The Journal of Supercomputing (2019) 75:4894–4917
https://doi.org/10.1007/s11227-019-02765-1

Decentralized iterative approaches for community clustering in the networks

Amhmed Bhih[1] · **Princy Johnson**[1] · **Martin Randles**[1]

Published online: 9 February 2019

Abstract

In this era of big data, as the data size is scaling up, the need for computing power is exponentially increasing. However, most of the community detection algorithms in the literature are classified as global algorithms, which require access to the entire information of the network. These algorithms designed to work on a single machine cannot be directly parallelized. Hence, it is impossible for such algorithms working in stand-alone machines to find communities in large-scale networks and also the required processing power far exceeds the processing capabilities of single machines. In this paper, a set of novel Decentralized Iterative Community Clustering Approaches to extract an efficient community structure for large networks are proposed and devalued using the LFR benchmark model. The approaches have the ability to identify the community clusters from the entire network without global knowledge of the network topology and will work with a range of computer architecture platforms (e.g., cluster of PCs, multi-core distributed memory servers, GPUs). Detecting and characterizing such community structures is one of the fundamental topics in network systems' analysis, and it has many important applications in different branches of science including computer science, physics, mathematics and biology ranging from visualization, exploratory and data mining to building prediction models.

Keywords Community detection · Connectivity-based graph clustering · Distributed algorithm

✉ Amhmed Bhih
a.a.bhih@2011.ljmu.ac.uk; Amhmed_bhih@hotmail.com

Princy Johnson
P.Johnson@ljmu.ac.uk

Martin Randles
M.J.Randles@ljmu.ac.uk

[1] Department of Electronics and Electrical Engineering/Computer Science, LJMU, Liverpool L3 3AF, UK

1 Introduction

Many real-world complex systems can be represented as networks (also referred to as graphs), with nodes representing functional units and links describing the interactions between nodes.

Recently, it has become common to analyse interactions in the real-world by looking at the networks that underlie these interactions [1]. Real-world networks are not random networks, they usually exhibit inhomogeneity and reveal a high level of order and organization [2]. An interesting feature that real-world networks usually present is the community structure property, under which the topology of network is organized into modules commonly called communities or clusters [3]. Detecting and characterizing such community structures is one of the fundamental topics in network systems' analysis. The determination of communities in networks can help people better understand the structural makeup of the networks. Thus, the outcome of this research work has valuable applications in several fields such as biology, social science, physics, computer science, business science, etc. [4, 5].

In social networks, for example, clustering of communities can be beneficial for a range of applications including finding a common research area in collaboration networks and finding a set of likeminded users for marketing and recommendations [6]. Community structure is important not only in social networks, but also in various other networks. For example, determination of community structure in the Internet can address questions such as how to route data as packets in an efficient way, how to reduce the time consumption for such traffic, what is the fast and safe path to consider to reach the destination, etc. It can go further in depth, by elucidating questions like how computer viruses are spreading through the Internet, what mechanisms they follow to hit organizations, etc. Also in dark networks, community structure can reveal the hidden relationships between individual terrorists [7]. Similarly, in the case of the World Wide Web (WWW) pages related to the same subject are typically organized into communities, so that the identification of these communities can help the task of seeking for identifying the category of the network as well as understanding its dynamic evolution and organization [8]. Thus, the problem of finding the community structure of networks has attracted a huge amount of research work and the range of proposed algorithms is rich and diverse. However, most of the research on community detection algorithms has been designed to work on a single machine employing a form of basic random access to the entire network, so they require access to the entire network at all times [3, 9].

Driven by the recent emergence of big data, clustering of real-world networks using traditional methods and algorithms is almost impossible to be processed in a single machine. The existing methods are limited by their computational requirements, and most of them cannot be directly parallelized. Furthermore, in many cases the data set is very big and does not fit into the main memory of a single machine and therefore needs to be distributed among several machines [10].

Faced with the challenge of a big data set, many researchers pay great attention to parallel clustering algorithms that would improve the bottleneck of traditional clustering methods on a single machine. To cope with this scenario, a distributed and parallel computing model is needed to process a large data set by scaling

the data set out to multiple machines across a cluster and process it. Some novel parallel computing frameworks shine, of which MapReduce is one of the most popular [11]. However, the traditional clustering algorithms are centralized (need global information) and do not have the capability to process data across multiple servers in parallel (or distributed manner).

The main goal of this work is to design and implement novel techniques and algorithms for the problem of clustering and community detection in large and undirected networks. The proposed approaches all assume that the given network structure is needed to be divided into communities in such a way that every node belongs to one of the communities (non-overlapping communities).

The following summary provides a short overview of the key contributions of this work:

1. A novel Decentralized Iterative Community Clustering Approach (DICCA) to extract an efficient community structure for large networks is proposed. A major advantage of this approach is eliminating the need for the global knowledge of the network in order to efficiently cluster networks. This allows the DICCA to be run in parallel and the network data need not be loaded into a single memory. Hence, the proposed approach is adapted to cluster communities in large networks without the penalties involved. This cannot be done in the majority of the existing community detection algorithms as they implicitly assume that the entire structure of the big network is known and is available. Another perspective of DICCA approach is reducing the problem size by aggregating the nodes in the network to cluster the large-scale data set efficiently.
2. A Parallel Decentralized Iterative Community Clustering Approach (PDICCA) that transforms the operations of the DICCA approach from a serial process into a parallelized approach is presented. The PDICCA is a pipelined parallel implementation and maintains the overall structure of the serial method (DICCA). The novelty of the design comes from the following fact: even though the PDICCA solves the same problem and maintains the overall structure as does the serial method, the PDICCA is distinguished due to the features of exploiting the use of distributed memory and extracting parallelism under the MapReduce framework. The proposed algorithm does not require any global knowledge of the network topology, is scalable and will work with a range of computer architecture platforms (e.g., cluster of PCs, multicore distributed memory servers, GPUs), where the master and slave workers could represent either different threads in a single machine or different machines in a computing cluster. Also, one of the main contributions of this work is to take advantage of the graph partitioning when performing parallel community clustering in order to speed up the process by minimizing the communication between slave–workers. Furthermore, a parallel implementation of PDICCA based on the most popular MapReduce model to accelerate processing in large-scale networks is proposed.

The rest of this paper is organized as follows: Sect. 2 presents a brief overview of the related literature on graph partitioning and community detection algorithms. Sections 3 gives a detailed description of Decentralized Iterative Community Clustering

Approach, for detecting communities. Section 4 centres around the design and implementation of the parallel framework version of the DICCA approach. In this section, the principle and implementation of the proposed PDICCA approach is detailed. The mathematical model to obtain optimal parameter values for the proposed approaches is presented in Sect. 5. The data benchmarks and experimental results are presented in Sects. 6 and 7 respectively. Finally, discussion and future work are presented in Sect. 8.

