Knowl Inf Syst (2019) 59:1–31
https://doi.org/10.1007/s10115-018-1209-7

REGULAR PAPER

Fast detection of community structures using graph traversal in social networks

Partha Basuchowdhuri[1] · **Satyaki Sikdar**[2] · **Varsha Nagarajan**[1] · **Khusbu Mishra**[1] · **Surabhi Gupta**[1] · **Subhashis Majumder**[1]

Received: 19 March 2016 / Revised: 26 December 2017 / Accepted: 6 May 2018 /
Published online: 31 May 2018

Abstract Finding community structures in social networks is considered to be a challenging task as many of the proposed algorithms are computationally expensive and does not scale well for large graphs. Most of the community detection algorithms proposed till date are unsuitable for applications that would require detection of communities in real time, especially for massive networks. The Louvain method, which uses *modularity maximization* to detect clusters, is usually considered to be one of the fastest community detection algorithms even without any provable bound on its running time. We propose a novel graph traversal-based community detection framework, which not only runs faster than the Louvain method but also generates clusters of better quality for most of the benchmark datasets. We show that our algorithms run in $O(|V| + |E|)$ time to create an initial cover before using modularity maximization to get the final cover.

Keywords Community detection · Influenced Neighbor Score · Brokers · Community nodes · Communities

1 Introduction

Networks can be realized by a graph data structure defined by $G = (V, E)$, where V denotes the vertex set, E denotes the edge set and $n = |V|, m = |E|$. These networks exhibit certain implicit characteristics like community structures. *Communities* in a network represent groups of vertices having dense intra-connections but sparse inter-connections. In this paper, we use the terms *cluster* and *community* interchangeably. In a social network, a basic

Satyaki Sikdar: The work was done when the author was at Heritage Institute of Technology.

✉ Partha Basuchowdhuri
parthabasu.chowdhuri@heritageit.edu

[1] Department of Computer Science and Engineering, Heritage Institute of Technology, Kolkata, WB, India

[2] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA

assumption is that the shortest paths are used to propagate information. Communities are connected via nodes, termed as *brokers*, that are present on a large number of shortest paths among pairs of nodes in the network and hence, control the spread of information between communities. These nodes mark the boundary of communities. In our algorithm, we find such broker nodes that help to reach a new community and then spread the information among its members. In the process, we identify the nodes that are influenced by the brokers and also exhibit a high probability of belonging to the same community.

Most of the existing community detection algorithms involve lots of computations and hence are time-consuming. In 2002, in one of the early attempts to find communities from social networks, Girvan and Newman [9] proposed an algorithm to detect hierarchical communities using repeated occurrence of an edge in all pair shortest paths as a metric. It was termed as the *brokerage* value of an edge. This is one of the first papers that points out the significance of finding broker nodes or edges for detecting communities in social networks. This algorithm has a very high computational demand even for the sparse networks. Likewise, most of the other community detection algorithms are also computationally expensive and some of them suffer from the major disadvantage of detecting only disjoint communities. However, real-world networks are generally found to have overlapping communities. Some algorithms take the number of communities required as an input, but, it may not always be possible to estimate the number of clusters without prior analysis of the network, if someone wants to find the best set of clusters.

In our paper, we try to overcome most of these existing drawbacks found in the known algorithms [8]. Our algorithm uses popular traversal techniques like depth-first traversal and breadth-first traversal and proves to perform considerably better and faster than other existing algorithms.

2 Prior works

In this section, we mainly focus on the community detection algorithms known for having lesser running time. OPTICS [1], a density-based clustering works well with the benchmark datasets. It chooses an outlier point to start the algorithm and then traverses through the points to draw a plot to mark the denser and sparser regions and thereby detecting the clusters. We have guided our algorithm in a similar manner using a graph structure but have achieved a linear running time in the process. An overlapping community detection algorithm, ONDOCS [4], takes help of visualization like OPTICS. It orders the nodes on the basis of their *reachability scores*, which helps the user to understand the emerging network structure. After initial visualization, selected parameters are used for extracting communities, hubs, outliers from the network. It finds overlapping communities in a network with a worst-case running time of $O(n \log n)$.

A modified version of overlapping Girvan–Newman (GN) algorithm [12] was proposed to detect overlapping communities on the basis of a local form of betweenness. It discovers small-diameter communities in large networks and has a time complexity of $O(n \log n)$ for sparse networks. In one of his seminal papers, Mark Newman presented the notion of modularity (Q) [21] of a clustering or a cover in a network, using a concept of minimization of inter-cluster edges and maximization of intra-cluster edges. Modularity, equipped with mathematical versatility, was very popularly used by the researchers in measuring goodness of covers and related topics. It opened up a new problem, popularly known as the modularity maximization problem. The decision problem corresponding to this optimization problem was later proved to be NP-complete [3], and it is well accepted that heuristics can provide reasonably good solutions to the problem [10]. Newman himself presented a solution [20] to the problem, but it was computationally expensive and practically infeasible for massive graphs. Later, as an extension of that work, a disjoint community detection technique was proposed by Clauset, Newman and Moore, now popularly known as CNM [5]. It is essentially a greedy algorithm that uses efficient data structures to store and find the maximum gain in modularity incrementally and eventually finds communities in sub-quadratic running time. Another greedy modularity maximization algorithm by Blondel et. al. [2] used a simplistic approach of looking into the neighbors of a node to look for increase in modularity. After deciding on a tentative split, it shrinks the network, thereby drastically reducing future computation. This method is popularly known as the Louvain method. Local notion of modularity has also been used for detection of communities by modifying the equation of modularity and including a parameter to address the resolution limit [29].

Raghavan et. al. proposed a fast community detection algorithm [23] popularly known as the label propagation algorithm (LPA). It runs multiple breadth-first searches in successive iterations in a random manner such that labels propagate locally and after a few iterations converge to provide a stable final cover. This algorithm has a running time $O(m+n)$. The algorithm has several disadvantages—for example, it searches for the similar nodes locally by spreading labels to adjacent nodes and it needs multiple iterations of breadth-first traversals. The convergence of node labels can be mathematically guaranteed but it is not known if the number of iterations needed for the convergence of the node labels is dependent on n, m or some other network parameter. Another community detection algorithm that claims to work fast in practice is Infomap [24]. In this algorithm, the problem of community detection has been transformed into compression of information during its flow in the network. The algorithm uses random walks to move within the network and uses entropy-based information compression policies to find out the final cover. More recently, another linear time community detection technique (CGA) was presented by Wang et. al. [27] that detects communities in social networks taking into account information diffusion. It detects community structure in social networks using an approach of label propagation with a worst-case running time of $O(m)$. Although it has a provable bound, there was no empirical proof to justify that it runs faster than the Louvain method. Also, their results could not be reproduced due to unavailability of any public release from the authors of the papers. The theoretical background of our method is different from these algorithms or methods but it has a similar bound for its running time. Another community detection technique that uses depth-first traversal is LexDFS [6]. The worst-case time complexity of LexDFS algorithm has been reported to be $O(n \log n)$. We pit the performances of our method against the popular fast community detection techniques with publicly available releases. The Louvain method is widely accepted as the present state of the art in terms of finding disjoint communities from a network. Therefore, any improvement on the results obtained from the Louvain method can be considered as an improvement of the state of the art. If Louvain method is started from a bias (i.e., a cover is fed as an input) instead of starting from the original network, the method can be faster and the structure of the final cover will largely depend on the initial bias. Therefore, a cover generated from a given bias might be quite different from the final cover generated by the Louvain method when it is run on the original network. This motivates us to generate a bias such that we can generate final covers that are better in quality.

Recently, local searches have been used to look into the neighborhood of a vertex to find the best possible community for that vertex [7]. A new area of classifying nodes and thereby predicting communities is node and community embedding [16,26,32]. Usually, node embedding outputs a vector representation for each node in the graph, such that two nodes being "close" on the graph have similar vector representations. Another recent method has used Jaccard coefficient to find communities [18]. In this paper, Jaccard coefficient has been used as a measure to detect locally dense group of nodes as the metric finds similarity between two adjacent nodes by detecting common neighbors between them. Similar to GN method, it detects communities by successive removal of edges ordered on the basis of non-decreasing values of Jaccard coefficient.

3 Problem formulation

A vertex v is said to be influenced if a piece of information spreads to it from its neighbors (referred to as $\Gamma(v)$). Evidently, this is a temporal feature of the nodes and we assume that if a node u gets influenced at time $t = i$, it remains influenced thereon, and all its uninfluenced neighbors get influenced at time $t = i + 1$. This is similar to applying the breadth-first traversal algorithm in a graph.

Definition 1 (*Influence*) A node $v \in V$ is said to be *influenced*, once it has been discovered by using a graph traversal method.

Definition 2 (*Influenced Neighbors Score*) The *influenced neighbors score* of a vertex v at time $t = i$, $INS(v)_{t=i}$, is calculated as the fraction of the number of neighbors of vertex v that have been influenced up to the previous time-stamp, i.e., $t = i - 1$.

$$INS(v) = \frac{\textit{Number of influenced neighbors of v}}{\textit{Degree(v)}} \tag{1}$$

Clearly, $INS(v)$ lies between 0 and 1. For the starting node, it is zero. If all the neighbors of v have been influenced before v is processed, then $INS(v)$ will be 1. Initially, INS value for all the nodes is unassigned. During the influence propagation, each node is accessed and its INS value is calculated. Clearly, INS value of a node is calculated only once, i.e., when it is discovered for the first time from the neighborhood of the node that is being processed. Therefore, calculation of INS value of a node is dependent on the order of information spread (i.e., the traversal order), which, in turn, is dependent on the choice of the starting node.

If t is not explicitly mentioned for $INS(v)$, then it means INS value has to be calculated for the current time-stamp. We explain how to calculate INS value of a node with an example shown in Fig. 1. In this figure, we see that the INS value for node B is being calculated. At the time of calculating $INS(B)$, a few nodes have already been influenced. In Fig. 1, we can see that C, D, I, K L, M and N have been influenced. They have been shown in gray color in order to categorize them differently from the other nodes. If INS value of B is being calculated at $t = i$, then the nodes in gray have been influenced at any time-stamp between time-stamps $t = 1$ to $i - 1$. At the time of calculating INS(B), we see how many nodes are in the neighborhood of B and how many of them are already influenced. C is the only influenced neighbor of B out of its four neighbors {A, C, E, F}; therefore, $INS(B)$ is calculated to be $\frac{1}{4}$.

The intuition behind the definition of INS comes from the fact that if two neighboring nodes are in same community and they are highly likely to have lots of common neighbors. During the influence propagation, if one of them is reached first, the influence will reach the other node and the common neighbors in the next time-stamp. If the other node is processed in the next time-stamp, then it will find that many of its neighbors have already been influenced.

