[1] H. A. Abdelbary, A. M. Elkorany, and R. Bahgat. Utilizing deep learning for content-based community detection. In *Science and Information Conference*, pages 777–784, 2014.

[2] N. Ailon, Y. Chen, and H. Xu. Breaking the small cluster barrier of graph clustering. In *International Conference on Machine Learning*, pages 995–1003, 2013.

[3] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[4] V. D. Blondel, J. L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics*, 2008(10):155–168, 2008.

[5] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the Acm*, 16(9):575–576, 1973.

[6] J. Chen and B. Yuan. Detecting functional modules in the yeast protein-protein interaction network. *Bioinformatics*, 22(18):2283–2290, 2006.

[7] A. Cho, J. Shin, S. Hwang, C. Kim, H. Shim, H. Kim, H. Kim, and I. Lee. Wormnet v3: a network-assisted hypothesis-generating server for caenorhabditis elegans. *Nucleic Acids Research*, 42(W1):W76–W82, 2014.

[8] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E Statistical Nonlinear and Soft Matter Physics*, 70(2):066111, 2004.

[9] H. Feng, J. Tian, H. J. Wang, and M. Li. Personalized recommendations based on time-weighted overlapping community detection. *Information and Management*, 52(7):789–800, 2015.

[10] S. Fortunato and M. Barthlemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences of the United States of America*, 104(1):36–41, 2007.

[11] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, 2002.

[12] P. Jaccard. The distribution of the flora in the apline zone. *New Phytologist*, 11(2):37–50, 1912.

[13] M. Jebabli, H. Cherifi, C. Cherifi, and A. Hamouda. Community detection algorithm evaluation with ground-truth data. *Physica A: Statistical Mechanics and Its Applications*, 492:651–706, 2017.

[14] D. Jin, Z. Chen, D. He, and W. Zhang. Modeling with node degree preservation can accurately find communities. *New Media and Society*, 18(7):1293–1309, 2016.

[15] D. Jin, M. Ge, Z. Li, W. Lu, D. He, and F. Fogelmansoulie. Using deep learning for community discovery in social networks. In *IEEE International Conference on TOOLS with Artificial Intelligence*, pages 160–167, 2017.

[16] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[17] M. Kozdoba and S. Mannor. Community detection via measure space embedding. In *Advances in Neural Information Processing Systems*, pages 2890–2898, 2015.

[18] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E Statistical Nonlinear and Soft Matter Physics*, 78(4 Pt 2):046110, 2008.

[19] Y. Li, C. Wu, and Z. Wang. An information-theoretic approach for detecting communities in networks. *Quality and Quantity*, 49(4):1719–1733, 2015.

[20] L. Liu, L. Xu, Z. Wangy, and E. Chen. Community detection based on structure and content: A content propagation perspective. In *IEEE International Conference on Data Mining*, pages 271–280, 2016.

[21] R. D. Luce and A. D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

[22] F. Mcsherry. Spectral partitioning of random graphs. In *IEEE Symposium on Foundations of Computer Science*, page 529, 2001.

[23] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 69(6 Pt 2):066133, 2004.

[24] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[25] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E Statistical Nonlinear and Soft Matter Physics*, 69(2):026113, 2004.

[26] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

[27] X. Pei. *The algorithm to enumerate maximal k-plexes in network and the resume mining of community in network*. PhD thesis, Beijing University of Posts and Telecommunications, 2008.

[28] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E Statistical Nonlinear and Soft Matter Physics*, 76(3):036106, 2007.

[29] R. Rossi and N. Ahmed. The network data repository with interactive graph analytics and visualization. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 4292–4293, 2015.

[30] M. Rosvall and C. T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. *Proceedings of the National Academy of Sciences of the United States of America*, 104(18):7327–7331, 2007.

[31] B. Saoud and A. Moussaoui. Node similarity and modularity for finding communities in networks. *Physica A Statistical Mechanics and Its Applications*, 492:1958–1966, 2018.

[32] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6(1):139–154, 1978.

[33] J. Shao, Z. Han, Q. Yang, and T. Zhou. Community detection based on distance dynamics. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1075–1084, 2015.

[34] H. W. Shen. Detecting the overlapping and hierarchical community structure in networks. In *Community Structure of Complex Networks*, pages 19–44. Springer, 2013.

[35] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans.pattern Anal.mach.intell*, 22(8):888–905, 2000.

[36] H. A. Simon. The architecture of complexity: Hierarchic systems. *Proceedings of the American Philosophical Society*, 106(4):183 – 216, 1962.

[37] M. Tasgin and H. O. Bingol. Community detection using preference networks. *Physica A Statistical Mechanics and Its Applications*, 495:126–136, 2017.

[38] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.

[39] Z. H. Wu, Y. F. Lin, S. Gregory, H. Y. Wan, and S. F. Tian. Balanced multi-label propagation for overlapping community detection in social networks. *Journal of Computer Science and Technology*, 27(3):468–479, 2012.

[40] M. Xiao, W. Lin, Y. Dai, and Y. Zeng. A fast algorithm to compute maximum k-plexes in social network analysis. In *AAAI Conference on Artificial Intelligence, North America*, pages 919–925, 2017.

[41] J. Xie, B. K. Szymanski, and X. Liu. Slpa: uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *IEEE International Conference on Data Mining Workshops*, pages 344–349, 2012.

[42] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *IEEE International Conference on Data Mining*, pages 1–8, 2012.

[43] J. Yang, J. Mcauley, and J. Leskovec. Community detection in networks with node attributes. In *IEEE International Conference on Data Mining*, pages 1151–1156, 2013.

[44] L. Yang, X. Cao, D. He, C. Wang, X. Wang, and W. Zhang. Modularity based community detection with deep learning. In *International Joint Conference on Artificial Intelligence*, pages 2252–2258, 2016.

[45] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich. Local higher-order graph clustering. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 555–564, 2017.

[46] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.