A Hadoop Based Approach for Community Detection on Social Networks Using Leader Nodes

Mohamed Iqbal
School of Computer Science and Engineering,
VIT-AP University, India
iqbalmecse@gmail.com

Kesavarao Latha
Department of Computer Science and
Engineering, Anna University, India
drklatha@aubit.edu.in

Abstract: *Community detection is the most common and growing area of interest in social and real-time network applications. In recent years, several community detection methods have been developed. Particularly, community detection in Local expansion methods have been proved as effective and efficiently. However, there are some fundamental issues to uncover the overlapping communities. The maximum methods are sensitive to enable the seeds initialization and construct the parameters, while others are insufficient to establish the pervasive overlaps. In this paper, we proposed the new unsupervised Map Reduce based local expansion method for uncovering overlapping communities depends seed nodes. The goal of the proposed method is to locate the leader nodes (seed nodes) of communities with the basic graph measures such as degree, betweenness and closeness centralities and then derive the communities based on the leader nodes. We proposed Map-Reduce based Fuzzy C-Means Clustering Algorithm to derive the overlapping communities based on leader nodes. We tested our proposed method Leader based Community Detection (LBCD) on the real-world data sets of totals of 11 and the experimental results shows the more effective and optimistic in terms of network graph enabled overlapping community structures evaluation.*

Keywords: *Social network, big data, map reduce, community detection, leader nodes*.

Received February 22, 2022; accepted March 12, 2023
https://doi.org/10.34028/iajit/20/6/2

1. Introduction

An essential issue in complex social network research is community identification [8]. It is essential to comprehending how these networks operate. Networks originating from various application fields, such as online social networks, collaborative networks, information networks, networks of web pages, biological networks, etc., are addressed. A community in a network graph is a collection of nodes with a disproportionately higher density of edges between them than between them and the other nodes [5]. Community detection's goal may vary depending on the application domain. For instance, recognizing communities in online social networks is akin to locating individuals who have a shared interest. In web page networks, it entails assembling web sites that are pertinent to one or more themes. Many studies have been done on finding communities in networks. The most widely used techniques include improving the modularity quality function. Unfortunately, a number of issues with the modularity optimization techniques have been discovered. In fact, because of its resolution limitations, even loosely related tiny groups are often merged to produce bigger ones. Moreover, it often splits sizable groupings, even when they are quite dense. We propose a new approach that enables the detection of more realistic communities while maintaining modularity score as close as possible to the maximum. This method aims to overcome some of the limitations of the modularity optimization technics, particularly the splitting of large dense groups and the overfitting.

In this paper, community detection problems to a given network graph modeled as a clustering problem. We have proposed the leader-based community detection technique based on fuzzy clustering algorithm [10]. We integrate fuzzy logic into clustering process to find the overlapping communities. Here we have proposed a modified version of Parallel Fuzzy C-Means (FCM) clustering based on HADOOP Map-Reduce Framework. We identify the communities from the given network using our Parallel FCM clustering algorithm, based on the leader nodes. Leader nodes are identified from the structural centrality of the node. The structural centrality is extracted from the basic centrality measurements (degree, closeness, and betwenness) and the relative distance of the nodes in the network. In this article we are dividing our work to following sections: section 2 related work of proposed method. Section 3. Architecture of our proposed work; section 4. Finding all pair shortest path using giraph Application Programming Interface (API); section 5. Calculating the degree of influence; section 6. Leader node identification; section 7. Community generation; section 8. Results and discussions; section 9 Conclusion and future work.

2. Related Work

There are two basic kinds of community identification methods: discovering communities without overlaps, where a node may only belong to one community, and detecting communities that overlap. A node can be shared by many communities in overlapping communities. In recent years there are several community detection methods without overlap are proposed like graph partitioning [20], hierarchical clustering [15], Spectral Clustering [4], optimization algorithms enabled modularity approach [11, 19]. These methods enable the social networks of complex structures to produce the reasonable communities. The graph partitioning method focus on the improvement in the modularity function. This method divides the optimisation process into two steps that are iteratively repeated. The first stage begins by giving each network node its own community (each community contains one and only one node). The next step is to relocate each node to the nearby community that produces the greatest gain in modularity (if no increase is possible, then node remains in its original community). A community to which node is connected is referred to as a neighbouring community. The algorithm gathers nodes belonging to the same community in the second phase and "builds" a new network with the communities from the first phase as its nodes. The connections linking the two matching communities are used to compute the edges between the new nodes. Up until the modularity achieves its local maximum, these procedures are repeated. In Hierarchical clustering, multilevel graph of grouping structures is established. This grouping is determined based on similarity score between the nodes in the graph. Here the similarity score is calculated using the neighbourhood similarity-based indices. The spectral methods enable to the spectral characteristics used to point out the communities. Here, computing the first k eigenvectors of the Laplacian or other matrix is necessary for spectral clustering. In general, spectral techniques are computationally intensive, and computing the eigenvectors for huge networks is mathematically impossible. Modularity-based approaches try to finding the community structure that maximizes the modularity function. This approach is greedy technique which is repeatedly looking the maximum modularity score. The inability of modularity-based algorithms to discover divisions with various sizes is one of its inherent limitations. All the algorithms discusses so far only assign the single community to each node and failures to detect the structures of overlapping community in networks. So there is rapid growth of interest underlying in uncovering overlapping community and the nodes. The several detecting methods for overlapping communities established currently. Palla*et et al*. [16] proposed an overlapping sets of cliques enabled Clique Percolation Method (CPM) depending on the assumption of the community. CPM detects all the size k cliques and identifies the structures of the community for neighbourhood cliques by searching. The community structure with the dense connected parts that enables unsuit for the networks. Based on link clustering for community detection, Ahn *et al*. [2] introduced a Link Clustering (LC) technique grouping extensively the nearest links with an equality measurement. LC commonly specifies the overlap of between communities in node communities from making the transformation among the link communities. Moreover, there is no LC displays guarantee as to attain the high-performance results. Because it emphasizes internal community links then avoids the unwanted links, which creates the multiple small communities. Also, to identify the overlapping community structures through the use of label propagation algorithm [17] by permitting a node to exposes the several labels like COPRA [7] and Speaker-listener Label Propagation Algorithm (SLPA) [21]. The distribution of label attains the detection of overlapping community with a liner time but also show some label clustering in as non-determinacy manner in the network. The proposed method utilizes the seed nodes to generate the community clusters. The seed nodes are identified based on proposed Degree of influence measure. The proposed method avoids the generating of small no of giant communities and large no of small communities by generating reasonable size communities with good modularity score described in results and discussion section.