2 Related work

2.1 Graph partitioning and community detection

Community detection is an active area of network science research, and over the years, a wide variety of community detection algorithms have been proposed to find the communities in the network. Community detection is also named as graph partitioning, in much of the literature [12, 13]. It is tempting to suggest that this community detection and graph partitioning are really addressing the same question; in both, their aim is to identify groups of nodes on a network that are better connected to each other than to the rest of the network. However, it is very important to stress that the task of graph partitioning and community detection can be distinguished from one another based on whether the experimenter fixes the number and size of the groups or it is unspecified [14]. Graph partitioning is the problem of partitioning a graph into a predefined number and size of clusters. It has been pursued particularly in computer science and related fields with applications in parallel computing and very-large-scale integration (VLSI) design. However, in the community detection, which has been pursued by sociologists and more recently by physicists and applied mathematicians, with applications especially to social and biological networks, the number and size of clusters are unspecified. Furthermore, the goal in the former is usually to identify the best division of a network regardless of whether or not a good division existed. In case there are no good divisions exist, the least bad one will be done as a solution. On the other hand, in community detection, the algorithm only divides the network when good divisions exist and leave the network undivided in case there are no existing good divisions [14].

The community detection algorithms can be classified into different ways, and depending on the selected criteria, one algorithm can belong to more than one category. Among them, those based on modularity maximization form the most prominent family of community detection algorithms such as Fastgreedy algorithm [15] and Louvain algorithm [16].

Fastgreedy algorithm is an agglomerative hierarchical clustering method proposed by Newman [15]. The algorithm greedily maximizes the modularity function Q and starts the process by assigning a different community to each node in the network. Then at each stage in the process, the pair of clusters that yields greatest increase of modularity or smallest decrease is merged until only one cluster remains containing all nodes in the network. The whole procedure can be represented by a dendrogram (hierarchical tree) that illustrates the order of the mergers. Cuts through the dendrogram at different levels give different partitions into communities. The optimal community cluster can be found by cutting the dendrogram at the level of maximum Q.

Louvain algorithm is a hierarchical agglomerative optimization method proposed by Blondel et al. [16] and attempts to optimize the modularity of a partition of the network. The optimization is performed in two steps that are repeated iteratively. This algorithm starts with each node in the network belonging to its own community. Then in the first step and for each node in the network, the algorithm uses the local moving heuristic to obtain an improved community structure by moving each node from its own community to its neighbours' community and evaluating the gain of modularity associated with the moving of the node. The node is then placed in the community for which the modularity change is the most positive. If none of these modularity changes is positive, the node stays in its original community. This process is applied repeatedly and sequentially for each node until all the nodes in the network are considered, and no further improvement can be achieved. This concludes the first step. The second step of the algorithm consists of building a new network from the communities discovered in the first step whose nodes are the communities. The weight of the links between communities is the total weight of the links between the nodes of these communities. Once the second step is completed, it is possible to replay the first step and iterate again if necessary. The two steps repeat iteratively and stop when there is no more change in the modularity gain and consequently a maximum modularity is obtained.

Another popular method widely used to find communities in the network is based on the random walk. An example includes Walktrap (WT) algorithm which is proposed by Pons and Latapy [17]. Walktrap algorithm is based on the principle that random walks on a network tend to get "trapped" into densely connected parts defining the communities. In this method, the authors propose using a node similarity measure based on short walks to capture structural similarities between nodes instead of modularity to identify community via hierarchical agglomeration. The algorithm starts by assigning each node to its own community, and the distance for every pair of communities is computed. Communities are merged according to the minimum of their distances and the process iterated. After $n-1$ steps, the algorithm finishes and gives a hierarchical structure of communities called a dendrogram. The best partition is then considered to be the one that maximizes modularity.

Information theoretic algorithms are another major type of community detection clustering algorithms that use the concept of information theory to find community clusters in the networks. Infomap algorithm is an example of information theoretic algorithms proposed by Rosvall and Bergstrom [18]. Infomap algorithm characterizes the problem of finding the optimal community clustering in the network as the problem of finding the most compressed (shortest) description length of the random walks on the network. It uses a random walk as a proxy for information flow in a network and minimizes a map equation, which measures the description length of a random walker, over all the network clusters to reveal its community structure. To represent the community structure, the algorithm uses a two-level nomenclature based on Huffman coding: a level to distinguish communities in the network and the other to distinguish nodes in the community. In practice, the random walker is likely to stay longer inside communities; therefore, in the process of finding a community containing few inter-community links, only the second level is needed to describe its path, leading to a compact representation.

Recently, there have been several studies [19–22] proposed to find the proper cluster for specific applications. For example in [20, 22], an intelligent clustering method for energy-efficient cluster-based routing of data packets in a wireless sensor network application have been proposed. However, the above-mentioned algorithms are classified as global algorithms, which require access to the entire information of the network and are designed to work on a single machine [3].

2.2 Clustering without global knowledge

There are other algorithms apart from DICCA [23] and PDICCA that achieve some degree of locality within the graph by considering partial information instead of global information. The examples include Connectivity-based Decentralized Node Clustering scheme (CDC) proposed by Ramaswamy et al. [24], Distributed Diffusive Clustering algorithm (DiDiC) proposed by Joachim and Henning [25] and Ja-be-Ja [10]. CDC is a distributed and scalable algorithm for discovering clusters in peer-to-peer networks. However, the nodes executing CDC algorithm need to communicate with their direct neighbours and require knowledge of all the neighbouring nodes.

Similarly, though DiDiC is designed to work based on the method of distributed diffusion to eliminate global operations, DiDiC communication takes place between neighbouring graph nodes thus requiring the knowledge about all the neighbouring nodes. Ja-be-Ja is a decentralized local algorithm that uses local search for graph partitioning; however, it is designed to find balanced size partitions rather than good-shaped partitions. This is usually not the case for real-world networks.

3 Decentralized Iterative Community Clustering Approach for graph clustering (DICCA)

DICCA is an agglomerative clustering algorithm based on random walk and reachability, which is carried out through message propagation between neighbours. There are two phases, local clustering and network reduction, that are run in an iterative fashion. The former phase is used to define an originator node for each community cluster and associate each node in the network to the best-fit originator. The reduction phase is used to rebuild the network using the communities resulting from the previous phase, where each detected community becomes a node and the weight of the edges in the new network represent the sum of the edges between two communities. The DICCA algorithm uses two parameters named threshold value and time to live (TTL) [23]. The concept of the DICCA approach is presented in Algorithm 1.

Each round of the iteration process comprises of choosing a node randomly to be an originator. The originator node acts as a cluster head and advertises itself by sending a message (Msg) to all its neighbours in the network. This message contains three fields, Originator node ID (OnID), Message Weight (WMsg) and TTL. OnID represents the node id of the originator of the message. WMsg is the weight carried by the message that represents the estimated probability of reaching any node in the network starting from the originator node. TTL represents the maximum distance in

hops before a message (Msg) expires. It is worth noting that, in order to avoid the originator being assigned to any other clusters, the WMsg is set to 1 at the originator.