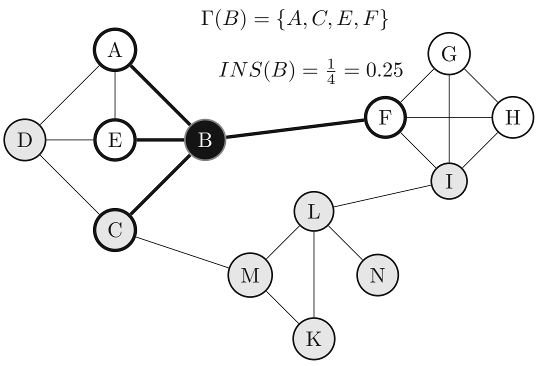

Fig. 1 Calculating INS value of a node. The figure shows how $INS(B)$ is calculated

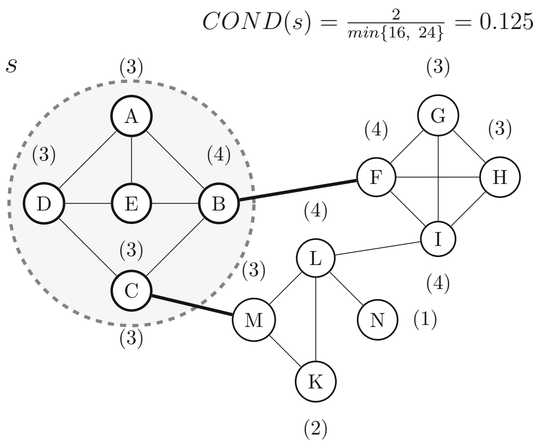

Fig. 2 Calculating conductance of a cut set

It gives us an idea that the node is in a closely knit community and the information it wanted to propagate is already with many of its neighbors. The idea is intuitive and was first mentioned by Granovetter [11]. Here, we present a mathematical form to use it in our proposed method.

Definition 3 (*Cut*) In a graph $G(V, E)$, *cut* of a cluster s is defined as the sum of the weight of edges from cluster s to its complement $V \setminus V_s$ [28].

Definition 4 (*Conductance*) Let V_s be the set of vertices in a cluster s. The *conductance* of s can be defined as the cut dividing the least number of edges incident on either the cluster or the rest of nodes in the network $(V \setminus V_s)$. In other words, it is the probability of leaving the cluster by a one-hop walk starting from the smaller set between V_s and $V \setminus V_s$ [28].

The formulation of *conductance* can be given by,

$$COND(s) = \frac{C(V_s, V \setminus V_s)}{min(C(V_s, V), C(V \setminus V_s, V))}, \quad (2)$$

where, $C(A, B)$ is defined as the sum of the weights of edges between subgraphs with node sets A and B. Note that $A \cap B$ may not always be a null set. Figure 2 shows an example how conductance of a cut set $s = \{A, B, C, D, E\}$ is computed for the given graph.

Definition 5 (*Broker nodes*) A vertex v is said to be a *broker node*, if at present time-stamp, $INS(v)$ is less than a predefined threshold r, where $0 \leq r \leq 1$.

The fraction r is termed as *community threshold* and may differ from one network to another for finding the best available community structure present in the network. Empirically, it was found (as shown in Table 8) that the variation of the quality of the communities obtained is marginal when r lies in the range of 0.6 to 0.8. In a sparse graph, the community threshold for a network may have a smaller value than that of in a dense graph. The nodes that are not broker nodes are declared as *community nodes*.

Definition 6 (*Community nodes*) A vertex v is said to be a *community node*, if at the present time-stamp or at $t = i$, $INS(v)$ is greater than or equal to a threshold r, where $0 \leq r \leq 1$.

When using conductance as the objective function, broker nodes are defined as those nodes that when added to a cluster, cause an increase or no change in the conductance value. Nodes that are not brokers are declared as community nodes. These definitions have been repeatedly used in the methods we have proposed in this paper.

Problem statement

Given a graph $G = (V, E)$, where $n = |V|$ and $m = |E|$, find a cover (set) of k partitions (communities) from the hypothesis space consisting of all possible covers, represented by $C = \{C_1, C_2, C_3, \ldots\}$, where C_i is the ith cover from the hypothesis space, $V_i = \cup_{j=1}^{k} V_{ij}$ and $E_i = \cup_{j=1}^{k} E_{ij}$, such that the value of the clustering goodness measure is maximized over all such possible covers where $1 \leq k \leq n$.

Note that the goodness measure can be performed by any goodness function such as modularity [21] or overlapping modularity [19,22,25]. In other words, for a graph G, if the value of the goodness function for cover C_i is given by $Q(G, C_i)$, then our aim is to find

$$\arg\max_{c \in C} Q(G, c) = \{Q(G, C_i) \mid \forall C_i, C_j \in C, Q(G, C_i) \geq Q(G, C_j)\}.$$

The target is to find the c, for which $Q(G, c)$ is maximum out of all possible covers from the hypothesis space C. Cover c is considered to be an overlapping partition if intersection of the vertex sets of any two clusters produces a set that is not null. Otherwise, the cover c is considered to be a disjoint partition. The problem presented here is essentially a goodness maximization problem. The decision version of goodness maximization problem, with modularity being the function for measuring goodness, has been proved to be NP-complete [3].

For clarification, it should be noted that in our proposed method we try to achieve final clusters with good cluster quality. It is independent of any particular clustering goodness measure. We try to achieve high goodness measure without using any particular goodness measure as the objective function. Instead, we use $INS(v)$ and $COND(s)$ as objective functions to maximize intra-cluster edges and minimize inter-cluster edges.

4 Traversal-based community detection

In this section, we propose a framework for two community detection methods which aim to find communities from a social network in linear running time (linear in terms of the size of the network) using traversal techniques. During the traversal, we first label nodes as brokers or community nodes based on an objective function. Next, we place the broker nodes in the community where most of its neighbors lie. We then reduce the graph on the basis of this initial split, where every cluster is merged into one super-vertex, with a weighted self-loop depicting the total number of intra-cluster edges. Finally, modularity maximization is run to generate the final cover. The method works as if a seed node is chosen to spread some information to the whole network. So the seed node spreads the information to all its neighbors and they in turn spread the information to their neighbors, who do not have the information yet. This process goes on iteratively, and in the process, we analyze the path of traversal to find out the communities. In this community detection algorithm, we use traversal techniques in sequential manner and thereby achieve a linear running time. We use the same framework to generate different clusters by devising two different methods on the basis of the objective functions mentioned in Sect. 3.

4.1 Part 1: Finding the broker nodes to outline the communities

The main idea of our method is to traverse the whole graph and partition it into a set of communities by observing how much of influence has reached a node's neighborhood or how connected its neighborhood is. As mentioned earlier in this paper, communities are subgraphs with dense intra-connections and sparse inter-connections. Broker nodes reside in the bordering areas of a community, i.e., the areas where communities overlap. As a result, a piece of information that is exclusive to one community can spread to an adjacent community only via the broker nodes. These broker nodes behave as transition points between two or more communities. When a piece of information reaches to a community, it first reaches the broker nodes and then spreads among its members. Then, through some other broker nodes, the information is passed on to another adjacent community. In our method, we follow the pattern of information flow via the broker nodes to explore and thereby detect the community structure of a graph in the process.

INS-based algorithm:

In this method, a node in a graph is identified as either a broker node or a community node on the basis of its INS value. When a node is encountered during the traversal, we find the number of its neighbors that are carrying the information during that time-stamp. If the fraction of neighbors of a node v carrying the information is less than r (say 0.75) then we assume that the information has not yet reached the community v belongs to, and v is one of the first nodes from its community to receive the information. Therefore, v is termed as a *broker* node and it is marked with a community label, which is same as its node label. Otherwise, the node is categorized as a *community* node.

Conductance-based algorithm:

In this method, a node is identified as a broker when addition of the node in the presently growing cluster s increases $COND(s)$. Initially, when the process starts from a single node, the cluster consists of a single node and $COND(s)$ is 1. As new nodes are picked up during the traversal, we try to include the newly reached node in the cluster and check how the conductance changes. If it increases then it is considered to be a broker node, else the process to grow the cluster continues. Intuitively, it should stop at the borders of the prospective clusters. Nodes, other than the broker nodes, are marked as community nodes.

The community nodes identified during the traversal from one broker node to the next identified broker node are said to belong to the same community as the brokers mark the border of two communities. The community label for a group of community nodes found subsequently during traversal is same as the label of the broker node that led to the traversal of those community nodes.

4.2 Part 2: Allocating broker nodes to the communities

In this part, we put the broker nodes in the communities they are most likely to belong. This decision is made based on a metric called belonging probability that calculates the ratio between the number of edges that exists from a broker node to all the nodes in a community out of the maximum number of edges possible between the broker node and that community. The idea behind defining such a metric is to find out the community to which a broker node is connected to with most number of edges. The absolute value of the number of connections to a community may not correctly represent the association of a broker node to that community. Increase in size of communities leads to increase in the possible number of connections with the broker node. Hence, if the number of actual connections is normalized by the possible number of connections that can be made with a community, the probability of a broker node belonging to that community remains independent of the size of the community. Eventually, broker nodes are placed in the community that leads to the highest belonging probability.

Some of the broker nodes may still fall into communities they are not supposed to be a part of as identified by the ground truth. The sequential nature of the traversal-based detection of communities may lead to such a situation. We solve this problem by identifying the community in which most of the neighbors of a broker node lie. In case of a tie (i.e., if the cardinality of the set of neighbors in a particular community maximizes for more than one community), we do not assign a community label to the broker node but leave it to the modularity maximization step to merge it with one of the clusters on the basis of the topological structure. A similar policy is maintained for the broker nodes with no community node in its neighbor. In the first part of modularity maximization, we reduce the present clusters into super-vertices and thereby transform the network into an undirected network of super-vertices. Every super-vertex consists of a self-loop, which has a weight equivalent to twice the number of intra-cluster edges in that cluster. The edges between a pair of super-vertices have weights equivalent to the number of inter-cluster edges between those two clusters. This step is termed as the reduction of the network. After the reduction, modularity changes for potential merges are calculated and increase in modularity is greedily maximized. In this step, the super-vertices are merged iteratively to produce the final cover. Due to the nature of the algorithms, the initial clusters are expected to be fragmented, however, with high precision. Therefore, a modularity maximization process may merge those high precision community fragments to achieve communities with both high precision and high recall. Such communities are likely to be more similar to the ground truth than the initial fragments.

4.3 Mathematical definition of community nodes

The following definition of conductance is equivalent to the one defined in Eq. (2). In this section, we also abbreviate *COND* to *C* for convenience.

The definition of conductance is given by,

$$C(S) = \frac{\sum_{i \in S, j \notin S} A_{ij}}{min\{k_S,\ k_{\bar{S}}\}}$$

where the numerator is the cut size, i.e., the number of edges from S to $\bar{S}$, k_S is the sum of degree of nodes in cluster S.