2.1. Discussions

The main objective of the proposed work is to derive the overlapping communities from the given social network. Since social network is logically represented as directed or undirected graph, the basic centrality measures of the graph such as Degree, closeness and betweenness are utilized in this work. A new measure of Degree of Influence is proposed to find the seed nodes from the social network graph in order to start the community detection process. The Fuzzy c means based community detection process is proposed to find the overlapping community structure in the given social network graph. The entire process of the proposed work is parallelized under the Hadoop Map Reduce framework and the designed Map Reduce jobs are tested on variable size Hadoop clusters.

2.2. Managerial Benefits

Detecting the communities are useful in many applications such as detecting criminal in social network, detecting spam in emails, developing recommendation systems [12]. The proposed method having following benefits compare to other existing methods,

- The proposed work detects the communities using only basic graph measures without the need of personal information of each node in social network.
- Since the entire proposed work is parallelized, so that time complexity will be reduced.
- Community information can be utilized for predicting new links in social networks such as suggesting new friends on Facebook and Twitter.
- In real world social networks, a user always associates with many communities. So, the proposed method focused on this aspect and derived the overlapping communities based on Fuzzy C means clustering method.

3. Proposed Architecture

The proposed method's architecture depicted in Figure 1. The proposed methodology have the following six steps.

- *Step* 1: Finding all pair shortest path for given social network graph using Giraph API.
- *Step* 2: Finding degree, betweenness and closeness centralities using HADOOP Map Reduce.
- *Step* 3: Calculating Degree of Influence (DI) by aggregating all the centralities identified in step 2.
- *Step* 4: Identifying leader nodes by calculating the structural centralities
- *Step* 5: Node vector formation using leader nodes
- *Step* 6: Generating soft and hard community clusters using Parallel Fuzzy C-Means Clustering (FCM) Algorithm.

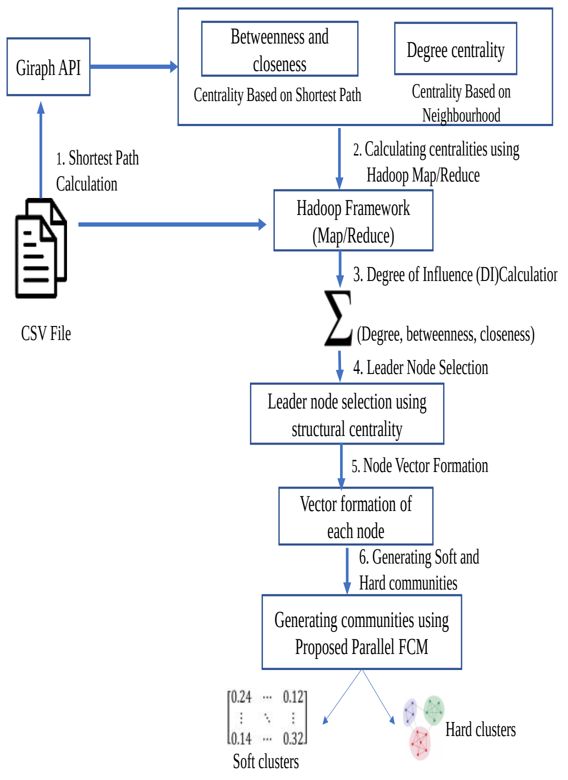

Giraph API
Betweenness and closeness
Centrality Based on Shortest Path
Degree centrality
Centrality Based on Neighbourhood
1. Shortest Path Calculation
2. Calculating centralities using Hadoop Map/Reduce
Hadoop Framework (Map/Reduce)
3. Degree of Influence (DI)Calculatior
CSV File
(Degree, betweenness, closeness)
4. Leader Node Selection
Leader node selection using structural centrality
5. Node Vector Formation
Vector formation of each node
6. Generating Soft and Hard communities
Generating communities using Proposed Parallel FCM
Soft clusters
Hard clusters

Figure 1. Architecture of proposed method.

4. Finding All Pair Shortest Path Using Giraph

The social network file (CSV format) goes to the Giraph [9] job input, which performs the processing needed and produces the shortest output paths and writes them into a file. From a source node in the network, the shortest path algorithm used for one or several super steps to find the shortest path. A superstep is defined as the interaction, barrier synchrization, and the association of computation. In each superstep, a subset of nodes or all the nodes perform the local computation. The participating nodes after computation alter the information with their neighbors in a superstep by the set of nodes. Each node has the interest to the next superstep participation after the message(s) received. If during the interaction process a node receives nothing, it never executes in the next superstep condition. In current superstep, active node indirectly stops from the participation in the next superstep when the computation communication approach. But there may be some scenario that its neighbors have also received some messages from an active node. In such instances, in the next superstep must remain active in the active node. In the current superstep, the next superstep never starts unless the completed computation of all the nodes computation-communication. This is referred to as synchronization of barriers. The next step never starts until all the exchange of information in the participation nodes across the current superstep. The algorithm ends when there is no information exchange in the nodes. This state of halting can be done implicitly or explicitly stated. One can impose high number of supersteps and compel all nodes to stop if that limit exceeds the number of supersteps. Thus, the goal is achieved in one or more supersteps. Giraph framework writes each node's values in an output file at the end of the execution. The important keep in mind as, before starting the computation of the shortest path identification, to know from each node's prospective. To reach the shortest path in the each node and it also knows, and when it finds a shorter path updates its value. Tool Runner.run (arg1; arg2) [1] method invoked to start the process. The method begins the Giraph job, for example, from a single node, SHORTEST PATH operation. Image Figure 2 shows Giraph's execution template.