Algorithm 1 The proposed DICCA approach

```
Input: underlying network graph G, time_to_live and threshold value
Output: C communities as a final division of G.
Repeat
    Outlier list ← all nodes // local clustering phase
    While outlier list ≠{}
        Oi ← Rand select (outlier list) // choose a node randomly to be an originator.
        //creat new message (Msg)
        OnId ← Oi // originator ID
        TTL ← time_to_live
        WMsg ← 1
        Msg ←{ OnId , TTL, WMsg }
        While TTL ≥ 0
            Total_weight (Oi, Vi) = sendmessages(G, Oi ,OnId, TTL, Msg) // Total //weight
            between Oi and its neighbout nodes (Vi)
            TTL ← TTL-1
            Oi ← Vi
            Msg ←{ OnId , TTL, Total_weight (Oi, Vi) }
        end while
        for each Node Vi ∈ G
            if Total_weight(Vi, onID) ≥ threshould then
                C(Vi) ← Join the cluster lead by max onID
            else
                Remain outlier
            end if
        end
        end while
        Ĝ=Aggregate (G,C) // Network reduction phase "Compact each community to one
                          // new node and build new network"
    if (C_current=C_ previous) // no membership change
    break;
  return C / / return the final division of G
end Algorithm

Function sendmessages (G, Oi ,OnId, TTL, Msg)
    for each Node Vi ∈ Nbr (Oi) do
        Send WMsg to Vi ← WMsg(Oi ,Vi)=WMsg(Oi ,Vi) *W(Oi,
        Vi)/ ∑_{Vj∈Nbr(Vi)} W(Vi,Vj)
        If Ni have seen message from onID before then
            Total_weight(Vi, Oi) ← Total_weight (Vi, Oi) + WMsg
        else
            Total_weight(Vi,Oi) ← WMsg
        end if
    end
    Return Total_weight(Vi,Oi)
end function
```

Consider two nodes, the originator O_i and its neighbouring node V_i, the model used to compute the weight of the message sent from the originator O_i to node V_i depends on the weight of the edges between O_i and V_i. This is defined as [23]:

$$\mathrm{WMsg}(O_i, V_i) = \frac{W(O_i, V_i)}{\sum_{V_j \in \mathrm{Nbr}(O_i)} W(O_i, V_j)} \tag{1}$$

Every single node in the network maintains the information about the originator IDs and the total weights of the messages it has received for each originator. This information is represented as Total Message Weight. When the node V_i receives a message Msg from its neighbouring node, it first updates the Total Message Weight value and then checks whether TTL > 0. If TTL > 0, it decrements the TTL of the message by one and forwards the message to all its neighbours except the sender.

The weight of the new message WMsg(V_i, V_k) sent from node V_i to its neighbouring node V_k is defined as [23]:

$$\mathrm{WMsg}(V_i, V_K) = \mathrm{WMsg} \times \frac{W(V_i, V_k)}{\sum_{V_j \in \mathrm{Nbr}(V_i)} W(V_i, V_j)} \tag{2}$$

However, if TTL = 0 or WMsg becomes insignificantly low compared to the predefined threshold value, the Node V_k processes the message and stops the forwarding phase.

The nodes join the closest originator O_i if the total weight of the message from the originator is greater than the specified threshold value. If not, those nodes will remain as outliers and do not join any cluster.

This procedure is iteratively repeated by adding one more originator and updating communities and outlier nodes until there is no outlier node remains left. However, some nodes may receive multiple messages generated from different originator nodes. In that case, each node attaches itself to the cluster lead by the originator from which it has received the highest total message weight.

The second phase of the algorithm uses the communities that are found in the first phase to build a new network, with each community from the previous phase represented as a node in the new network. Multiple edges between any two communities are collapsed into a single edge in the new network, and its weight being the sum of the edges between them. The edges within each community in the first phase are represented as self-loops in the new network [23].

Once the second phase is completed, the first phase process is repeated with the new network. The two phases are iteratively applied until there is no more change in the communities between two iterations, and consequently optimized community clusters are obtained.

Although the exact computational complexity of DICCA is harder to formalize, this algorithm behaves as $O(m \log(n\ m)^2))$, in which n is the total number of nodes in the network and m the number of edges. However, the most effort is in the first phase of the algorithm.

4 Description of the Parallel Decentralized Iterative Community Clustering Approach (PDICCA)

The core idea of PDICCA is to divide the data set into blocks and then iteratively repeat the following three phases: clustering, re-clustering and rebuilding phase: the clustering phase is responsible for finding local community clusters for each block independently and in parallel. In the second phase, the local clusters thus extracted from the individual blocks are aggregated to find the initial community clustering for the entire network. The third phase involves building a new, but smaller network for each block of data based on the initial community clustering. Each cycle of this process through all the three phases is referred to as an iteration. The three phases iterate until the old and the new community clustering list does not converge anymore.

4.1 Framework of the PDICCA approach

The PDICCA approach consists of two worker schemes: master and slave-clustering workers. The master worker creates the blocks as it reads the data set, and passes them to slave-clustering workers. The master worker is also responsible for receiving and aggregating the cluster assignment results from all the slave-clustering workers, perform some computation, assign the overlapped nodes into the best community and return the final solution. On the other hand, slave-clustering worker's functionality is to identify local communities by going through its own data set and applying the first phase of the DICCA approach. The overview of PDICCA approach is shown in Fig. 1.

Slave-clustering worker runs in parallel and stores the community clustering lists in its local memory. However, since each slave-clustering worker has some part of the data and does not have a global knowledge of the network, consequently, different slave-clustering workers could cluster the same node into different communities. Thereby, when all the blocks are clustered and the local communities have been identified, the master worker loads the local community clustering lists to aggregate.

Since the PDCCA approach is proposed to find non-overlapping clusters, the partition C of N nodes should form a partition such that $N = \cup_{i=1}^{k} C_i$ and $C_i \cap C_j = \emptyset$ for any $i \neq j$. So, the master worker is responsible for finding the set of overlapping nodes. The overlapping node list is then sent back to the slave workers to calculate the strength of clustering solutions for each overlapped node among different machines. This is then sent back to the master worker for the re-clustering phase. In the re-clustering phase, the master worker finds out the best solution for overlapped nodes, the solution corresponding to the highest strength of clustering, and updates the community clustering list. At the end of the re-clustering phase, the network is partitioned into a number of communities.

Next step is the rebuild phase, which involves building a new network by each of slave-clustering workers. Using the same method presented in Sect. 3 where the nodes in the new network are the communities from the re-clustering phase.

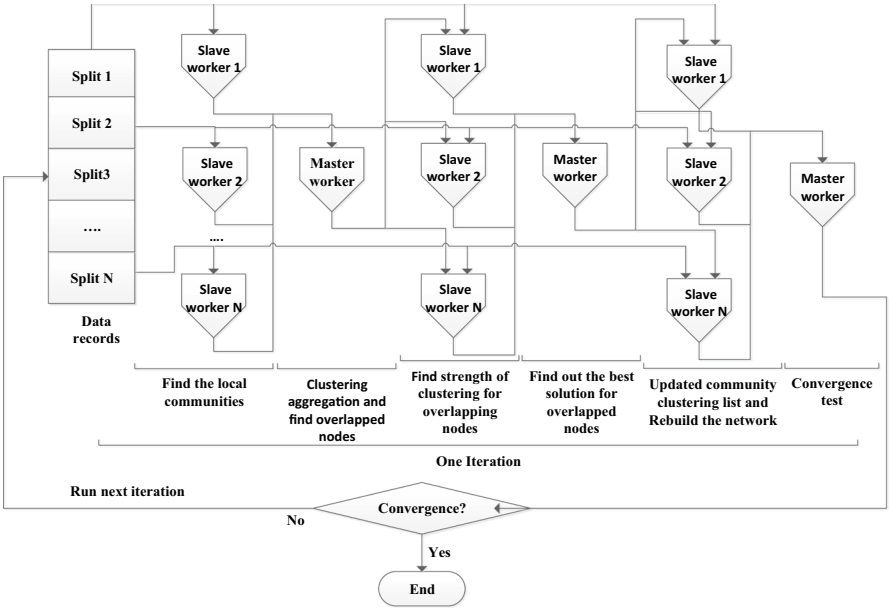

Fig. 1 Framework of the PDICCA approach

The weight of the link between two nodes in this new network is the total weight of the links between the nodes of the two corresponding communities in the original network. The links between the nodes of the same community become self-loops of the corresponding node in the new network. The iteration is then repeated until a stable set of community clusters (fulfilling the convergence condition) is obtained.