We are interested in the change in conductance of cluster S triggered by the addition of a single neighboring node (called the target node) to the cluster. If the conductance decreases after the addition, the target node is classified as a **community** node and added to the community S; otherwise, it is classified as a **broker** node.

Given a graph and a partial cluster S, we can classify the nodes in the graph into three types,

1. The nodes currently in the cluster S.
2. The target node t, which could be potentially added to S,
3. The other nodes o which belongs to O (i.e., $V - S - \{t\}$).

Say, k_t is the degree of the target node, k_S and k_O are the degrees of the clusters S and O, respectively, $k_{t,S}$ is the number of the edges incident from the target t to the cluster S, α is the number of edges incident from the cluster S to the set O. In other words, α represents the number of edges in the cut set **not** incident on the target node t.

We investigate the numerator of the definition, i.e., the cut size. Notice that the cut size initially (before t is added to S) is $\alpha + k_{t,S}$. After t is added to S, the cut size becomes $\alpha + (k_t - k_{t,S})$. This is because when t becomes a part of S, it brings k_t many edges with itself. Out of which only $k_t - k_{t,S}$ contribute to the cut size, since $k_{t,S}$ edges become part of the cluster S after the merge.

For the denominator, initially it is $min\{k_S, (k_t + k_O)\}$. After t's addition, it becomes $min\{(k_S + k_t), k_O\}$. Since we pick the minimum of the two quantities, three cases arise:

1. $k_S < k_t + k_O$ and $k_S + k_t < k_O$
2. $k_S < k_t + k_O$ and $k_S + k_t \geq k_O$
3. $k_S \geq k_t + k_O$ (this guarantees that $k_S + k_t > k_O$, so there is no 4th case).

The initial conductance before t is added to S is represented as C_{old} and after t's addition it is C_{new}. For each case, we find the condition on $k_{t,S}$.

Case I: $k_S < k_t + k_O$ and $k_S + k_t < k_O$

$$\begin{aligned}
C_{\text{old}} &= \frac{\alpha + k_{t,S}}{k_S}, C_{\text{new}} = \frac{\alpha + k_t - k_{t,S}}{k_S + k_t} \\
C_{\text{old}} - C_{\text{new}} &= \frac{\alpha + k_{t,S}}{k_S} - \frac{\alpha + k_t - k_{t,S}}{k_S + k_t} \\
k_S(k_S + k_t)(C_{\text{old}} - C_{\text{new}}) &= (\alpha + k_{t,S})(k_S + k_t) - k_S(\alpha + k_t - k_{t,S}) \\
&= \alpha k_S + \alpha k_t + k_S k_{t,S} + k_t k_{t,S} - \alpha k_S - k_S k_t + k_S k_{t,S} \\
&= k_{t,S}(2k_S + k_t) + k_t(\alpha - k_S)
\end{aligned}$$

For t to be a community node, $C_{\text{old}} - C_{\text{new}} > 0$. Note that $k_S(k_S + k_t)$ is always positive. Therefore,

$$\begin{aligned}
k_{t,S}(2k_S + k_t) + k_t(\alpha - k_S) &> 0 \\
k_{t,S}(2k_S + k_t) &> k_t(k_S - \alpha) \\
k_{t,S} &> \frac{k_t(k_S - \alpha)}{2k_S + k_t} \qquad (3)
\end{aligned}$$

Case II: $k_S < k_t + k_O$ and $k_S + k_t \geq k_O$

$$\begin{aligned}
C_{\text{old}} &= \frac{\alpha + k_{t,S}}{k_S}, C_{\text{new}} = \frac{\alpha + k_t - k_{t,S}}{k_O} \\
C_{\text{old}} - C_{\text{new}} &= \frac{\alpha + k_{t,S}}{k_S} - \frac{\alpha + k_t - k_{t,S}}{k_O} \\
k_S k_O (C_{\text{old}} - C_{\text{new}}) &= (\alpha + k_{t,S})k_O - k_S(\alpha + k_t - k_{t,S}) \\
&= \alpha k_O + k_O k_{t,S} - \alpha k_S - k_S k_t + k_S k_{t,S} \\
&= k_{t,S}(k_O + k_S) + \alpha(k_O - k_S) - k_S k_t
\end{aligned}$$

For t to be a community node, $C_{\text{old}} - C_{\text{new}} > 0$. Note that $k_S k_O$ is always positive. Therefore,

$$\begin{aligned}
k_{t,S}(k_O + k_S) + \alpha(k_O - k_S) - k_S k_t &> 0 \\
k_{t,S}(k_O + k_S) &> k_S k_t + \alpha(k_S - k_O) \\
k_{t,S} &> \frac{k_S k_t + \alpha(k_S - k_O)}{(k_S + k_O)} \qquad (4)
\end{aligned}$$

Case III: $k_S \geq k_t + k_O$

$$\begin{aligned}
C_{\text{old}} &= \frac{\alpha + k_{t,S}}{k_t + k_O}, C_{\text{new}} = \frac{\alpha + k_t - k_{t,S}}{k_O} \\
C_{\text{old}} - C_{\text{new}} &= \frac{\alpha + k_{t,S}}{k_t + k_O} - \frac{\alpha + k_t - k_{t,S}}{k_O} \\
k_O(k_O + k_t)(C_{\text{old}} - C_{\text{new}}) &= (\alpha + k_{t,S})k_O - (k_t + k_O)(\alpha + k_t - k_{t,S}) \\
&= \alpha k_O + k_O k_{t,S} - \alpha k_t - k_t^2 + k_t k_{t,S} - \alpha k_O - k_O k_t + k_O k_{t,S} \\
&= k_{t,S}(2k_O + k_t) - k_t(\alpha + k_t + k_O)
\end{aligned}$$

For t to be a community node, $C_{\text{old}} - C_{\text{new}} > 0$. Note that $k_O(k_O + k_t)$ is always positive. Therefore,

$$\begin{aligned}
k_{t,S}(2k_O + k_t) - k_t(\alpha + k_t + k_O) &> 0 \\
k_{t,S}(2k_O + k_t) &> k_t(\alpha + k_t + k_O) \\
k_{t,S} &> \frac{k_t(\alpha + k_t + k_O)}{2k_O + k_t} \qquad (5)
\end{aligned}$$

Equations 3, 4 and 5 give the conditions for classifying the target node to be a community node. Combining them together, we can say,

$$k_{t,S} > \begin{cases} \frac{k_t(k_S - \alpha)}{2k_S + k_t} & \text{if } k_S < k_t + k_O \text{ and } k_S + k_t < k_O \\ \frac{k_S k_t + \alpha(k_S - k_O)}{(k_S + k_O)} & \text{if } k_S < k_t + k_O \text{ and } k_S + k_t \geq k_O \\ \frac{k_t(\alpha + k_t + k_O)}{2k_O + k_t} & \text{if } k_S > k_t + k_O \end{cases} \qquad (6)$$

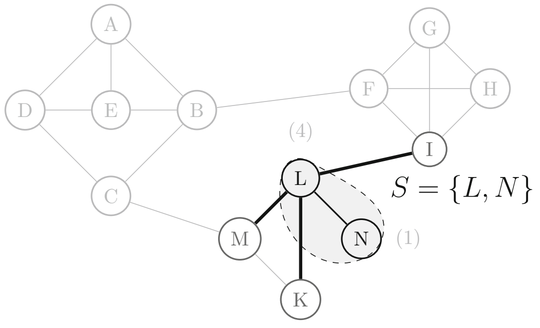

Fig. 3 The graph with the cluster $S = \{L, N\}$. The cut edges are drawn with thick lines. The number in parentheses near the nodes represent the degree

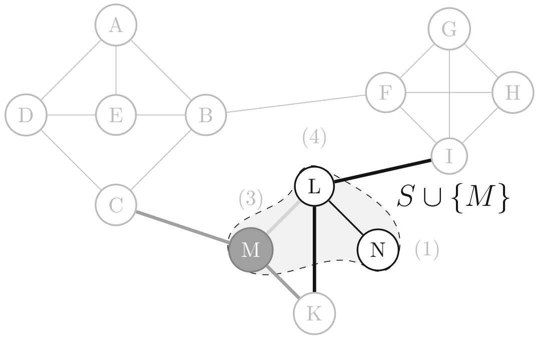

Fig. 4 *M* is the target node. The number in parentheses above the nodes represent the degree of the nodes. The edges contributing to $k_{t,S}$, $(k_t - k_{t,S})$ and α are shown in light gray, gray and black, respectively

4.4 Illustrating examples: classifying community and broker nodes

4.4.1 Example 1

Figures 3 and 4 illustrate the first example. From Fig. 3, we can calculate the parameters needed to classify the target node M as a community node or a broker node. The parameters, in this example, are as follows:

$$S = \{L, N\},\ k_S = 5,\ t = M,\ k_t = 3,\ k_{t,S} = 1,\ \alpha = 2,\ k_O = 32.$$

We have $k_S < k_t + k_O$ and $k_S + k_t < k_O$. Thus, we check the condition for case I to check if M is a community node.

$$k_{t,S} > \frac{k_t(k_S - \alpha)}{2k_S + k_t}$$

Here, LHS $= k_{t,S} = 1$. Plugging in the values in the RHS, we obtain,

$$\begin{aligned} \text{RHS} &= \frac{k_t(k_S - \alpha)}{2k_S + k_t} \\ &= \frac{3 \times (5 - 2)}{2 \times 5 + 3} \\ \text{RHS} &= \frac{9}{13} \end{aligned}$$

Thus, we have LHS > RHS, so, M is a **community** node, and is therefore added to S (as shown in Fig. 4).

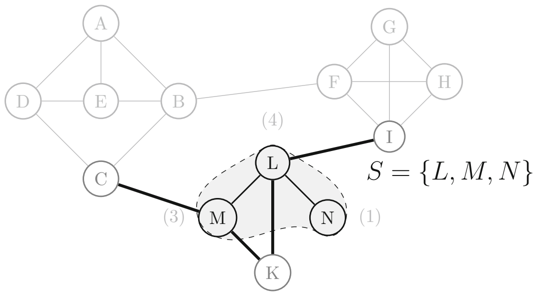

Fig. 5 The graph with the cluster $S = \{L, M, N\}$. The cut edges are drawn with thick lines. The number in parentheses near the nodes represent the degree

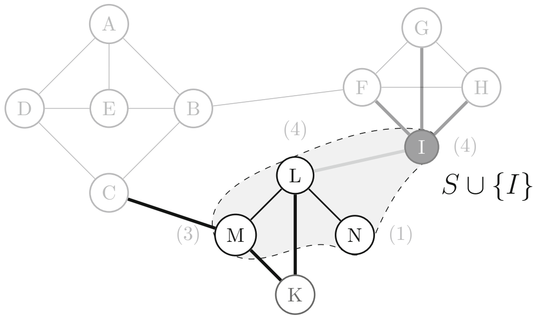

Fig. 6 I is the target node. The number in parentheses above the nodes represent the degree of the nodes. The edges contributing to $k_{t,S}$, $(k_t - k_{t,S})$ and α are shown in light gray, gray and black, respectively

4.4.2 Example 2

Figures 5 and 6 illustrate the second example. From Fig. 3, we can calculate the parameters needed to classify the target node I. The parameters, in this example, are as follows:

$$S = \{L, M, N\},\ k_S = 8,\ t = I,\ k_t = 4,\ k_{t,S} = 1,\ \alpha = 3,\ k_O = 28.$$

We have $k_S < k_t + k_O$ and $k_S + k_t < k_O$. Thus, we check the condition for case I to check if I is a community node.

$$k_{t,S} > \frac{k_t(k_S - \alpha)}{2k_S + k_t}$$

Here, LHS $= k_{t,S} = 1$. Plugging in the values in the RHS, we obtain,

$$\begin{aligned} \text{RHS} &= \frac{k_t(k_S - \alpha)}{2k_S + k_t} \\ &= \frac{4 \times (8 - 3)}{2 \times 8 + 4} \\ \text{RHS} &= \frac{4 \times 5}{16 + 4} = 1 \end{aligned}$$

Thus, we have LHS equal to RHS. From Eq. 6, we can say, I is **not** a community node, but a **broker** and is therefore not added to S.