5. Calculating Degree of Influence Using Map-Reduce Framework

In this proposed method, the Degree of Influence (DI) illustrates the given social network with the importance of a node, which evaluated by the basic centrality measurement of the given graph. We are taking three cores degree, closeness and betweenness for DI calculation. The degree centrality depends on the neighbors node is called as the node of local density of the node. The remaining betweenness and closeness centralities are purely based on the shortest distance to all other nodes in the network, so we are combinedly calling these as global density. In this section, we have proposed the Map-Reduce mbased centralities calculations and finally these centralities are aggregated to find the DI is described.

5.1. Finding Betweenness Centrality Using Map Reduce

Betweenness centrality establishes among the two nodes over the shortest path and the total node acts as a bridge. In a social network, it is the quantifying control measurement for human that enabled communication between the any two human discussed by Linton Freeman [6]. By the concept, the vertices to chosen the random condition that have a high probability of smallest path amid two non-linear selected vertices with a consideration of higher betweenness.

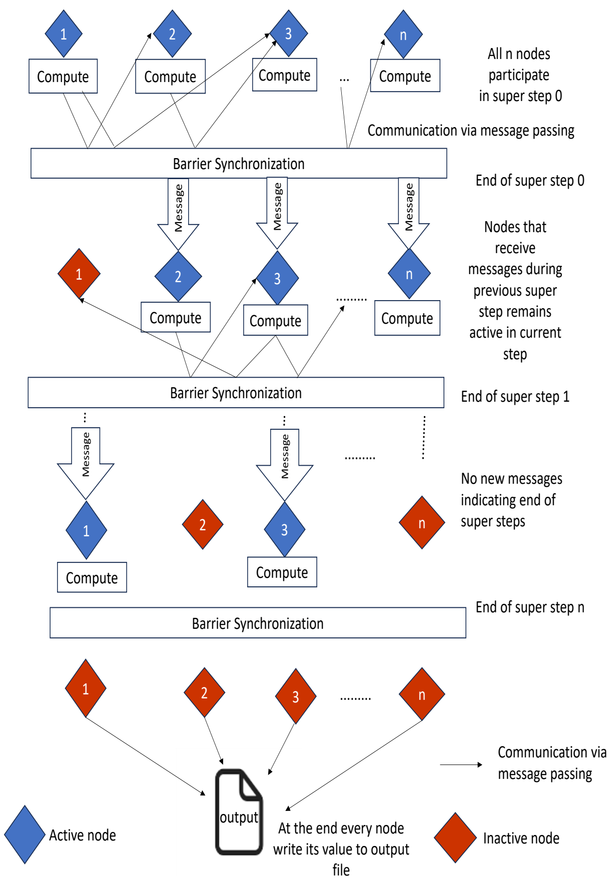

1
2
3
n
Compute
...
All n nodes participate in super step 0
Communication via message passing
Barrier Synchronization
End of super step 0
Message
Nodes that receive messages during previous super step remains active in current step
End of super step 1
No new messages indicating end of super steps
End of super step n
Communication via message passing
output
Active node
At the end every node write its value to output file
Inactive node

Figure 2. Giraph's execution model.

The betweenness of a vertex v in a graph G=(V, E) with V vertices is evaluated as below:

1. Calculate the shortest paths for each node pair (*s*, *t*).
2. Calculate the shortest path fractions for each node pair (*s*, *t*) allow through the vertex v.
3. Over all pairs of vertices (*s*, *t*) gets sum this fraction.

The betweenness centrality represented as below:

$$CB_{v=s\neq v\neq t\epsilon V} = \rho_{st}(v)/\rho_{st} \tag{1}$$

Where ρ_{st} is the overall shortest paths from‘s’ node to node *t* and $\rho_{st}(v)$ is the total paths to pass through *v*.

We have proposed the map-reduce based betweenness centrality in Algorithms (1) and (2). For the map task, the file contains the shortest path for all the possible node of pairs, which is generated by Giraph is given as input. The input is (key, value) pairs, here node_id is taken as key and path along with the cost is taken as value. Each node that appears in the path is emitted as an output key along with the value of 1. The output of the Map process is given as input for reduce job which is described in Algorithm (3). The reduce task takes each node and aggregates the values of every selected node. The aggregated value is standardized by dividing the total no of the shortest path in a given network graph. The final normalized value along with the node id is written in the output file.

Algorithm 1: Betweeness centrality Map Job

```
Input: file contains list of shortest path created by Giraph API
        key: <node_id>
        value:<path, path_weight>
  1.    begin
  2.    P ←value. Path  // extract the path P from value
  3.    for each $v_i \epsilon$ P do
  4.    emit ($v_i$, 1)
  5.    end-for
  6.    end
```

Algorithm 2: Betweeness centrality reduce Job

```
Input: key, value pairs of algorithm 1
key: <node_id>
value:<array of occurrence of paths for node_id>
Output: normalized betweenness value of each node.

  1.    begin
  2.    sum ← 0;
  3.    for i=0 to values. Length do
  4.    sum← sum+values[i]; // aggregating the occurrence
        of node vi in all the path P.
  5.    normalized_sum ← sum / |P| ; // betweeneess value is
        normalized between 0 to 1.
  6.    end-for
  7.    output(key,normalized_sum);
  8.    end
```

5.2. Finding Closeness Centrality Using Map Reduce

In connected graph network, the centrality of closeness [3] of a node is the shortest path with an average length in the graph between the neighbourhood nodes. The higher central of a node is, the closer it is to every other node, which can be stated in Equation (2).

$$C(v) = \frac{N-1}{\sum_u dist(v,u)} \tag{2}$$

Here, *dist* *(v,u)* describes the smallest path distance between vertices *v* and *u*, *N* denotes the total nodes in a provided network.