It is to be noted that each slave-clustering worker has its own private non-shareable memory, and there are no communications between the workers in the clustering phase. Thus, each slave-clustering worker operation is independent of the others and each of the slave-clustering worker's operations can be performed in parallel.

To calculate the strength of overlapped nodes, the clustering strength of overlapped node V_m is formalized in the following definition:

Definition 1 (*Cluster strength*)

Given a network set $G=(V, E)$, with $n=|V|$ nodes and $m=|E|$ edges is presented. During the clustering phase, each slave-clustering worker clusters these nodes into C clusters and assigns V_m node to different communities. To find the best community that fits V_m node, the proposed scheme carries out the following two steps.

First, the node V_m obtains two sets of information from each of its neighbours, namely the degree of the neighbour node and the cluster to which it belongs to and then calculates the neighbour attraction between V_m and its neighbour V_i, which is defined as:

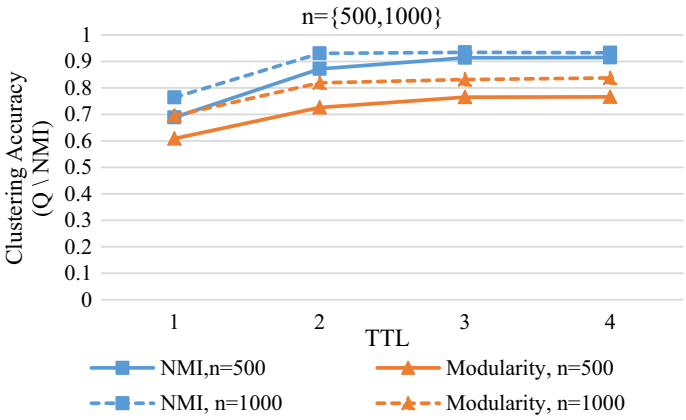

Fig. 2 Performance of the DICCA algorithm using different TTL values

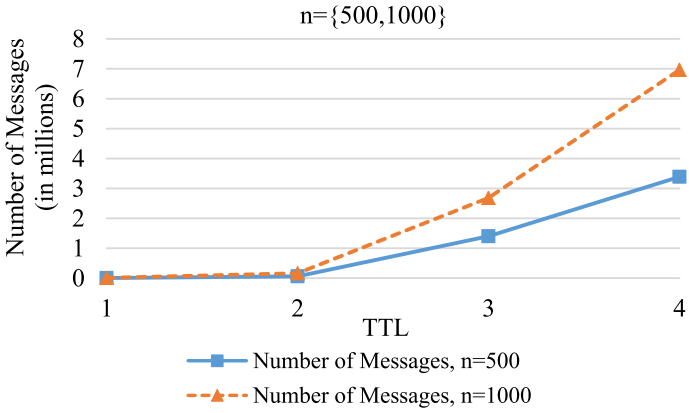

Fig. 3 Comparison between computing time and the message complexities over different TTL values

$$\text{Nbr Attraction } V_m(V_i) = \frac{W(V_m, V_i)}{\sum_{V_k \in \text{Nbr}(V_i)} W(V_i, V_k)} \tag{3}$$

where $W(V_m, V_i)$ represents the weight of the edge between V_m and V_i.

Then the strength value of V_m for all the clusters (C) where V_m belongs to is calculated by computing the sum of the attractions for V_m towards its neighbours (Nbr Attraction) within these C clusters and as follows:

$$\text{Cluster strength}(V_m, C1) = \sum_{V_i \in C1 \& V_i \in \text{Nbr}(V_m)} \text{Nbr Attraction } V_m(N_i) \tag{4}$$

5 Setting the optimal values for the parameters

As mentioned in Sects. 3 and 4, DICCA and PDICCA use two parameters to be defined. The first parameter TTL is defined as the number of hops that a message is permitted to travel before being discarded. The next parameter is threshold value that determines the difficulty of merging communities.

5.1 Time to live

Each message has a time to live (TTL) field that is initiated with some value $T>0$ that limits the number of times the message is forwarded. In reality, care needs to be applied in choosing an appropriate TTL value, because a small TTL value means the message may expire before reaching all relevant nodes in the network. On the other hand, a large TTL means more nodes than needed are visited, thus increasing both the message load on the network and the running time of the algorithm. Therefore, in this work, it is proposed to rebuild the network before starting a new iteration to address this issue. Furthermore, based on the real-world network properties, it is stated that networks from real-world applications are often small-world networks [26, 27]. The small-world concept in simple terms describes the fact that even if the network has many nodes, there exists a relatively small number of intermediate steps (short path) connecting any pair of nodes within the network [28]. For example in social networking sites, it is stated that people (and things and places) in the world are just six or fewer interpersonal connections away from each other [29]. This is known as six degrees of separation theory.

In order to determine the effect of TTL value on the community clustering accuracy, the TTL value ranging from 1 to 4 has been used in this evaluation. Figures 2 and 3 present the accuracy values of synthetic networks with 500 and 1000 nodes and the message complexity, respectively. The results demonstrate that the DICCA yields good community clusters when the TTL is set to be 3. Increasing the TTL value does not have significant impact on the quality of community detection but may result in a very high communication load. However, selecting a small TTL value can reduce the broadcast overhead but will compromise the accuracy.

In this work to achieve good trade-off between high modularity and low message complexity (running time), TTL is set to a value of 3.

5.2 Threshold value

The threshold value is a parameter set at the beginning of the process in a range between 0 and 1. If the total weight of the message received by the node from originator O_i is equal to or greater than the threshold value, then the node is able to join the cluster led by the O_i. This means the higher the threshold value there is less chance for the node to be merged into the community. For example, setting

the threshold value close to zero will produce a single community cluster containing all the nodes in the network. On the other hand, setting the threshold value close to one will make each node to be in its own cluster. In other words, low threshold value produces high number of small-sized clusters; meanwhile higher value will produce lower number of larger sized clusters. Therefore, the threshold value has an important effect on clustering accuracy as well as the size of the detected clusters. Obviously tuning the threshold value could be seen as a possible practical remedy to control the desired size and the number of communities.

The choice of selecting a suitable threshold value is very crucial and requires a priori knowledge of network structure. However, generating a priori knowledge is usually time-consuming since networks are usually big and have large amounts of information [29]. Hence, in this work a mathematical model to automatically calculate the threshold value is proposed by the authors. The model makes use of density, size and layout structure of the network to find the optimal threshold value.

The set of Eqs. (5 to 11) presented below define the mathematical model for setting the threshold value for undirected network:

$$\text{Threshold value} = \text{avg\_}t + (t-1) \times ((1-C) \times \text{avg\_}t) \tag{5}$$

$$\text{avg\_}t = \frac{\log(\log(n))}{\log(n)} \sum_{i=1}^{n} \left(\frac{1}{K_i} + \frac{K_i - 1}{K_i^2} + \frac{K_i - 2}{K_i^3} \right) \tag{6}$$

$$K_i = \sum_{j=0}^{n} A_{ij} \tag{7}$$

where i is the main node, j represents all other nodes, A is the adjacency matrix where the entry A_{ij} represents the connectivity if the value is 1; otherwise, it is 0, n is the total number of nodes in the network, t is the iteration number, K_i is the degree of node i and C is network clustering coefficient which is defined as:

$$C = \frac{1}{n} \sum_{i=1}^{n} \frac{2L_i}{K_i[K_i - 1]} \tag{8}$$

where L_i is the number of edges between neighbours of node i [8].