4.5 Proposed algorithms

We have described our proposed method in Algorithms 1, 2, 3, 4, 5, 6, 7, 8 and 9, incrementally. Algorithm 1 describes the central LINCOM method algorithm replaced by method, where a broker stack S and a community queue Q have been used as primary data structures to facilitate the traversal. We call other procedures from Algorithm 1 to update S and Q and detect communities in the process. The phenomenon termed as "spread of information" has been described by Algorithms 3 and 2. The process of spread imitates breadth-first traversal. As a part of the data structures used, we have also used three node attributes to store flags for all the nodes, regarding whether it is a broker node or a community node (using nodeType), whether it has been covered during the traversal or not (using covered) and which community it presently belongs to (using community).

Algorithm 1: Traversal-based Linear Time Community Detection (LINCOM(G, r))

```
Input  : Undirected, unwtd. graph $G(V, E)$, threshold $(r)$
Output: Cover of $k$ communities, $G_s = \{G_{s_1}, G_{s_2}, \ldots, G_{s_k}\}$
1  begin
2      find $v_{start} \in V, \ni v_{start}$ is the node with lowest degree
3      forall the $v \in V$ do
4          $v$.nodeType $\leftarrow$ $v$.covered $\leftarrow 0$
5          $v$.community $\leftarrow v$
6      $S \leftarrow Q \leftarrow \emptyset$                     ▷brokerStack($S$), communityQueue($Q$)
7      coverCount $\leftarrow 1$
8      $v \leftarrow v_{start}$
9      NODE-CAT($G, v, Q, S$)                  ▷NODE-CAT-INS or NODE-CAT-COND
10     while $coverCount < n$ do
11         if $Q$ is non-empty then
12             $v \leftarrow$ dequeue($Q$)
13         else
14             $v \leftarrow$ pop($S$)
15         NODE-CAT($G, v, Q, S$)              ▷INS or COND-based NODE-CAT
16     forall the $v \in V$ do
17         check $v$.community to form $G_s$
18     POST-PROCESS($G_s$)
19     REDUCE($G_s$)
20     MOD-MAXIMIZE($G_s$)
21     return $G_s$
```

Algorithm 2 describes a procedure NODE-CAT-INS that is called by LINCOM (when using INS), until all the nodes in the graph have been categorized. In Algorithm 1, we call a generic version of the procedure referred to as NODE-CAT. But while presenting the pseudocodes, we have named the INS-based and COND-based versions of NODE-CAT as NODE-CAT-INS (presented in Algorithm 2) and NODE-CAT-COND (presented in Algorithm 9), respectively.

The purpose of NODE-CAT-INS is to traverse the untraversed nodes, mark them as traversed, categorize them as broker nodes or community nodes and finally, place the broker nodes in brokerStack and community nodes in community queue. Categorization and placement of nodes are performed in a few steps. In line 2 of the procedure NODE-CAT-INS, we use a procedure called SPREAD, which spreads the influence from a node v to all its uncovered neighbors. NODE-CAT very often makes use of a procedure that calculates the INS values

Algorithm 2: Categorizing uncovered nodes in $\Gamma(v)$ (NODE-CAT-INS(G, v, Q, S))

```
Input  : Undirected, unwtd. graph $G(V, E)$, threshold ($r$),
         brokerStack ($S$), communityQueue ($Q$)
Output: Update $Q$, $S$
1  begin
2      SPREAD($v$)
3      for $u \in \Gamma(v)$ do
4          if $u.nodeType \neq 0$ then continue
5          if INS($u$) $< r$ then
6              $u$.nodeType $\leftarrow 1$                    ▷marking the broker nodes
7              push($S$, $u$)
8          else
9              $u$.nodeType $\leftarrow 2$                    ▷marking the community nodes
10             enqueue($Q$, $u$)
11             $u$.community $\leftarrow$ $v$.community
```

Algorithm 3: Spreading Influence to the Neighbors (SPREAD(v))

```
Input  : Undirected, unwtd. graph $G(V, E)$, root node ($v$)
Output: Update coverCount, covered list
1  begin
2      for $u \in \Gamma(v)$ do
3          if $u.covered = 0$ then
4              $u$.covered $\leftarrow 1$
5              coverCount $\leftarrow$ coverCount + 1
```

Algorithm 4: Influenced Neighbors Score (INS(v))

```
Input  : Undirected, unwtd. graph $G(V, E)$, root node ($v$)
Output: INS($v$)
1  begin
2      coveredCount $\leftarrow 0$              ▷keeps a count of the covered neighbors of $v$
3      for $u \in \Gamma(v)$ do
4          if $u.covered = 1$ then
5              coveredCount $\leftarrow$ coveredCount + 1
6      INS($v$) $\leftarrow$ coveredCount / deg($v$)
7      return INS($v$)
```

Algorithm 5: Conductance ($COND(s)$)

```
Input  : Undirected, unwtd. graph $G(V, E)$, set of nodes $V_s$ in cluster $s$
Output: Conductance score of the set of nodes $V_s$ in cluster $s$
1  begin
2      cutSize $\leftarrow 0$
3      forall the $u \in V_s$ do
4          for $v \in \Gamma(u)$ do
5              if $v \in V \setminus V_s$ then
6                  cutSize $\leftarrow$ cutSize +1
7      return $cutSize / min\{\sum_{w \in V_s} deg(w), \sum_{w \in V \setminus V_s} deg(w)\}$
```

Algorithm 6: Post-processing of the Broker Nodes (POST-PROCESS(G_s))

```
Input  : Undirected, unwtd. graph $G(V, E)$, root node $(v)$
Output : Update $v$.community value for broker nodes
1  begin
2      forall the $v \in V$ do
3          $max(v) \leftarrow 0$
4      forall the $v \in V$ do
5          if $v.nodeType = 1$ then
6              forall the $G_{s_i} \in G_s$ do
7                  if $\frac{|Neighbors\ of\ v\ in\ G_{s_i}|}{|G_{s_i}|} > max(v)$ then
8                      $max(v) \leftarrow \frac{|Neighbors\ of\ v\ in\ G_{s_i}|}{|G_{s_i}|}$
9                      $v$.community ← community label that leads to max($v$)
10                 else
11                     if $\frac{|Neighbors\ of\ v\ in\ G_{s_i}|}{|G_{s_i}|} = max(v)$ then
12                         $v$.community ← append community label to community list
```

Algorithm 7: Reduction of the Network (REDUCE(G_s))

```
Input  : $G_s$
Output : Updated $G_s$ $(G'_s)$
1  begin
2      forall the $G_{s_i} \in G_s$ do
3          $v'_i$ represents all the nodes in $V_{s_i}$
4          $w(v'_i, v'_i) \leftarrow \Sigma_{u,v \in V_{s_i}} w(u, v)$
5          $w(v'_i, v'_j) \leftarrow \Sigma_{u \in V_{s_i}, v \in V_{s_j}} w(u, v) | u \neq v, i \neq j$
6      return $G'_s$
```

Algorithm 8: Modularity Maximization (MOD-MAXIMIZE(G_s))

```
Input  : $G_s$
Output : Updated $G_s$ $(G'_s)$
1  begin
2      repeat
3          forall the $v \in G_s$ do
4              forall the $u \in \Gamma(v)$ do
5                  find $\Delta Q$ if $v$ moved to $u$.community
6              if $\Delta Q_{\max} > 0$ then
7                  move $v$ to $u_{\max}$.community with $\Delta Q_{\max}$
8      until there is no $\Delta Q_{\max} > 0$
9      return $G'_s$
```

Algorithm 9: Categorizing uncovered nodes in $\Gamma(v)$ (NODE-CAT-COND(G, v, Q, S))

```
Input  : Undirected, unwtd. graph $G(V, E)$, brokerStack ($S$), communityQueue ($Q$)
Output : Update $Q, S$
1  begin
2      for $u \in \Gamma(v)$ do
3          if $u.nodeType \neq 0$ then continue
4          if $u$ satisfies condition from Eq. 6 then
5              $u.nodeType \leftarrow 2$                    ▷marking the community nodes
6              $enqueue(Q, u)$
7              $u.community \leftarrow v.community$
8          else
9              $u.nodeType \leftarrow 1$                    ▷marking the broker nodes
10             $push(S, u)$
```

of the neighbors of a node v, as described in Algorithm 4. Similarly, NODE-CAT-COND, is also used for categorization of nodes. Note that we do not use SPREAD in the COND-based method. In NODE-CAT-COND procedure, COND(u) represents the subroutine to find out the value of conductance, as described in Algorithm 5. Please note that for a network with multiple connected components, just like any other traversal method, LINCOM can also be applied sequentially to all the components.

Algorithm 6 describes the procedure POST-PROCESS, which places the broker nodes in the clusters of the present cover G_s. Community label of a broker node v is assigned to the community that contributes to the largest fraction of community nodes in its neighborhood. If there is a tie or all the neighbors of a broker node consist only of brokers then the community label of the broker remains unassigned. Such brokers are assigned a community label during one of the latter procedure calls, referred to as MOD-MAXIMIZE. MOD-MAXIMIZE merges such brokers with the existing clusters such that the merged clusters would generate a cover with improved modularity. In MOD-MAXIMIZE, first we reduce the network by converting each of the initial clusters into a super-vertex. This is done by the procedure REDUCE. After reduction of the network, MOD-MAXIMIZE is applied in a way similar to the Louvain method [2].

Complexity Analysis: Our algorithm uses breadth-first and depth-first traversal techniques, both known to have worst-case time complexities of O($|V| + |E|$), where $|E|$ denotes the number of edges in the graph. The categorization of nodes takes O($|E|$) time as it involves traversal through all the edges. The time taken to place the broker depends upon the number of brokers obtained and their respective degrees. Number of brokers can never exceed the number of nodes and as the sum of degrees of all nodes is bounded by $2|E|$, the time complexity for this step is O($|E|$).