We have proposed the Map-Reduce based closeness centrality in Algorithms (3) and (4). For the Map task, the file contains the shortest path for all the possible pairs of nodes are given as input. Here the input is (key, value) pairs, node id is taken as key and path along with the cost is taken as value. For each path, the source node id and cost of the path are extracted and emitted as the key output, pairs of value. The output of the Map process is given as input for reduce job which is described in Algorithm (4). The reduce task takes each source node and aggregates the cost of all the paths of the selected node to all other nodes. The aggregated value gets normalized by taking the reciprocal of aggregated value and multiplying with N-1. The final normalized value along with the node id is written in the output file.

Algorithm 3: Closeness Centrality Map Job

```
Input: files contain list of shortest paths generated by Giraph API
key: <node_id>
value:<path, path_weight>
Output: node_id and array of path weights.
1.  begin
2.  For each node vi do
        //extract the path weight
3.  PW← value.pathWeight
        //extract source node id from the path
4.  SourceNode<-extracSourcenode(value.path)

5.  Emit(SourceNode,PW)
6.  end-for
7.  end
```

Algorithm 4: Closeness Centrality Reduce Job

```
Input: Output of Map job in Algorithm 3
key: <node_id>
value:<array of shortest path weights>
Output: normalized closeness value of each node.
    1.  begin
    2.  sum = 0;
    3.  for i = 0 to value. Length do
    // aggregate the total cost of each path from node vi to all
    other nodes.

    4.  sum = sum + value[i];
    5.  end-for
    // closeness value normalized between 0 to 1.
    6.  if sum≠ 0 then
    7.  sum =1/sum;
    8.  output. Write (key, sum);
    9.  end-if
    10. end
```

5.3. Finding Degree Centrality Using Map Reduce

Degree centrality [18] is a simple centrality measure that counts how many neighbors a node has. This centrality can be mathematically represented in (3), and we can normalize this value between 0 to 1 by dividing it with number of anticipated links (N-1) of each node. Here we are calling this centrality as local density of given node.

$$Cd(v) = \frac{\deg(v)}{n-1} \tag{3}$$

Deg (v) - represents the count of actual links of node v.n-1- no of anticipated links of each vertex.

We have proposed the Map-Reduce based degree centrality in Algorithms (5) and (6). The input contains node_id and its adjacency list of neighborhood nodes for all the nodes in the form of <node id, <adjacency list>> is given to Map task. Here the input is <key, value> pairs; node_id is taken as key and list of adjacency nodes is taken as value. For each node and it's, each neighbor node the map job emits the (key, value) pair as (node_id, 1). The output of the Map process is given as input for Reduce job which is described in Algorithm 6. The Reduce task takes each node_id and aggregates its neighborhood count. The aggregated value is standardized by dividing the total count of nodes in the given network graph. The final normalized value along with the node id is written in the output file.

Algorithm 5: Map Job for Degree centrality

```
Input: file contains node id and adjacency list of neighborhood nodes
key: <node_id>
value:<adjacency list>
Output: degree value of each node.

    1.  Begin
    2.  For each node_id
       // finding size of adjacency list
    3.  deg=sizeof(adjacency_list);
    4.  emit(node_id, deg)
    5.  end-for
    6.  end
```

Algorithm 6: Reduce Job for Degree centrality

```
Input: file contains node id and adjacency list of neighborhood nodes
key: <node_id>
value:<array of degree values>
Output: normalized degree centrality.

    1.  begin
// degree of each node initially set to 0
    2.  deg []={0}
    3.  for i=0 to values. length do
// degree of each node is calculated.
    4.  deg[i]=deg[i]+values[i]
    5.  end-for
    6.  for i=0 to values.length do
    // degree centrality is aggregated between 0 to 1.
    7.  emit(vi, deg[i]/n-1)
    8.  end-for
    9.  end
```

5.4. Calculating the Degree of Influence

Here we aggregated both the local density (degree centrality) and global density (closeness and betweenness centralities) of the network graph. Before aggregation, we are calculating the ratio of all the centralities for every node in a given network graph. The centrality ratios are defined in Equations (4), (5) and (6). Then the ratio of all the centralities taken for aggregation process and also, we have assigned the weightage for each centrality ratio in order to provide the importance for global density, because it is more common in large networks that having small value for closeness and betweenness centrality than degree centrality. The aggregation is defined in (7) and we called as Degree of Influence (DI) and it is normalized between 0 to 1. Here we have taken α, β=0.4 and γ=0.2.

$$\text{deg}_{\text{ratio}}(v_i) = \frac{\text{deg}(\text{v}_i)}{\sum_{\text{v}_i \epsilon \text{V}} \text{deg}(\text{v}_i)} \tag{4}$$

$$\text{between}_{\text{ratio}}(v_i) = \frac{\text{between}(\text{v}_i)}{\sum_{\text{v}_i \epsilon \text{V}} \text{between}(\text{v}_i)} \tag{5}$$

$$\text{closeness}_{\text{ratio}}(v_i) = \frac{\text{closeness}(\text{v}_i)}{\sum_{\text{v}_i \epsilon \text{V}} \text{closeness}(\text{v}_i)} \tag{6}$$

$$DI(v_i) = \alpha * \text{degratio}(\text{v}_i) + \beta * \text{closenessratio}(\text{v}_i) + \gamma * \text{betweenratio}(\text{v}_i) \tag{7}$$

- *DI* (v_i) represents Degree of influence of node *vi*.
- α, β, γ represents weightage parameters such that $\beta \geq \gamma > \lambda$ and α+ β+ γ=1

6. Identification of Leader Nodes

6.1. Relative Distance

The relative distance ρ_i of node v_i described as the minimum distance between the node v_i and any other nodes with the larger *DI* value, which represented as,

$$\rho_i = \min_{j:DI_j > DI_i} (d_{ij}) \tag{8}$$

Conventionally with the largest density for the node that take $\rho_i = max_j(d_{ij})$. By consider for the nodes that conclude the relative distance with the maximum of local density that attained higher and significantly than the adjacent nodes in the network. So, the anomalous condition of large value in the nodes enable both in the relative distance and the local density could identified as leader nodes.