A fully connected network is a simple undirected graph in a network of n nodes, in which every pair of distinct nodes is connected by a unique edge. Based on the graph theory, the network clustering coefficient for a fully connected network is 1 and the degree of each node is defined as:

$$K_i = n-1 \tag{9}$$

Thus, the total edges of the complete network having n nodes will be:

$$\sum_{i=0}^{n} K_i = n(n-1) \tag{10}$$

Using Eq. (5) to calculate the threshold value for complete network:

$$\text{Threshold Value} = \frac{\log(\log(n))}{\log(n)} \sum_{i=1}^{n} \left(\frac{1}{K_i} + \frac{K_i - 1}{K_i^2} + \frac{K_i - 2}{K_i^3} \right) \tag{11}$$

In Eq. (11), the first part $\left(\frac{\log(\log(n))}{\log(n)}\right)$ is always less than 1. Whereas, the second part of equation $\left(\sum_{i=1}^{n} \left(\frac{1}{K_i} + \frac{K_i-1}{K_i^2} + \frac{K_i-2}{K_i^3}\right)\right)$ represents the maximum weight of messages received by node i, when the TTL = 3. For a fully connected network and using Eqs. (5–11) to set the threshold parameter, the proposed algorithm will produce one cluster containing all the nodes in the network. This is acceptable since there is no meaningful subsets that are clusters in the fully connected network.

It is worthwhile mentioning that, in each iteration, the threshold value for a given network is stepwise increased by $(t-1) \times ((1-C) \times \text{avg\_}t)$ as seen in Eq. (5), so that it becomes progressively difficult for clusters that are not so densely connected to join with each other. Only the strongly connected ones will be able to merge. Additionally, the maximum threshold value cannot be larger than 1 [23].

6 Experimental data sets

To analyse the efficiency of the community detection algorithm on a range of network size and due to the scarce availability of real networks that have ground-truth communities, LFR benchmark is used to generate synthetic data sets. The LFR benchmark model was proposed by Lancichinetti et al. [30] to generate undirected and unweighted networks that closely resemble real-world networks with community structure. LFR model has become a popular choice for assessing the performance of community detection algorithms, and the model was subsequently extended to generate weighted and/or directed networks, with the possibility of overlapping communities.

Most of the real-life network have been defined and modelled as undirected and unweighted/weighted networks. This paper focuses on this type of networks with non-overlapping communities. The LFR model is proposed to address most characteristics of real networks, e.g., size of the network and heterogeneous degree distribution. In the LFR benchmark, both the node degrees of a network and the size of each community are controlled by a power-law distribution with exponent γ and β, respectively. However, it has been observed that real-world graphs have such a power-law degree distribution [28] with typical values of: $2 \leq \gamma \leq 3$, $1 \leq \beta \leq 2$ [30].

LFR model provides some other parameters to control the network topology, including the number of nodes, maximum degrees and mixing parameter μ. Mixing parameter $\mu \in [0, 1]$ is used to control the fraction of intra-cluster and inter-cluster edges on the network. For small values of μ, there will be small number of edges going outside the communities, which indicates that there are clear clusters available in the networks. The larger the μ value, the more challenging it is to detect communities in the network. The code of LFR mode is made publicly available by the authors [31].

7 Analysis of results and discussion

7.1 Environment setup

Using the LFR networks a set of undirected networks are generated. The default benchmark parameter values are used as the benchmark parameters for the exponents of the degree distribution and community size, viz. $\gamma=2$, $\beta=1$. The average degree and the maximal degree are 25 and 50, respectively. The mixing parameter is varied from 0.1 to 0.75 and the number of nodes is varied from 500 to 5000.

The DICCA and PDICCA are implemented in Matlab and the experiments are performed on a system configured with 4® Core™ i7 6700 K CPU 4.00 GHz and 16 RAM available memory running windows. Because the approaches initialize the originator randomly, and in order to neglect the effect of randomness in our method each result is averaged over 100 runs.

7.2 Accuracy measure for graph clustering

The true community structure (ground truth) is known for the benchmark network. Therefore, Normalized Mutual Information (NMI) [32] is used to evaluate the performance of DICCA and PDICCA by comparing the obtained partitions in the experiments with the ground truth for the LFR benchmark. NMI metric quantifies the accuracy of the proposed methods by evaluating the level of correspondence between detected and ground-truth communities. In addition, modularity measurement introduced by Newman and Girvan in [33] is used to evaluate the effectiveness of the algorithms in terms of modularity optimization.

Definition 2 [*Normalized Mutual Information (NMI)*]

Normalized Mutual Information (NMI) is a similarity measure for comparing two partitions based on the information theory concept. It is introduced in the community detection domain by Danon et al. [32], and since then it has been widely used to evaluate the accuracy of community detection algorithms.

For an n-node network with two partitions $X=\{X_1, X_2, X_3, \ldots, X_k\}$ and $Y=\{Y_1, Y_2, Y_3, \ldots, Y_{\bar{K}}\}$ where X and Y represent the real communities and found communities, respectively, the normalized mutual information NMI(X, Y) of two divisions X and Y of a network is defined as follows [34]:

$$\mathrm{NMI}(X,Y)=\frac{-2\sum_{K=1}^{k}\sum_{\bar{K}=1}^{\bar{K}}P\left(K,\bar{K}\right)\mathrm{Log}\left[\frac{P\left(K,\bar{K}\right))}{P(K)P\left(\bar{K}\right)}\right]}{\sum_{K=1}^{K}P(K)\mathrm{Log}[P(K)]+\sum_{\bar{K}=1}^{\bar{K}}P\left(\bar{K}\right)\mathrm{Log}\left[P\left(\bar{K}\right)\right]} \tag{12}$$

where $\left(K,\bar{K}\right)=\frac{X_K\cap Y_{\bar{K}}}{n}$, $P(K)=\frac{X_K}{n}$ and $P\left(\bar{K}\right)=\frac{Y_{\bar{K}}}{n}$.

If the found partition by the algorithm is identical to the real community, then NMI takes its maximum value of 1. If the partition found is totally independent of the real partition then NMI = 0 [34].

Definition 3 [*Modularity (Q)*]
Modularity (Q) is a prominent measure for the quality of a community structure, and it has become a widely accepted quality of measure for community detection. Modularity states that a good cluster should have a bigger-than-expected number of connections between the nodes within modules and a smaller-than-expected number of connections between nodes in different modules. The higher the value of modularity, the better is its community strength.

The general concept of modularity optimization algorithms is to detect the best community structure in terms of modularity by searching over possible divisions of a network that have high modularity.
Formally, modularity can be defined as [3]:

$$Q = \frac{1}{2|m|} \sum_{ij} \left[A_{ij} - \frac{K_i K_j}{2|m|} \right] \delta_{c_i c_j} \tag{13}$$

where A_{ij} is an element of the adjacency matrix, K_i is the degree of node i. $\delta_{c_i\, c_j}$ is the Kronecker delta symbol, which is equal to 1 if $c_i = c_j$ and 0 otherwise, and c_i is the label of the community to which node i is assigned.