Lastly, the reduction procedure merges all the nodes in a community into one super-vertex and adds up all the inter-cluster and intra-cluster edges separately. Given, that community membership of a node can be accessed in constant running time, it has to traverse all the edges once. Therefore, the network reduction runs in O($|E|$). In modularity maximization part, we already work on a reduced graph and so its running time will not exceed the order of the reduced graph, because every node will need to traverse its neighbors trying to find the maximum modularity gain. Therefore, the overall worst-case running time of our algorithm to create the initial bias for the community detection will be O($|E|$). The time complexity of the modularity maximization part is unknown [2] but as it works on a reduced graph it works very fast in practice. We proceed to verify the performance of our method by showing some of the experimental results that we performed on some benchmark datasets.

5 An example to illustrate the split produced by INS-based method

We illustrate our proposed method with a graph consisting of 13 nodes and 20 edges. The nodes are labeled by uppercase letters, as shown in Fig. 7a. For this example, we take the threshold value of r to be 0.66. As in OPTICS, our algorithm also starts from an outlier-like point, i.e., a pendant node, which does not play a significant role in a community. The starting node will always be considered as a broker node with an INS value of 0. Initially, from the only pendant node N, influence is propagated to L, as seen in Fig. 7a, b. INS(L) evaluates to 0.25 because only one node (N) out of all the four of its neighbors (I, K, M, N) has become influenced up to this stage. According to our algorithm, L is identified as a broker and pushed into the broker stack. Community queue is still empty at this stage. So we pop the top element from the broker stack, i.e., L, and process it. By processing a node v we mean, propagating the influence to all the neighbors of v, then calculating INS value for all its neighbors, thereby categorizing them as broker nodes and community nodes and finally storing them in broker stack and community queue, respectively. For L, we spread the influence to I, K and M. INS values of I, K and M turn out to be 0.25, 1.0 and 0.67, respectively. Since INS(I) being less than the value of r (0.66), I is placed in the broker stack and K and M are placed in community queue (we store them in lexicographic order, but it is not necessary). It should be noted that all the broker nodes discovered till this point have been assigned a community label that is same as their node label, as shown in Table 1.

But when a community node is discovered, the node is given a community label same as the label of the last processed broker node, which is essentially the broker node that leads to its discovery. For example, here community(K) = community(M) = L. Now, after insertion of K and M, community queue is non-empty and hence the elements in the queue will be dequeued and processed until the queue is empty. The stack is not processed until the queue is empty. Therefore, K is processed first without spreading further influence to any other

Table 1 Sequence of discovering the nodes in the example

Node	INS	Category	Label	Final label
N	0	Broker	N	L
L	0.25	Broker	L	L
I	0.25	Broker	I	F
K	1.00	Community node	L	L
M	0.67	Community node	L	L
C	0.33	Broker	C	D
D	0.33	Broker	D	D
B	0.25	Broker	B	D
A	1.00	Community node	D	D
E	1.00	Community node	D	D
F	0.50	Broker	F	F
G	1.00	Community node	F	F
H	1.00	Community node	F	F

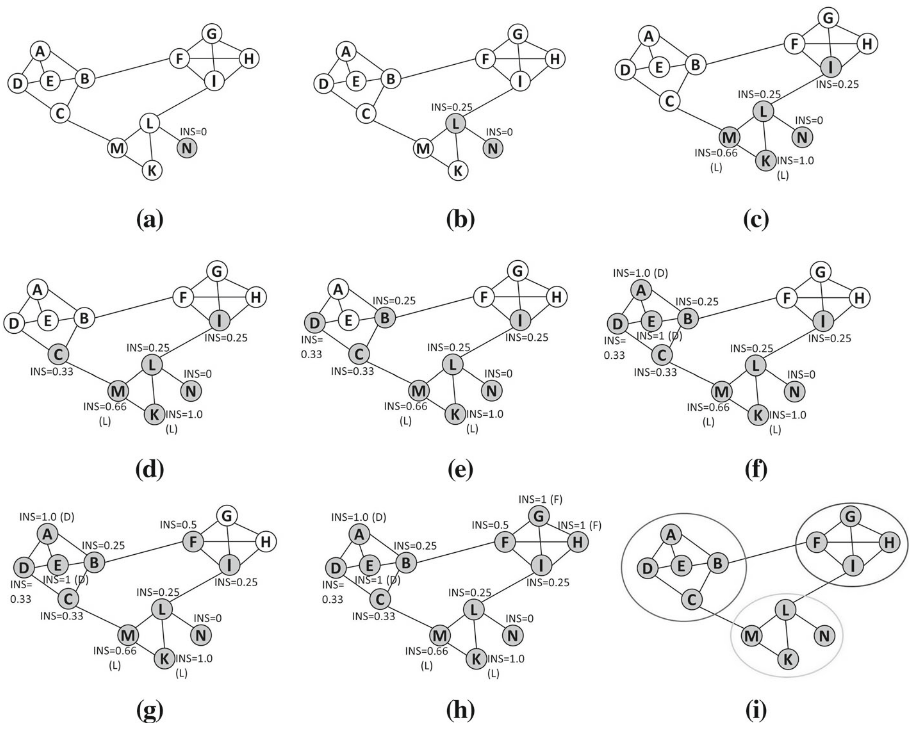

Fig. 7 Steps to detect communities during the traversal-based algorithm LINCOM. **a** Broker stack (BS): *N*, Community queue (CQ): *empty*. Starts from lowest degree node *N*. **b** BS: *L*, CQ: *empty*. Processing *N*. Neighbor of *N* is *L* (INS = 0.25), goes to broker stack. **c** BS: *I*, CQ: *K M*, processing *L*. Neighbor of *M* with INS ≥ 0.66, goes to community queue. **d** BS: *I C*, CQ: *empty*. *e* BS: *I B D*, CQ: *empty*, Processing *C*. **f** BS: *I B*, CQ: *A E*, Processing *D*. **g** BS: *I F*, CQ: *empty*, Processing *B*. **h** BS: *I*, CQ: *G H*, Processing *F*. **i** Three communities were found after modularity maximization

node in the graph. Then M reaches out to a single uncovered node C. INS(C) is 0.33 and it is subsequently placed in the stack. At this stage, the stack consists of I and C. Figure 7c, d shows these steps. In Fig. 7e, we see the broker node at the top of the stack is getting popped and thereby getting processed. C spreads influence to all its uncovered neighbors (B, D). Due to low INS values, B and D are both identified as broker nodes and get pushed into stack. Now, after popping D and processing it, we reach nodes A and E. Both of them have the same INS value 1 and are identified as community nodes. The nodes in the graph's adjacency list are stored and processed in lexicographic order prompting B to be pushed to the stack before D. Therefore, D first comes to the top while popping and is processed before B. That is why A and E will also have a community label of D. This is shown in Fig. 7f. Next node to be processed is B. It spreads to a new node F. As INS(F) = 0.5, F is stored at the top of the stack. Community queue being empty, F is processed in the next step. F spreads to the remaining two uncovered nodes (G and H) of the network. For both of them, INS value evaluates to 1.0 and hence they are categorized as community nodes with community label F, as shown in Fig. 7g, h. After post-processing, the communities turn out to be as shown in Fig. 7i. Please note that this is an intermediate cover and not the final cover. The final cover is obtained by applying the procedure MOD-MAXIMIZE on the intermediate cover.

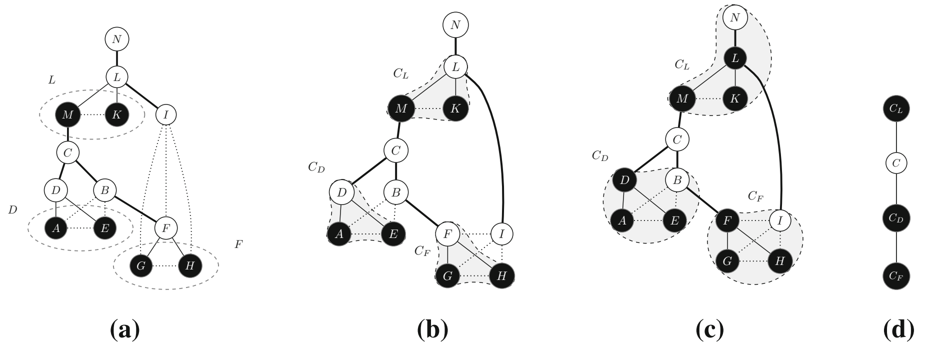

Fig. 8 Transforming the network into a depth-first tree of clusters. Breadth-first traversal is used when traversing within the communities. The white nodes are the broker nodes and the black nodes are the community nodes. The thick lines mark the discovery of a broker node. **a** Categorization of nodes into broker and community nodes. **b** The broker nodes D, L and F lead to finding community nodes (thereby clusters) from their neighbors. During the post-processing, broker nodes like D, L and F become part CD, CL and CF, respectively. **c** Rest of the broker nodes are placed in the community they have highest edges with. **d** Final depth-first tree with clusters as nodes

It is important to understand that the initial clustering generated by LINCOM is done by running two graph traversal methods, i.e., depth-first traversal and breadth-first traversal, in parallel. We start from a node assuming it is getting the spread started. Since it is getting the spread started, irrespective of the node actually being a broker node or a node inside a community, it will be considered as a broker node (once the traversals are over, during the post-processing, it will be placed into the correct community eventually). The traversal methods and the corresponding trees are shown in Fig. 8. In Fig. 8a, we see how the nodes have been categorized into two different types - namely the broker nodes (white) and the community nodes (black). Note that the community nodes store the label of the broker node from which it was discovered. In Fig. 8b, such broker nodes, that are used to label a community, is placed inside that community. In Fig. 8c, the other broker nodes are placed inside the community, which has most number of edges connected to it. If there is a tie, then the broker node is considered as a singleton cluster and is passed on to modularity maximization process as a part of the initial cover. In Fig. 8d, we observe that node C has equal number edges to both C_D and C_L. Modularity maximization agglomerates such a node to a cluster such that the overall modularity of the final cover is maximized.

From the illustrations, it can be observed that initially a lot of broker nodes are obtained. This is because, at the beginning of the spread propagation inside a community, most of the nodes within that community are unvisited. Therefore, in this method, some small-sized (possibly singleton) communities may be generated at the initial stage. But as the brokers are placed and the community labels of the nodes are stabilized, the communities grow larger and we get the final labels for all the nodes (shown as the final labels in Table 1). Similarly, the initial split can also be generated by using conductance. After applying modularity maximization on the obtained cover, the average size of the clusters grows further.