6.2. Structural Centrality

Here we are calculating the structural centrality of every node. It can be evaluated by the relative distance and degree of effect of a node in the network. To characterized the structural centers by a higher *DI* value than the nearby nodes and by a large distance relatively from nodes with high *DI* value. The structural centrality sc_i of the node v_i is stated in Equation (9):

$$Sc_i = DI_i * \rho_i \tag{9}$$

The structural centrality measures the degree of influence of a node and the nodes effect with the higher density, which enables the metric as effectively ignores the situation that neighboring nodes with the maximum degree of influence are identified as structural centers. We are selecting some of the nodes which are having higher structural centrality as Leader nodes. The process of identifying the leader nodes centers are specified in the Algorithm (7). Here we are calling these leader nodes as structural centers and these leader nodes state the count of communities to be identified in the network.

Algorithm 7: Finding Leader Nodes

Input: Vertex set V, Adjacency Matrix, DI(Degree of influence) vector
Output: Leader nodes set S.

```
1.  Begin
2.  $S_c=0$
3.  $C_c=\{\phi\}$
4.  $S=\{\phi\}$
5.  for each $v_i \epsilon V$
            6.  $\mu = \min_{j:DIj>DIi} d_{ij}$
                $Sci = DI_i * \mu_i$
7.  end-for
8.  $S_{cut}=avg(S_c)$;
9.  for each $v_i \epsilon V$
10. if $Sc_i \geq S_{cut}$
11. $C_c=C_c \cup v_i$;
12. End-for
13. //$C_c$=sort Cc in descending order based on Sc.
14. while $C_c \neq \phi$
15. do
16. $C_{max}=max(C_c)$
17. $C_c=C_c-C_{max}$
18. $S=S \cup C_{max}$
19. For each $v_i \epsilon C_c$
20. If $d_{st}(C_{max},v_i) \leq \rho$
21. remove $v_i$ from $C_c$
22. end-for
23. end-while
24. End
```

7. Community Generation

In this section, the detection of community clusters based on the leader nodes derived from the previous section is discussed. First, the node vector for every node in the given network is generated. Then the community clusters are derived by using the proposed Parallel FCM Algorithm.

7.1. Node Vector Set Generation

We have generated the node vector for each node of given network by using the leader nodes identified by the pervious process. Here we have taken each leader node as the dimension (origin) of the node vector and the distance (shortest path) from the given node to leader node taken as dimensional value. The node vectors of some of the nodes for karathe network is depicted in Figure 3.

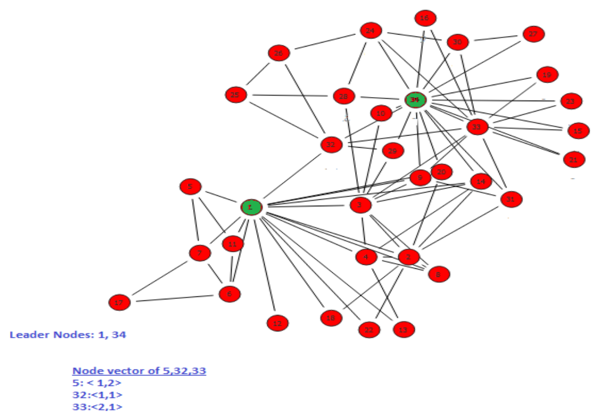


Figure 3. Seed nodes and node vector of karathe network.

7.2. Soft Community Clusters Generation

To generate the soft community structures, we have proposed the modified version of parallel FCM clustering algorithm which is shown in Algorithm 8. This algorithm generates the membership matrix of size n*k of all the possible communities. Here n represents count of nodes in the network, k denotes count of possible communities in the network. Here we have parallelized the clustering process using HADOOP Map-Reduce logic. The algorithm having two major steps:

1. Updating the cluster centers.
2. Updating membership matrix.

Here both the steps are parallelized separately using HADOOP Map-Reduce logic that is represented in Algorithms (9) to (12). Initially we have selected k nodes randomly as cluster centers from the given candidate nodes (nodes excluding the leader nodes) and perform the Fuzzy clustering process by means of two major steps iteratively.

Because we are assuming that count of communities is dependent on the leader nodes available in the given network, so we are setting value for k as |S|. In this algorithm we are assuming that the nodes which are located with the similar distances from the leader nodes are similar to each other. Thus, we have utilized the cosine similarity to measure the similarity between the pair of any nodes of given network graph which is described in Algorithms (11). Here we have taken fuzziness index m=2 and threshold parameter Δ as 0.0001 for small datasets and 0.00001 for larger data sets.

Algorithm 8: Map_Reduce_FCM

Input: node_id& its dimensional vector, threshold Δ
Output: Final membership matrix M

```
1.  Begin
2.  Choose k initial cluster centers
3.  Initialize Membership matrix $M^{(0)}$
4.  X=1, i=0
5.  while X> Δ
6.  Call cluster_center_MapReduce
7.  Call membership_MapReduce
8.  $X=M^{(i+1)}-M^{(i)}$
9.  end-while
10. return $M^{(i)}$
11. end
```

Algorithm 9: Cluster_Centre_Mapper

Input: List of<Key,Value> pairs [node_id:d1,d2,…,dn|m1,m2,..,mk]

*Output: List of <key,value> pairs[cluster_id$_i$:m_{ij}*X_j|m_{iJ}]*

```
1.  Begin
2.  for each (key,value) do
3.  Mi= Take Membership vector from value;
4.  Xi=Take dimensions vector from value;
5.  J=1
6.  for each $m_i \in Mi$
7.  $u=m_i^m*Xi$
8.  emit($j:u|m_i^m$)
9.  j=j+1
10. end-for
11. end-for
12. end
```