7.3 Experimental results

In this section, the results from the experiments conducted using synthetic networks are presented, analysed and discussed in detail.

7.3.1 Horizontal scalability in relation to the number of parallel cores

To demonstrate how well the proposed approaches handle data sets when more workers are available, the number of nodes in the network used in this evaluation is kept constant and the number of workers is varied from 1 to 4. It is worth mentioning that if the number of workers is 1, the algorithm simply represents DICCA. Figure 4 shows the results of different cores when the number of nodes is constant, $n \in \{500, 1000\}$.

7.3.1.1 Quality From Fig. 4, the approaches show a good scalability close to the optimal value, which is indicated by average modularity and NMI values. In addition, it is clear that using more than one worker to parallelize the algorithm does not adversely affect the accuracy of the result. Consequently, the results prove that the algorithm is effective and able to achieve very high-quality results in a parallel manner. More especially, PDICCA is capable of exploiting multi-core architecture efficiently.

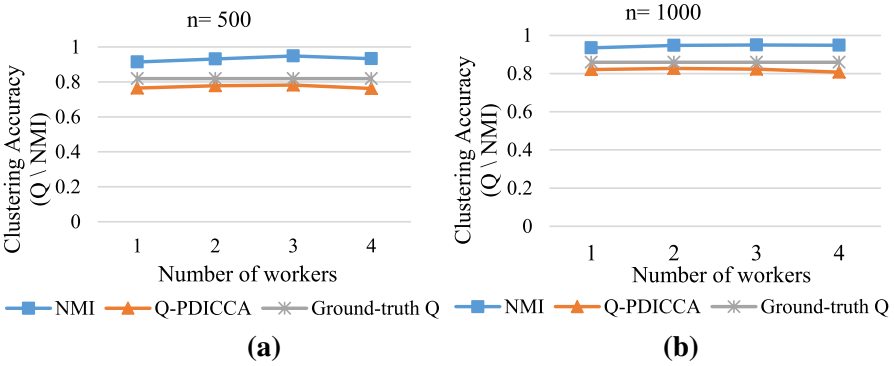

Fig. 4 NMI, Q-PDICCS and ground-truth Q scores (y-axis) as number of workers (x-axis) changes number of nodes: **a** 500 and **b** 1000

7.3.1.2 Message complexity Considering the number of exchanged messages for each worker, Fig. 5 shows the percentage of exchanged messages at each iteration by each worker processor. As can be observed in each iteration, each worker generates almost the same number of messages; this can be clarified by the fact that the data has been partitioned equally among the workers so each worker has to process the same size of data. Hence, at each iteration, the master worker must wait until all workers have completed their processes. So, splitting the data equally over workers can significantly reduce the expected time needed to wait until the slowest machine worker returned data.

For more in-depth analysis, Fig. 6 shows the average percentage of exchanged messages in each iteration. It can be easily observed from the figure that data exchange for the algorithm is much greater at the first stage of iteration when each node is in its own cluster. Just after 2 to 3 initial iterations, most nodes have their cluster labels and the algorithm has merged the nodes belonging to the same cluster to be one node. It also becomes clear from the percentage of exchanged messages between master and slaves as seen in Table 1, the communication cost is negligible. However, this is less compared with the cost of information exchanged locally in slaves, which is costly and constitutes the main body of the time consumption of the algorithm.

7.3.2 Clustering results for increasing network size

To demonstrate the performance influenced by scalability, the number of nodes is increased linearly from 500 to 5000 and the number of workers is kept constant at 1 and 3. All other parameters and factors remain the same as previous evaluations.

7.3.2.1 Quality The modularity values of the solutions obtained by the DICCA and PDICCA are presented in Fig. 7. It can be observed from the figure that the performance of the both DICCA and PDICCA are consistently good and close to the optimal value with NMI above 0.90 on average.

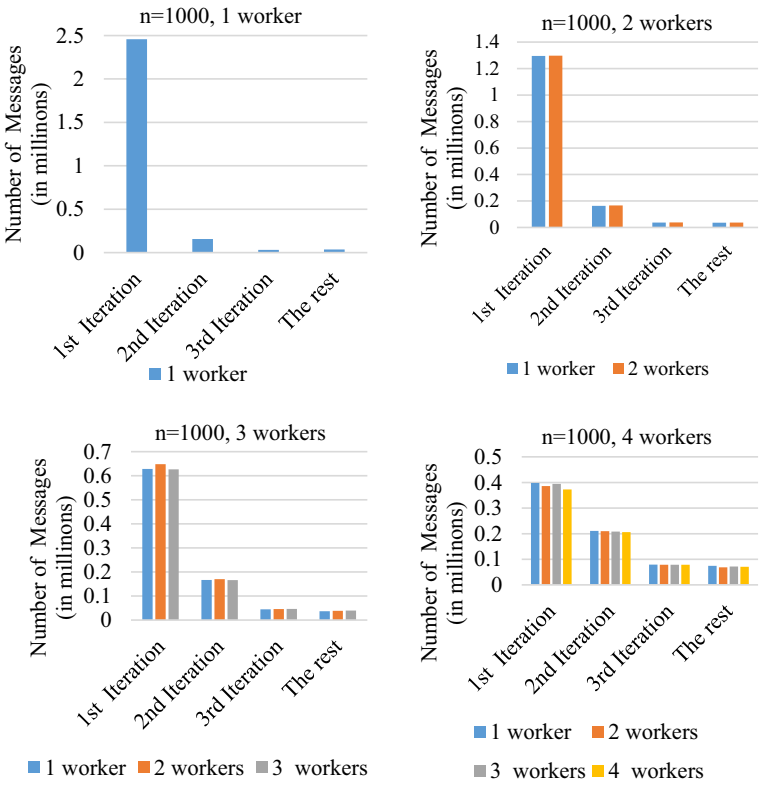

Fig. 5 Number of message exchanged in each iterations and for each worker with respect to the number of workers varied from 2 to 4 for number of nodes 1000

7.3.2.2 Evaluating repeatability of the performance To further investigate the ability of the DICCA and PDICCA to produce consistent results across random starts across random data partitioning and initialization, the standard deviation of the clustering results is measured where both the DICCA and PDICCA are run 100 times each time with different random data partitioning and algorithm initialization. The standard deviation value of both NMI and modularity for the data sets with different network size are displayed in Fig. 8, which is relatively very small and in some cases around zero variation.

7.3.2.3 Evaluation of complexity of the DICCA and PDICCA approaches To investigate the relationship between the number of nodes and complexity of approach, the total number of exchanged messages as a function of the network size is presented in Fig. 9. Since the DICCA and PDICCA require a large number of exchanged messages between nodes, which is the most time-consuming part during execution, the performance of DICCA and PDICCA highly depended on the total number of exchanged messages. Therefore, the number of exchanged messages increases with the network size. For example, the total number of messages exchanged by PDICCA for $n \in \{500; 5000\}$ is {1,344,282; 15,633,691}, respectively.