5.1 Effect of different starting points on the clusters

Due to the nature of the algorithm, we can pick any node to be the starting point for the traversal. Depending on the starting point, the order of traversal may change and the broker node that leads to the discovery of a community may change but discovery of the communities

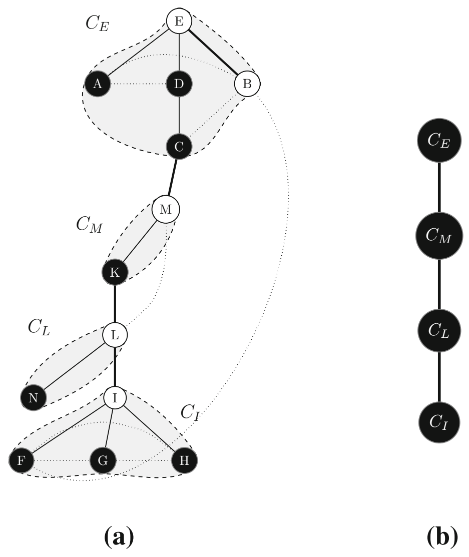

Fig. 9 Discovery of clusters with E as the starting node. **a** Traversal tree. **b** Final tree of clusters

remain more or less similar. We have already provided an example in Fig. 7. If other starting points are used on the same network, same cover is produced every time. Figure 9a shows the traversal tree for starting node E, with back edges and cross edges in addition to the tree edges. In Fig. 9b, only the tree edges have been shown along with the clusters after the first phase of the algorithm. In this figure, node B is actually connected to cluster C_E with a back edge. After modularity maximization, node B moves into C_E, C_M and C_L get merged and C_I remains as it is. This leads to the same final cover as in Fig. 7. In experimental results part, we have provided sufficient empirical evidence that our method is independent of starting point.

6 Experimental results

We have performed our experiments, including running the public releases of the Louvain method and CNM, on an Intel Xeon 2.4 GHz quad-core CPU desktop with 32GB RAM, 500 GB hard disk and Fedora LINUX version 3.3.4 OS. The source code has been written in C and it is publicly available.[1]

We evaluate the performance of our algorithm on different well-known benchmark datasets [9,13,15,17,30,31] by assessing the accuracy of its covers obtained, using a Newman modularity for the disjoint case and Nicosia modularity for the overlapping case. We further test our algorithm on large datasets to test its efficiency. We evaluate the effect of parameters used in our algorithm, i.e., threshold values for INS and choice of the starting nodes. The threshold value r for the INS value has a great impact on the size of the communities obtained. If r is assigned a low value, many nodes move to the same community during traversal, leading to mostly large-sized communities and only a few smaller ones. On the other hand, taking high value for r leads to fragmented and small communities along with many broker nodes, specially in the initial part of the traversal. A large number of broker

[1] The code can be downloaded from https://github.com/sna-lincom/LINCOM.

Table 2 Comparison of running time for different types of community detection algorithms for different benchmark datasets

Social network dataset	Nodes (n)	Edges (m)	LINCOM with INS time (s)	LINCOM with COND time (s)	Louvain time (s)	CNM time (s)
Karate	34	78	0	0	0	0
Dolphin	62	159	0	0	0	0
Lesmis	77	254	0	0	0	0
Football	115	613	0	0	0	0
GrQc	4158	13,422	0	0	0	4
Enron	33,696	180,811	0.21	0.22	0.38	362
Epinions	75,877	405,739	0.71	0.76	0.97	1953
Amazon	334,863	925,872	4.58	4	6	3578
DBLP	317,080	1,049,866	5.141	4	11	10,440
Orkut	3,072,441	117,185,083	246	377	456	–

nodes may converge to a single community in the latter part of our method, thereby forming large-sized communities. We have observed that setting $r = 0.75$ as threshold value gives us appropriately sized (with significant relevance to ground truth) communities and therefore in other experiments, INS-based LINCOM was run with $r = 0.75$.

6.1 Testing efficiency of LINCOM

We run INS-based LINCOM and COND-based LINCOM along with the implementation of the Louvain method and CNM, released by their respective authors for a comparative analysis, using the same set-up declared above. The time taken to run those algorithms is summarized in Table 2. Results show that both our methods run faster than the Louvain method. Clearly, CNM proves to be much slower, which is in coherence to the provable bounds for the running time of the algorithm. CNM cannot even find communities for massive datasets such as Orkut even in several hours. So, from the results, it can be empirically established that the concept of traversing the graph to detect communities turned out to be faster than the present state of the art. The variation of the objective function did not seem to generate much perturbation in terms of running times. Hence, other objective functions suitable to maximize intra-cluster edges and minimize inter-cluster edges can also be tested with similar efficiency unless the objective function itself is computationally expensive.

We have used CODACOM platform [6] to run all the state-of-the-art community detection algorithms and have tested their performance on the real-world benchmark datasets and the synthetic graph data originally used by Newman and Girvan [9]. In Table 3, we have compared LINCOM with other methods in terms of efficiency using real-world benchmark datasets. In Table 6, we have shown the results of running LINCOM and a few other state-of-the-art community detection algorithms on synthetic datasets. We created these synthetic datasets using the LFR benchmark generator [14]. These graphs have 128 nodes with each node having degree of 16. All the nodes in the network are divided into 4 communities with each community consisting of 32 nodes. In these graphs, mixing parameter (μ) defines the fraction of edges with nodes outside its own community. We have changed the value of μ from 0.1 to 0.3 to test the robustness of our algorithm as the communities become less recognizable with increase in μ.

Table 3 Comparing running times (in seconds) of the state-of-the-art community detection algorithms with LINCOM for different benchmark datasets

Methods	Amazon	DBLP	Dolphins	Enron	Epinion	Football	GrQc	Karate	LesMis
Louvain	**3.46**	4.36	0.0	0.28	**0.73**	0.0	0.02	0.0	0.0
LexDFS	13.02	14.18	0.0	3.40	11.24	0.0	0.10	0.0	0.0
Label Prop	27.72	43.87	0.0	0.69	0.93	0.0	0.03	0.0	0.0
Infomap	432.71	507.85	0.0	22.08	110.30	0.0	0.54	0.0	0.0
LINCOM	4.00	**4.00**	0.0	**0.22**	0.76	0.0	**0.0**	0.0	0.0

Bold numbers highlight the best time obtained for a given dataset

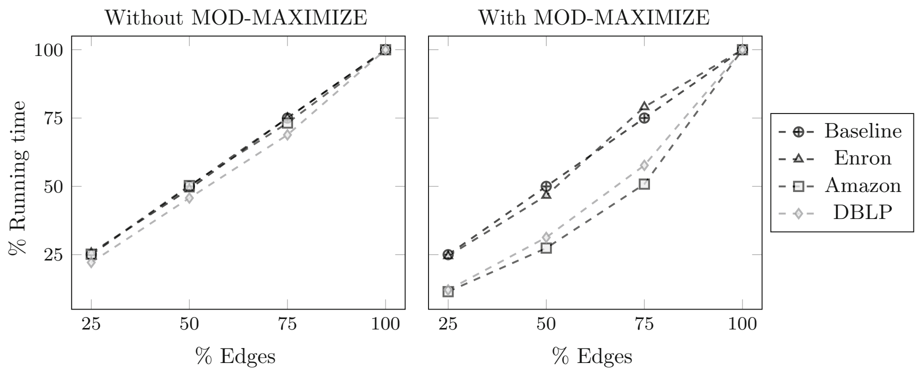

Fig. 10 Growth of running time for LINCOM when tested on 25, 50 and 75% of the edges of Enron, Amazon and DBLP datasets

In order to reinforce our claim that the first phase of LINCOM (i.e., LINCOM without modularity maximization) runs in linear time, we have tested LINCOM on sample subgraphs of large datasets by randomly sampling 25, 50 and 75% of the total edges of the networks. When we use LINCOM with modularity maximization, it shows nonlinear growth in running time. If we run LINCOM without the modularity maximization part, results show that the running times scale up linearly. The results are shown in Fig. 10. This can be considered as an empirical confirmation of the fact that the running time of LINCOM is indeed bounded by a linear function of the number of edges in the network.

6.2 Testing quality of the clusters

Here, we have used overlapping modularity [19,22,25] for evaluating overlapping communities, whereas for disjoint communities, we use modularity defined by Newman [21]. The variants of LINCOM are naturally overlapping, however, in order to compare them with disjoint communities on the basis of one goodness measure, we convert the overlapping communities into disjoint communities. We place each overlapping node in one of its neighboring communities such that the modularity is maximized. A summary of the results is presented in Table 4. Overlapping modularity of the covers generated by the variants of LINCOM is consistently greater than the modularity values of the covers produced by the Louvain method and CNM. The comparisons between the modularity values of the disjoint covers generated by the variants of LINCOM and the modularity maximization methods are close. Variants of LINCOM always seem to produce better clusters than that of CNM, particularly in large

Table 4 Comparison of the modularity values of the final covers generated by the Louvain method, CNM and the variants of LINCOM, for different benchmark datasets

Social network dataset	Overlapping LINCOM with INS	Overlapping LINCOM with COND	Disjoint LINCOM with INS	Disjoint LINCOM with COND	Louvain	CNM
Karate	0.729	0.445	0.402	0.3793	0.415	0.38
Dolphin	0.75	0.756	0.518	0.526799	0.518	0.492
LesMis	0.579	0.579	0.544	0.448	0.55	0.5
Football	0.673	0.694	0.582	0.543	0.604	0.57
GrQc	0.879	0.87	0.847	0.841375	0.847	0.79
Enron	0.73	0.7	0.587	0.598	0.596	0.49
Epinions	0.529	0.51	0.44	0.44895	0.45	0.385
Amazon	0.931	0.963	0.961	0.92054	0.926	0.87
DBLP	0.831	0.819	0.818	0.809	0.819	0.73
Orkut	0.59	0.801375	0.548	0.679	0.679	–

Table 5 Modularity values of the final cover of the state-of-the-art community detection algorithms for different benchmark datasets

Method	Amazon	DBLP	GrQc	Enron	Epinion	Football	Dolphins	Karate	LesMis
Louvain	0.926	**0.821**	0.847	**0.613**	**0.452**	**0.6**	0.518	0.392	**0.554**
LexDFS	0.536	0.414	0.552	0.171	0.094	0.576	0.338	0.23	0.434
Label Prop	0.785	0.7	0.769	0.317	0.044	0.584	0.410	0.352	0.523
Infomap	0.825	0.722	0.771	0.511	0.347	0.6	0.517	0.402	0.546
LINCOM	**0.961**	0.818	**0.847**	0.596	0.44	0.582	**0.518**	**0.402**	0.544

Bold numbers highlight the highest values of modularity obtained for a given dataset

Table 6 Performance of the state-of-the-art community detection algorithms on LFR generated synthetic benchmark datasets

Method	$\mu = 0.1$	$\mu = 0.2$	$\mu = 0.3$
Ground truth	0.648	0.548	0.452
Louvain	0.648	0.548	0.452
LexDFS	0.648	0.548	0.328
Label Prop	0.648	0.548	0.230
Infomap	0.648	0.548	0.452
LINCOM	0.648	0.545	0.452
CNM	0.648	0.548	0.452

networks. The Louvain method and variants of LINCOM often produce covers with the same modularity value with a very low tolerance level.