Algorithm 10: Cluster_Center_Reducer

*Input: List of key values[cluster_no$_i$: m_{ij}*X_j|m_{ij}]*

Output: List of key values [Cluster_no:Cluster_center]

```
1. Begin
2. for each cluster_No $C_i$
3. Sum1= aggregate $m_{ij}*X_j$belongs to $C_i$
4. Sum2=aggregate $m_{ij}$ belongs to $C_i$
5. $C_{new}$= Sum1/Sum2;
6. emit($C_i$:$C_{new}$)
7. end-for
8. End
```

Algorithm 11: Membership_Mapper

Input: list of key value pairs[node_id: d1,d2,…,dn] , and cluster centers C={C_1,C_2, …,C_k}

Output: List of key value pairs[node_id:m1,m2,….,mk]

```
1.  Begin
2.  for each node_id do
3.  for each $c_i \epsilon C$
4.  for each $c_j \epsilon C$
5.  $total\_sim += \left(\frac{cosine\_sim(c_j,vector\_data)}{cosine\_sim(c_i,vector\_data)}\right)^{\frac{2}{m-1}}$
6.  $m_i = \frac{1}{total\_sim}$
7.  emit(node_id, $m_i$)
8.  end-for
9.  end-for
10. end-for
11. end
```

Algorithm 12: Membership_Reducer

Input: list of key value pairs [node_id:membership_degree(m)]

Output: list of key value pairs[node_id:m1,m2,…..,mk]

```
1. Begin
2. for each node_id$_i \in$ keys
3. $M_i=\varphi$
4. for each $m \in$ values
5. $M_i$ =$M_i \cup m$
6. end-for
```

```
7.  Emit(node_id,$M_i$)
8.  end-for
9.  end
```

7.3. Assigning Leader Nodes to Communities

The communities identified in the previous step not containing the leader nodes in the network. The leader nodes also one of the members in the identified communities and they will be center of identified communities. Here we are assigning the leader nodes to the identified communities by using the cluster centroids generated during the parallel FCM algorithm. We are calculating the cosine similarity between leader nodes and all the centroids; the leader node is assigned to community cluster whose centroid having most similar with that leader node. This is depicted as follow.

$$Cluster\ (V_{leader}) = \max_{c_i \in C}\{Cosine(c_i, V_{leader})\} \qquad (10)$$

V_{leader} represents Leader node of ith cluster
C_i represents centroid of ith cluster

7.4. Hard Community Clusters Generation

So far, we have generated the soft clusters by using parallel FCM algorithm. This FCM algorithm generates the community membership matrix. Based on the matrix score the hard community clusters are generated by assigning the node to community having more membership scores. And we are generating the overlapping clusters by assigning more than one community labels for the node which having at most similar membership score for multiple top scorer communities.

8. Results and Discussions

8.1. Dataset Description

The proposed approach is tested on 11 real world data sets [13]. The detailed information regarding the datasets is described in Table 1. |V| and |E| are the total vertices and links; D_{avg} denotes the network's average degree; C-coef is the clustering coefficient; PL_{avg} denotes the average path length.

Table 1. Graph properties of the different data sets.

Networks	|V|	|E|	D_{avg}	C-coef	PL_{avg}
Football	115	613	10.661	0.403	2.508
Dolphin	62	159	5.129	0.303	3.357
Karathe	34	78	4.588	0.285	1.274
Jazz	198	2742	27.7	0.52	2.21
Poll blocks	1490	19025	25.537	0.172	3.39
Power	4941	6594	2.669	0.107	18.989
Yeast	2375	11693	9.847	0.153	4.14
Polbooks	105	441	8.40	0.4875	2.341
Email	1133	5451	9.62	0.166	3.65
Netscience	1589	2742	3.45	0.123	2.123
Word	112	425	7.59	0.1431	3.214

8.2. Results on Data Sets

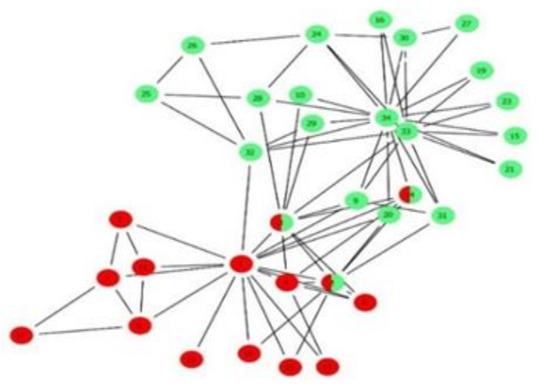
We have investigated our proposed method Leader Based Community Detection (LBCD) and some current SLPA, LFM, LBCD, LC, CORPA, and CPM methods on 10 real-world data sets here. Table 2 lists the results generated by these methods which include both the EQ [14] (Extended Modularity) values and detected the total communities. Extended modularity is a standard metric to evaluate the overlapping communities as purity. Some of these comparative approaches, namely SLPA, COPRA, and LFM, are indeterminate and difficult to obtain reliable performance, and others are prone to built-in parameters. Therefore, by adjusting the parameters for every network, to execute every algorithm 10 times to get the several results and took the largest EQ values as optimum results.

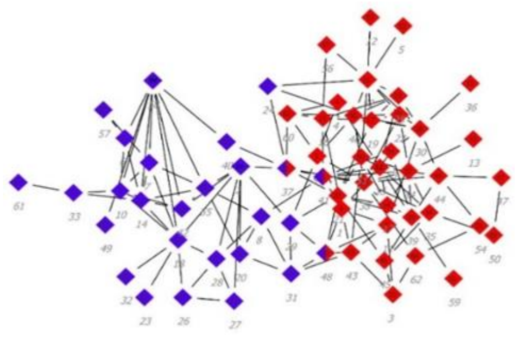
The following observations depicted in Table 2. In general, due to the strict concept of clique group, CPM struggles to deal with the complex networks. LC displays the community detection weakness, because the node similarity computation in the large networks that leads the multiple smaller communities. So, in larger networks, CPM and LC get minimum EQ scores and the datasets (power, poll blocks) compared to other approaches, but SLPA, COPRA, GCE, LFM and LBCD to attained maximum efficiency on the same datasets. Furthermore, some of the comparative algorithms are immune to unique network structure. For example, in some extremely sparsely organized networks such as Word, the methods COPRA and GCE detects only one single giant group structure which results EQ value near to zero. And another statement is that some algorithms incline to over detect the communities and the overlap.