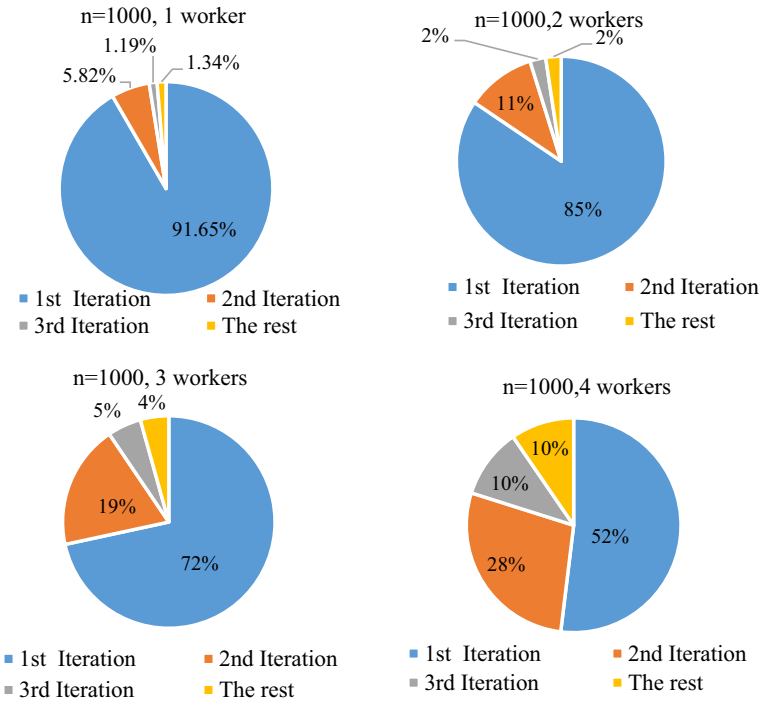

Fig. 6 Average percentage of message exchanged per each iteration with number of cores varied from 1 to 4 workers for network size 1000

Table 1 Comparison with message exchanged locally in hosts and messages exchanged between master and hosts

Number of nodes	500		1000	
No. of workers	% Messages exchanged locally among slaves	% Messages exchanged between master and slaves	% Messages exchanged locally among slaves	% Messages exchanged between master and slaves
1	100	0	100	0
2	99.9767	0.0233	99.9760	0.0240
3	99.9636	0.0364	99.9631	0.0369
4	99.9599	0.0401	99.9629	0.0371

7.3.3 Evaluation of clustering performance using mixing parameter

To investigate the ability of the DICCA and PDICCA to detect community clusters when the community structure is weakened by highly increasing the number of inter-community links (occurs when μ of the LFR benchmark has high values), the DICCA and PDICCA are evaluated with varying values of mixing parameter between 0.1 and 0.75, $\mu \in \{0.1, 0.15, \ldots, 0.75\}$, and keeping the number of nodes constant,

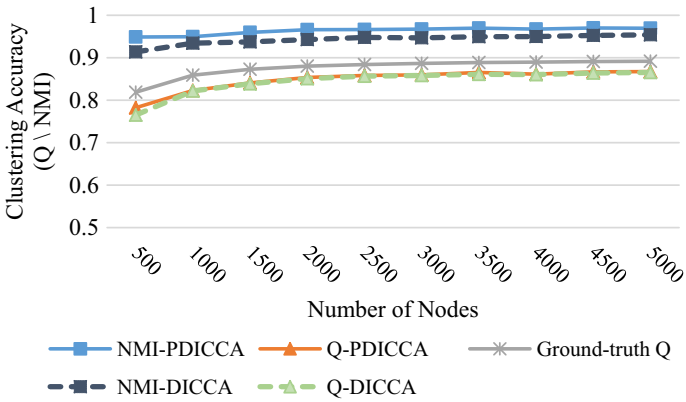

Fig. 7 NMI, Q and ground-truth Q scores (y-axis) as number of nodes (x-axis) changes

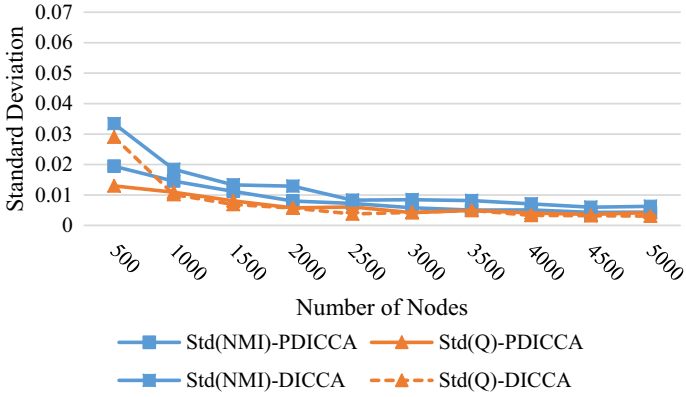

Fig. 8 Standard deviation of final modularity/NMI with network sizes

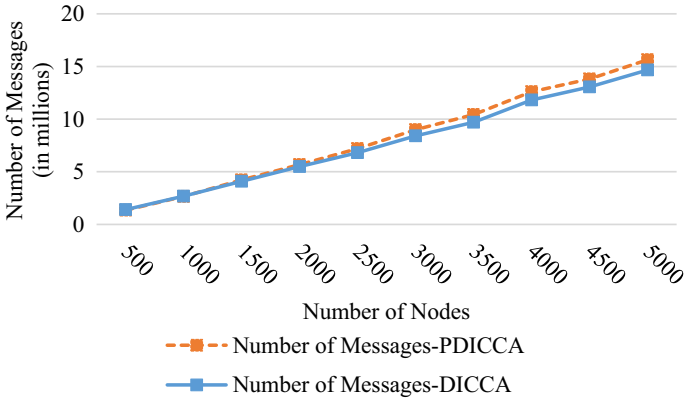

Fig. 9 Total number of exchanged messages (y-axis) as number of nodes (x-axis) changes

$n \in \{1000\}$. Figure 10 shows the results obtained for both modularity and NMI accuracy as a function of the mixing parameter for network sizes 1000 nodes. As can be clearly seen, the natural partitions of the network are always found (in principle) for the mixing parameter value of up to 0.5, after which the method starts to fail where the quality of DICCA and PDICCA was rather poor. The reason for this behaviour is that a small value of mixing parameter indicates well-defined community structures in the generated network. The reason is that most of the edges fall inside the communities. This is in line with the definition of a community that each node should have more connections within the community than with the rest of the graph [27]. On the other hand, networks with higher values of mixing parameter are more challenging to cluster accurately, as there are no clear divisions between communities in these networks as most of the edges fall outside the communities. Furthermore, the performance comparison of the proposed algorithms, the ground-truth network and the fast greedy modularity optimization proposed by Clauset et al. [35] in terms of modularity is shown in Fig. 10. This comparison shows low values of Q for all the algorithms considered here and the ground-truth network, when the mixing parameter value ≥ 0.5. This low value of Q indicates that the communities in the network are indistinguishable due to the network structure rather than poor performance.