In Table 5, we have compared performance of LINCOM with the performance of some of the state-of-the-art community detection algorithms. LINCOM consistently performs better than the others except the Louvain method. Louvain method outputs covers with similar modularity values in some cases. In Table 6, we have evaluated LINCOM's performance on synthetic networks with 4 existing communities where the community structure becomes more obscure as the mixing factor increases. The ground truth is available and the modularity

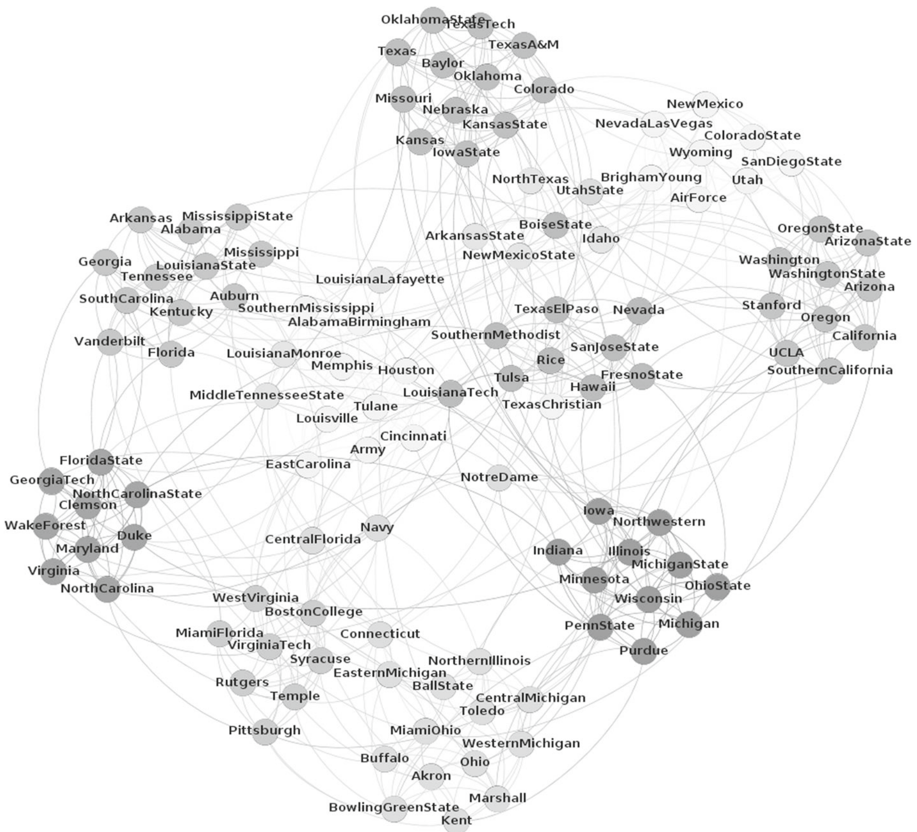

Fig. 11 Ground truth for the colleges in the twelve conferences [9]

of the ground truth cover decreases as the mixing factor increases. From the results, LINCOM seems to be more robust than some of the other methods as it can recognize the ground truth structure more readily than the others.

In every community detection algorithm, it is important to understand the practical significance of the method and what it can be used for. To understand that we compare the final cover obtained by INS-based LINCOM method (Fig. 12b) with the ground truth of the US College football network as described by Newman [9] (Fig. 11) and the final cover obtained by Louvain method (Fig. 12a). This is one of the very few datasets where ground truth from real world is available and therefore can be used for benchmarking communities. We observe that Louvain generates a cover with ten clusters whereas LINCOM ends up generating a cover with eight clusters. Interestingly, the significance of the ground truth communities is maintained both in Louvain as well as LINCOM. In the final cover generated by Louvain method, we can observe that there are two cases where a couple of conferences have merged. Similarly, a further degree of coarsening has taken place in LINCOM, where two pairs of communities have merged to reduce the number of clusters further. Understandably, due to LINCOM's use of modularity maximization technique, inability to detect small clusters has become an inherent problem. But the clustering found in LINCOM is defined by dense

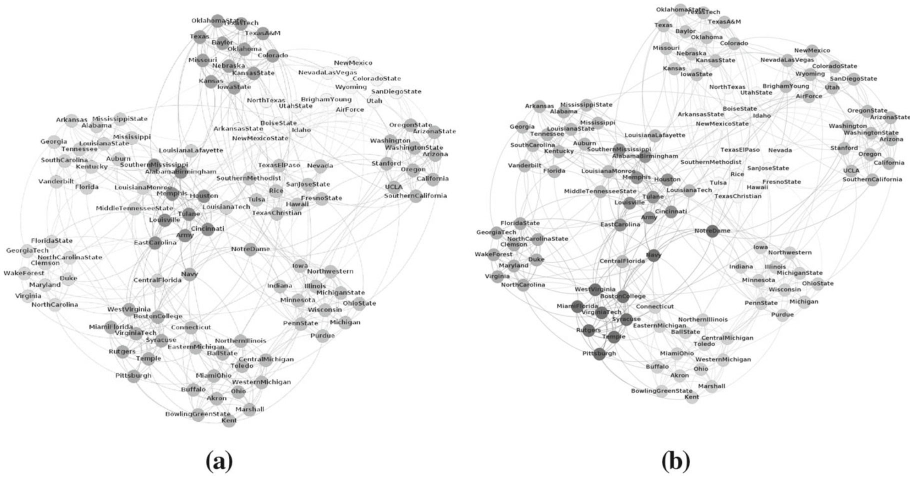

Fig. 12 Visualization of the final covers as obtained from Louvain method and LINCOM. **a** Final cover obtained from Louvain method. **b** Final cover obtained from LINCOM

connection, a higher modularity value and evidently has a high precision when compared to the ground truth.

6.3 Selection of starting node in LINCOM

Starting points may differ in traversal methods and we observed that LINCOM generated different covers from different starting nodes. We used INS-based LINCOM to generate covers exhaustively by using every node of the network as starting points. For large networks, this process proved to be time-consuming. In order to solve that problem, for large networks, we chose a random sample of starting nodes to figure out a similar statistics. We plotted the centrality measures of the starting node, such as degree centrality, PageRank, betweenness centrality and closeness centrality against the modularity of the final cover obtained. We observed that the modularity does not seem correlated with the centrality measures mentioned. From the plots in Fig. 13, we can clearly state that a good starting node cannot be selected by the usual network features of a node such as centrality measures. On the other hand, the experiment uncovered an interesting fact that whatever the starting node might be, the modularity does not change significantly. From experiments, we found out (see Table 7) that we can randomly pick any node in the network as a starting node to find a cover and still guarantee that the modularity of the cover will be within a very small relative standard deviation (usually 0–3%) of the mean modularity value of all the covers generated by using every node of the network as starting point.

6.4 Selection of INS threshold (r)

We have performed experiments to run INS-based LINCOM over several INS threshold values, thereby creating a variety of final covers (as shown in Table 8). We compared the quality of those final covers for each dataset and found that most of the benchmark datasets produce the best cover when INS threshold (r) is 0.75. Therefore, in other experiments, we have chosen to keep the value of r as 0.75.

Table 7 Comparison of the reported modularity values of the final covers generated by LINCOM, and the mean modularity value of all the covers generated by using every node in the network as starting point. Results have been reported for different benchmark datasets

Network dataset	Mean modularity	SD	Reported modularity
Karate	0.402	0.015	0.402
Dolphin	0.518	0.014	0.518
Lesmis	0.544	0.011	0.544
Football	0.581	0.017	0.582
GrQc	0.847	0.001	0.847
Enron	0.578	0.013	0.587
Facebook	0.818	0.023	0.835

Table 8 Comparison of the modularity values of the final covers generated by the INS-based LINCOM, by varying the value of INS threshold (r), for different benchmark datasets

Threshold INS (r)	Karate	Dolphin	Football	LesMis	Facebook	GrQc	Enron	Amazon	DBLP
0.4	0	0.475	0	0.331	0.651	0.765	0.574	0.959	0.70
0.45	0	0.483	0	0.408	0.700	0.822	0.574	**0.961**	0.74
0.5	0	0.480	0	0.282	0.776	0.832	0.582	**0.961**	0.78
0.55	0.372	**0.526**	0.428	0.504	0.826	0.840	0.598	**0.961**	0.80
0.6	0.372	0.518	0.435	0.529	0.821	0.843	**0.604**	**0.961**	0.81
0.65	0.372	**0.526**	0.541	0.529	0.832	0.842	0.581	**0.961**	**0.82**
0.7	0.372	**0.526**	0.541	0.529	**0.835**	0.843	0.602	**0.961**	**0.82**
0.75	0.372	**0.526**	**0.6**	0.521	**0.835**	**0.847**	0.575	**0.961**	**0.82**
0.8	**0.42**	0.525	0.586	0.529	**0.835**	0.846	0.6	0.960	**0.82**
0.85	0.39	0.524	0.574	**0.560**	0.834	0.846	**0.604**	0.959	**0.82**

Bold numbers highlight the highest values of modularity obtained for a given dataset

6.5 Size of communities produced by LINCOM

We observe the number of final clusters in each of these methods (as shown in Table 9). In some cases, LINCOM produces covers of similar size, when compared to the Louvain method. In most other cases, the size of the covers generated by our methods are smaller, i.e., the communities are larger compared to the Louvain method. Therefore, LINCOM guarantees that the communities produced are not too small to be considered insignificant. Smaller cover size ensures average size of clusters is larger. A cover with larger-sized clusters produces higher modularity only if the community structures are more meaningful than a cover with smaller clusters.