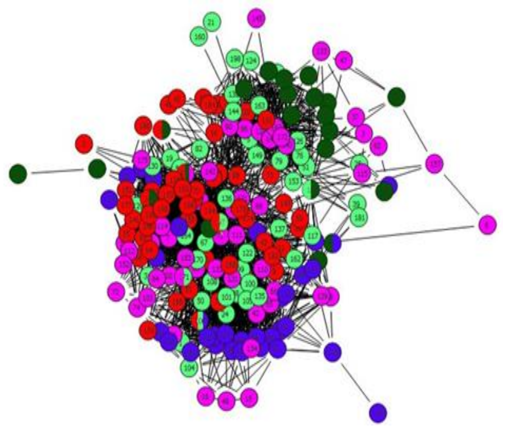
Table 2. Comparison of EQ values for different algorithms on real world data sets.

Data sets	**EQ (Extended Modularity score) / no. of communities**						
	CPM	**LC**	**SLPA**	**LFM**	**GCE**	**COPRA**	**LBCD**
Karathe	0.2708 / 3	0.2251 / 2	0.5192/2	0.4852/2	0.3474/2	0.2610/3	**0.5295 /2**
Dolphin	0.4195 / 4	0.216/10	0.4997/5	0.5183/5	0.4706/5	0.4640/4	**0.5486/2**
Football	0.4897 /15	0.1729/15	0.4712/11	0.4816/14	0.5907/12	0.3729/18	**0.6212/10**
Jazz	0.2396 /8	0.0381/6	**0.4944/6**	0.5198/4	0.3206/2	0.4560/4	0.4816/5
Polbooks	0.5008/5	0.0244/3	0.4959/3	0.4550/3	0.4865/3	**0.5115/3**	0.4996/3
Pollblocks	0.0106 /28	0.0118/27	0.4328/6	0.4425/8	0.2901/2	0.4290/9	**0.4815/4**
Power	0.1567/301	0.2149/322	0.559/661	**0.5649/167**	0.468/300	0.435/427	0.5105/290
Yeast	0.1153/17	0.6388/47	0.6268/53	0.6623/27	0.3544/40	0.2742/52	**0.6667/28**
Email	0.1163/4	0.0469/21	0.3634/16	0.4272/13	0.4273/35	0.00001/1	**0.5619/7**
Netscience	0.5745/169	0.8718/334	0.8683/325	0.8816/331	0.7250/126	0.6853/703	**0.8980/310**
Word	0.1003/4	0.0472/5	0.0910/3	0.0983/4	0.00001/1	0.00001/1	**0.1386/7**

For example, LC creates multiple overlapping nodes and the small community's networks such as Dolphin and poll blocks. Such over-detection for other algorithms, like COPRA in net science data set, CPM in poll blocks data set leads a poor performance due to the large number of small communities. Based on the above comparison results in terms of EQ, LBCD of better performance in relation to others. This conforms to the fact that our approach with complex network structures that demonstrated better performance for real-world networks. Specifically, in 8 of the 11 real-world networks LBCD acquires the highest EQ values and also nearly better results with remaining 3 data sets. Conversely, LFM, SLPA and COPRA perform better on one network of each, respectively. On almost all networks except the Poll blocks network, LBCD yields better performance than CPM and outperforms LC on all 11 data sets. It means that there are significant advantages to the local expansion strategy based on structural centers. Moreover, LBCD has a better performance compared to SLPA and COPRA algorithms based on label propagation. LBCD also exhibits highly stable results and compared to LFM on nearly except two other networks. The visualization of overlapped communities identified for karathe, Dolphins, Jazz and Email networks by our proposed method are shown in Figure 4. Here, some of the nodes are colored with more than one color indicates that belonging to more than one community (overlapping communities). The relationship between EQ and β parameter is depicted in Figure 5. EQ value is increased gradually along with β parameter and it reaches highest value between 0.4 to 0.45 in small datasets (Karathe, Dolphin, Football, Jazz and Word) as show in Figure 5 and it reaches the highest value during 0.35 to 0.4 in large datasets (power, yeast, Email, Netscience and Pollblocks) as show in Figure 5. The EQ is reached to least value whenever the β parameter reaches the maximum value (β =0.5), from the Equation (7). It is clear that the value of γ parameter reaches to 0 at β =0.5. From this it is clear that both global (Closeness and Betweenness centralities) and local measures (Degree centrality) are participated to derive the quality communities which having higher EQ value, but the participation of global measure is more than the local measures in community detection process. Since our method is HADOOP based Parallel community detection algorithm. Our proposed method is experimented on the HADOOP clusters. Here we have formed the Hadoop cluster of nodes and 8 nodes and experimented our methods. Intel octa Cores with i5-4440M 3.1GHz processor, 16GB memory and1TB Hard Disk. The execution time of our algorithms on each cluster for different data sets are specified in Table1. Here we are listed only listed the reasonable sized data sets to compare the execution time on different sized clusters. Here the execution time is measured in seconds. From the Table 3 it clears that execution time of our algorithm gradually decreases depending on the count of nodes in the HADOOP cluster increases. This scenario is visualized in Figure 6.


a) Communities detected in Karathe network.


b) Communities detected in Dolphin network.


c) Communities detected in jazz network.