8 Conclusion

In this paper, a novel Decentralized Iterative Community Clustering Approach (DICCA) and its parallel version (PDICCA) to extract an efficient community structure for large networks are presented. An important property of the proposed approaches is their ability to identify optimal community clusters from an entire network without the global knowledge of the network topology. This ability means that

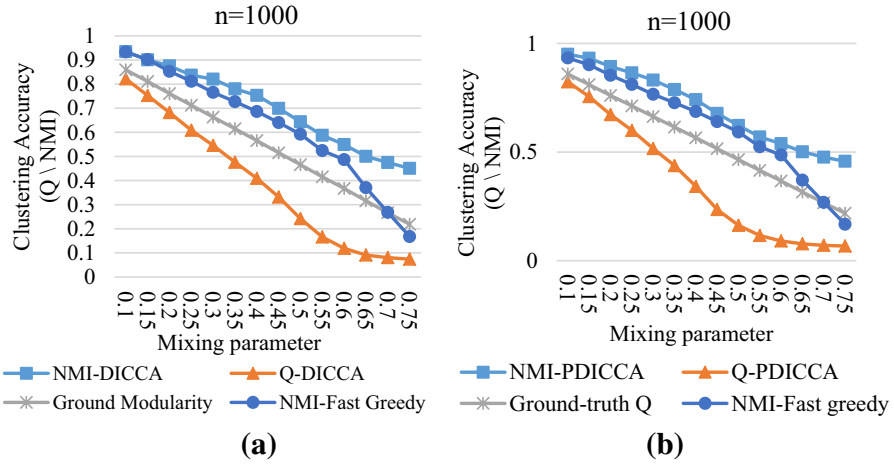

Fig. 10 Performance of the proposed algorithm using mixing parameter μ. **a** DICCA and **b** PDICCA

the entire network does not need to be loaded into one memory and could be easily adapted to run in parallel on as many processors as available to find community clusters in big networks. This cannot be done using the majority of existing community detection algorithms that implicitly assume that the entire structure of the network is known and is available.

The DICCA and PDICCA are based on the random walk procedure and reachability of nodes in the network. In addition, the proposed approaches address the issues surrounding computational demands for dealing with big data sets. They optimally utilize the hardware capabilities of modern multi-core systems for faster execution by processing multiple blocks in a parallel manner. Furthermore, when scalability issues occur as the data size grows beyond the processing power of a single machine, the proposed distributed approach based on the MapReduce computing platform will help address this. Finally, the effectiveness and complexity of the approaches are tested and analysed using synthetic networks with ground-truth communities. The experimental results of the approaches prove to be very promising.

Real-world networks often do not contain perfect communities, and in reality nodes may belong to multiple communities simultaneously. Identifying such overlapping communities (also known as fuzzy) is crucial for understanding the structure as well as the function of real-world networks. A further direction is to extend the proposed approaches to be able to detect such fuzzy communities. Further, in this work, only the undirected networks have been taken into consideration. Therefore, considering the directed networks may be an interesting direction for further research.

References

1. Chen J, Zaiane OR, Goebel R (2009) Detecting communities in large networks by iterative local expansion. In: International Conference on Computational Aspects of Social Networks, 2009. CASON'09, pp 105–112. IEEE
2. Mahata D, Patra C (2016) Detecting and analyzing invariant groups in complex networks. In: Behera H, Mohapatra D (eds) Computational intelligence in data mining, vol 1. Springer, New Delhi, pp 85–93
3. Fortunato S (2010) Community detection in graphs. Phys Rep 486(3):75–174
4. Orman GK, Labatut V, Cherifi H (2011) On accuracy of community structure discovery algorithms. ArXiv preprint arXiv:1112.4134
5. Schaeffer SE (2007) Graph clustering. Comput Sci Rev 1(1):27–64
6. Khatoon M, Banu WA (2015) A survey on community detection methods in social networks. Int J Educ Manag Eng 5(1):8
7. Warnke SD (2016) Partial information community detection in a multilayer network. Naval Postgraduate School, Monterey
8. Costa LF, Rodrigues FA, Travieso G, Villas Boas PR (2007) Characterization of complex networks: a survey of measurements. Adv Phys 56(1):167–242

9. Qi G-J, Aggarwal CC, Huang T (2012) Community detection with edge content in social media networks. In: 2012 IEEE 28th International Conference on Data Engineering (ICDE), pp 534–545. IEEE
10. Rahimian F, Payberah AH, Girdzijauskas S, Jelasity M, Haridi S (2013) Ja-be-ja: a distributed algorithm for balanced graph partitioning. In: 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pp 51–60. IEEE
11. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
12. Wang M, Wang C, Yu JX, Zhang J (2015) Community detection in social networks: an in-depth benchmarking study with a procedure-oriented framework. Proc VLDB Endow 8(10):998–1009
13. Aggarwal CC, Wang H (2010) A survey of clustering algorithms for graph data. In: Managing and mining graph data, pp 275–301. Springer, Boston, MA
14. Newman M (2010) Networks: an introduction. Oxford University Press, Oxford
15. Newman ME (2004) Fast algorithm for detecting community structure in networks. Phys Rev E 69(6):066133
16. Blondel VD, Guillaume J-L, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech: Theory Exp 2008(10):P10008
17. Pons P, Latapy M (2006) Computing communities in large networks using random walks. J Graph Algorithms Appl 10(2):191–218
18. Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci 105(4):1118–1123
19. Ganapathy S, Kulothungan K, Yogesh P, Kannan A (2012) A novel weighted fuzzy C-means clustering based on immune genetic algorithm for intrusion detection. Proc Eng 38:1750–1757
20. Munuswamy S, Saravanakumar JM, Sannasi G, Harichandran KN, Arputharaj K (2018) Virtual force-based intelligent clustering for energy-efficient routing in mobile wireless sensor networks. Turk J Electr Eng Comput Sci 26(3):1444–1452
21. Priya P, Ghosh D, Kannan A, Ganapathy S (2014) Behaviour analysis model for social networks using genetic weighted fuzzy c-means clustering and neuro-fuzzy classifier. Int J Soft Comput 9(3):138–142
22. Thangaramya K, Logambigai R, SaiRamesh L, Kulothungan K, Ganapathy AKS (2017) An energy efficient clustering approach using spectral graph theory in wireless sensor networks. In: 2017 Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM), pp 126–129. IEEE
23. Bhih A, Johnson P, Nguyen T, Randles M (2017) Decentralized Iterative Community Clustering Approach (DICCA). In: 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), pp 1–7. IEEE
24. Ramaswamy L, Gedik B, Liu L (2005) A distributed approach to node clustering in decentralized peer-to-peer networks. IEEE Trans Parallel Distrib Syst 16(9):814–829
25. Gehweiler J, Meyerhenke H (2010) A distributed diffusive heuristic for clustering a virtual P2P supercomputer. In: 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Ph.D. Forum (IPDPSW), pp 1–8. IEEE
26. Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world' networks. Nature 393(6684):440–442
27. Silva TC, Zhao L (2016) Machine learning in complex networks. Springer, Berlin
28. Newman ME (2003) The structure and function of complex networks. SIAM Rev 45(2):167–256
29. Griffiths MD, Kuss DJ, Demetrovics Z (2014) Social networking addiction: an overview of preliminary findings. In: Rosenberg KP, Feder LC (eds) Behavioral addictions. Elsevier, New York, pp 119–141
30. Lancichinetti A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. Phys Rev E 78(4):046110
31. Fortunato S. Benchmark graphs for testing community detection algorithms. www.santo.fortunato.googlepages.com/benchmark.tgz
32. Danon L, Diaz-Guilera A, Duch J, Arenas A (2005) Comparing community structure identification. J Stat Mech: Theory Exp 2005(09):P09008
33. Newman ME, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2):026113
34. Labatut V (2015) Generalised measures for the evaluation of community detection methods. Int J Soc Netw Min 2(1):44–63

35. Clauset A, Newman ME, Moore C (2004) Finding community structure in very large networks. Phys Rev E 70(6):066111