7 Conclusion

We have convincingly shown that efficient detection of *high quality* communities can be achieved in near linear running time (in terms of the size of the graph) by combining simple traversal methods such as breadth-first and depth-first traversals. Our methods run faster than

Table 9 Comparison of cover size for different types of community detection algorithms for different benchmark datasets

Social network dataset	Nodes (n)	Edges (m)	LINCOM with INS $\|G_s\|$	LINCOM with COND $\|G_s\|$	Louvain $\|G_s\|$	CNM $\|G_s\|$
Karate	34	78	2	3	4	3
Dolphin	62	159	3	2	5	4
Lesmis	77	254	2	2	6	5
Football	115	613	6	5	9	7
GrQc	4158	13,422	28	13	42	61
Enron	33,696	180,811	61	23	170	567
Epinions	75,877	405,739	671	541	733	2983
Amazon	334,863	925,872	121	59	246	1409
DBLP	317,080	1,049,866	101	228	244	3113
Orkut	3,072,441	117,185,083	136	6	11	−−

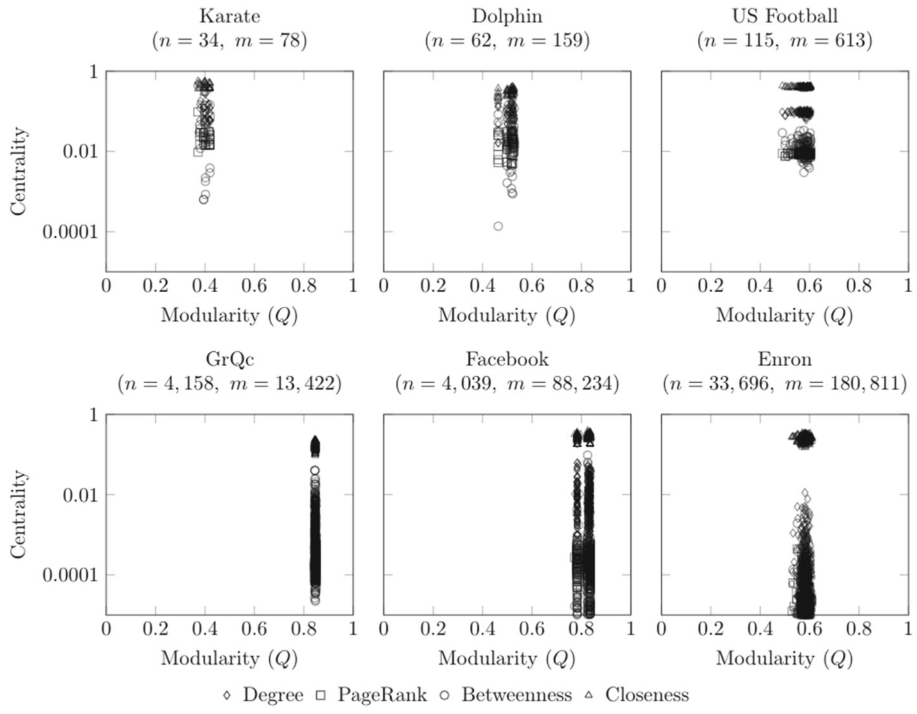

Fig. 13 For the networks in the first row, every node in the network was sequentially tested as the starting node. For the networks in the bottom row, 500 nodes were randomly sampled and then their centrality data is plotted. The modularity of the final cover found using that starting node was plotted against its degree centrality, PageRank, betweenness centrality and closeness centrality

the state-of-the-art community detection techniques. Also, the quality of the communities generated by the proposed methods are at least as good as the state-of-the-art community detection techniques.

Acknowledgements We would like to thank the reviewers for their effort for thoroughly reading our paper and for suggesting valuable changes. We would also like to thank Upasana Dutta for pointing out and correcting errors in some of the figures and algorithms that appear in this work.

References

1. Ankerst M, Breunig MM, Kriegel HP, Sander J (1999) Optics: ordering points to identify the clustering structure. ACM SIGMOD Rec 28(2):49–60. ISSN 0163-5808. https://doi.org/10.1145/304181.304187. http://portal.acm.org/citation.cfm?id=304187
2. Blondel VD, Guillaume JL, Lambiotte R, Mech ELJS (2008) Fast unfolding of communities in large networks. J Stat Mech https://doi.org/10.1088/1742-5468/2008/10/P10008
3. Brandes U, Delling D, Gaertler M, Goerke R, Hoefer M, Nikoloski Z, Wagner D (2006) Maximizing modularity is hard. http://arxiv.org/abs/physics/0608255
4. Chen J, Zaiane OR, Goebel R (2009) A visual data mining approach to find overlapping communities in networks. In: Memon N, Alhajj R (eds) ASONAM. IEEE Computer Society, pp 338–343. ISBN 978-0-7695-3689-7. http://dblp.uni-trier.de/db/conf/asunam/asunam2009.html#ChenZG09a
5. Clauset A, Newman MEJ, Moore C (2004) Finding community structure in very large networks. Phys Rev E 70:066111
6. Creusefond J, Largillier T, Peyronnet S (2017) A lexdfs-based approach on finding compact communities. In: Kaya M, Erdoğan Ö, Rokne J (eds) From social data mining and analysis to prediction and community detection. Springer, Berlin, pp 141–177
7. Cui W, Xiao Y, Wang H, Wang W (2014) Local search of communities in large graphs. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. ACM, pp 991–1002
8. Fortunato S, Hric D (2016) Community detection in networks: a user guide. Phys Rep 659:1–44
9. Girvan M, Newman MEJ (2002) Community structure in social and biological networks. Proc Natl Acad Sci 99(12):7821–7826
10. Good BH, De Montjoye YA, Clauset A (2010) Performance of modularity maximization in practical contexts. Phys Rev E 81(4):046106
11. Granovetter MS (1973) The strength of weak ties. Am J Sociol 78(78):1360–1380
12. Gregory S (2008) A fast algorithm to find overlapping communities in networks. In: Daelemans W, Goethals B, Morik K (eds) ECML/PKDD (1), volume 5211 of lecture notes in computer science. Springer, Berlin, pp 408–423. ISBN 978-3-540-87478-2
13. Klimt B, Yang Y (2004) Introducing the enron corpus. In: First conference on email and anti-spam (CEAS) proceedings
14. Lancichinetti A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. Phys Rev E 78(4):046110
15. Leskovec J, Kleinberg JM, Faloutsos C (2007) Graph evolution: densification and shrinking diameters. TKDD. https://doi.org/10.1145/1217299.1217301
16. Lin C, Ishwar P, Ding W (2017) Node embedding for network community discovery. In: 2017 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp 4129–4133. https://doi.org/10.1109/ICASSP.2017.7952933
17. Lusseau D, Newman MEJ (2004) Identifying the role that animals play in their social networks. Proc R Soc Lond Ser B Biol Sci 271:S477–S481
18. Meghanathan N (2016) A greedy algorithm for neighborhood overlap-based community detection. Algorithms 9(1):8
19. Nepusz T, Petroczi A, Negyessy L, Bazso F (2007) Fuzzy communities and the concept of bridgeness in complex networks. Phys Rev E 77:016107
20. Newman MEJ (2004) Fast algorithm for detecting community structure in networks. Phys Rev E 69:066133. https://doi.org/10.1103/PhysRevE.69.066133
21. Newman MEJ (2006) Modularity and community structure in networks. Proc Natl Acad Sci 103(23):8577–8582
22. Nicosia V, Mangioni G, Carchiolo V, Malgeri M (2009) Extending the definition of modularity to directed graphs with overlapping communities. J Stat Mech Theory Exp 2009(03):P03024

23. Raghavan UN, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76:036106. https://doi.org/10.1103/PhysRevE.76.036106
24. Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci USA 1118
25. Shen HW, Cheng XQ, Guo JF (2009) Quantifying and identifying the overlapping community structure in networks. J Stat Mech Theory Exp 2009(07):P07042. http://stacks.iop.org/1742-5468/2009/i=07/a=P07042
26. Wang X, Cui P, Wang J, Pei J, Zhu W, Yang S (2017) Community preserving network embedding. In: AAAI, pp 203–209
27. Wang Y, Cong G, Song G, Xie K (2010) Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining, KDD'10. ACM, New York, NY, USA, pp 1039–1048. ISBN: 978-1-4503-0055-1. https://doi.org/10.1145/1835804.1835935
28. Whang JJ, Gleich DF, Dhillon IS (2013) Overlapping community detection using seed set expansion. In: Proceedings of the 22nd ACM international conference on information & knowledge management, CIKM'13. ACM, New York, NY, USA, pp 2099–2108. ISBN: 978-1-4503-2263-8. https://doi.org/10.1145/2505515.2505535
29. Xiang J, Tao H, Zhang Y, Ke H, Li J-M, Xiao-Ke X, Liu C-C, Chen S (2016) Local modularity for community detection in complex networks. Phys A Stat Mech Its Appl 443:451–459
30. Yang J, Leskovec J (2012) Defining and evaluating network communities based on ground-truth. In: Zaki MJ, Siebes A, Yu JX, Goethals B, Webb GI, Wu X (eds) ICDM. IEEE Computer Society, pp 745–754. ISBN: 978-1-4673-4649-8. http://dblp.uni-trier.de/db/conf/icdm/icdm2012.html#YangL12
31. Zachary W (1977) An information flow model for conflict and fission in small groups. J Anthropol Res 33:452–473
32. Zheng VW, Cavallari S, Cai H, Chang KC-C, Cambria E (2016) From node embedding to community embedding. arXiv preprint arXiv:1610.09950

Partha Basuchowdhuri is an assistant professor in the Department of Computer Science and Engineering, Heritage Institute of Technology. He received his Doctoral, Master's and Bachelor's degrees from Jadavpur University, India, Louisiana State University, USA and Bengal Engineering College (presently IIEST, Shibpur), India, respectively. He also worked as a postdoctoral research fellow in Queen's University Belfast, UK and as a research programmer at DERI (presently Insight Centre at NUI Galway). His areas of research interest include network science, graph theory and data mining.

Satyaki Sikdar is a Ph.D. student at the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, Indiana, USA. He received his Bachelor's degree from Heritage Institute of Technology, Kolkata, India in 2017. His research interests include network science and graph mining.

Varsha Nagarajan currently works as a software engineer at Micro Focus (previously Novell), where she actively contributes to the development and enhancement of ZENworks, an endpoint management product. She received her Bachelor's degree in Computer Science from Heritage Institute of Technology, Kolkata, India in 2015. Her research interests include machine learning, data mining and optimization algorithms.

Khusbu Mishra is a full stack engineer at Aplopio Technology Pvt. Ltd., Bangalore, India, and has previously worked as a full stack developer at Paypermint Pvt. Ltd., Kolkata, India, and as a systems engineer at Infosys Ltd, Bangalore, India. She has an experience of over three years in Web Application development. She received her Bachelor's degree from Heritage Institute of Technology, Kolkata, India, in 2014. Her research interests include but are not limited to graph theory and data mining.

Surabhi Gupta is an end-to-end product engineer(full stack developer) at Paypermint Pvt. Ltd., India. She is a working actively in shaping up a digital payments platform which enables to move money online. She received her B.Tech. degree in Computer Science from Heritage Institute of Technology, India, in 2015. Her interests lie in developing scalable and user-friendly applications.

Subhashis Majumder received his M.Tech degree in Computer Science from the ISI, Calcutta, and got his Ph.D. degree in CSE from Jadavpur University, Kolkata. He also worked as a research assistant in the Computer Engineering Dept. of Rutgers University, for a year. He is presently serving as a full-time professor and the Head of the Computer Science and Engineering Department at Heritage Institute of Technology, Kolkata, where he is also the dean of undergraduate studies. He has over 50 publications in international conferences and journals and has jointly filed four Indian patents. His areas of interest in research include graph algorithms, physical design algorithms, algorithmic microfluidics, algorithms for social networking and empirical software engineering.