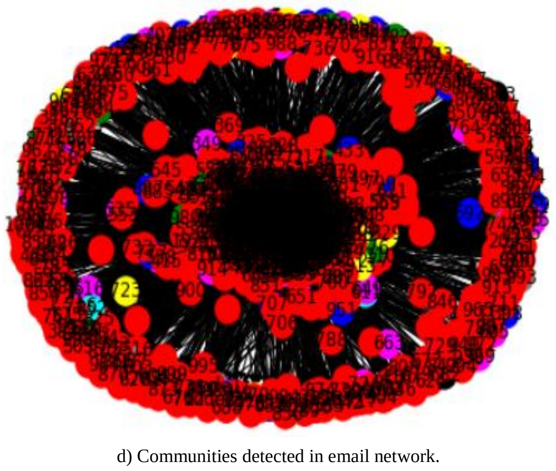
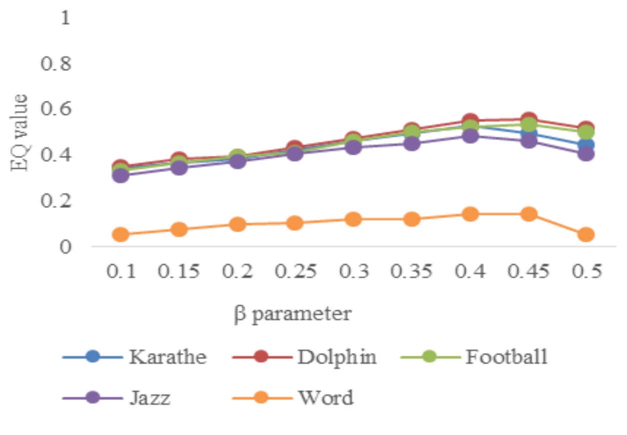
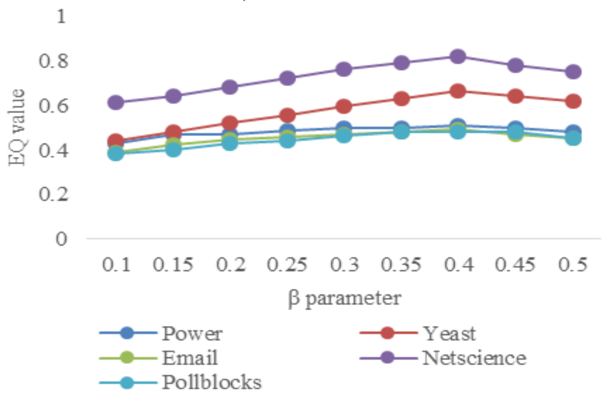

d) Communities detected in email network.

Figure 4. Communities generated in 4 social networks by proposed method.



a) Small networks.



b) Large networks.

Figure 5. EQ value against β parameter in small and large datasets.

Table 3. Execution time of four real world datasets.

Dataset	Time taken in seconds				
	Processor count(p)				
	|c|=1	|c|=2	|c|=4	|c|=8	|c|=16
Power	15,500	9850	4550	2560	1752
Yeast	8150	4456	2386	1326	786
NetScience	5256	3001	1656	932	532
PollBlock	4932	2621	1412	802	486

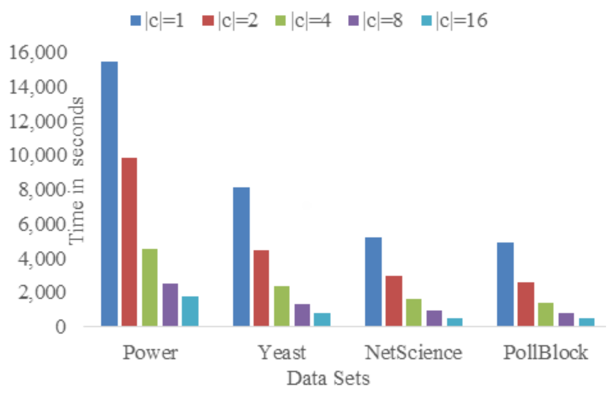


Figure 6. Execution time of four real world dataset.

9. Conclusions and Future Work

We have introduced an unsupervised machine learning based method of local expanding that depends on structural centers, to uncover the structures effectively in the overlapping community. A structural centrality introduced as a strategy location to compute the leader nodes in networks. Depends on these leader nodes we derived the Parallel MapReduce based FCM Clustering algorithm to find the overlapping communities from the given network. The proposed method (LBCD) defines the merits in experiments that can be concluded into two features. Initially, the count of community structures for a network calculated by the locating leader (seed) nodes. Secondly, to compared with other methods, the strategy of expansion leads around to the leader nodes that eliminates the seeds selection non-linearly and manually adjusting the parameters, which speeds up the convergence to optimal results and enables the most stable algorithm. Since we have experienced lot of disk access during map reduce jobs of proposed work, we are planned to upgrade this proposed work in spark environment in future. In future, we also planned to extend this leader-based technique to predict the future link and finding the influential users in social network graph.

References

[1] Apache Hadoop Manual. Available online: https://hadoop.apache.org/docs/r2.8.0/api/org/apache/hadoop/util/ToolRunner.html, Last Visited, 2023.

[2] Ahn Y., Bagrow J., and Lehmann S., "Link Communities Reveal Multiscale Complexity in Networks," *Nature*, vol. 466, no. 7307, pp. 761-764, 2010. https://doi.org/10.1038/nature09182

[3] Dangalchev C., "Residual Closeness of Generalized Thorn Graphs," *Fundamenta Informaticae*, vol. 162, no. 1, pp. 1-15, 2018. DOI: 10.3233/FI-2018-1710

[4] Filippone M., Camastra F., Masulli F., and Rovetta S., "A Survey of Kernel and Spectral Methods for Clustering," *Pattern Recognition*, vol. 41, no. 1, pp. 176-190, 2008. https://doi.org/10.1016/j.patcog.2007.05.018

[5] Fortunato S., "Community Detection in Graphs," *Physics Reports*, vol. 486, no. 3, pp. 75-174, 2010. https://doi.org/10.1016/j.physrep.2009.11.002

[6] Freeman L., Borgatti S., and White D., "Centrality in Valued Graphs: A Measure of Betweenness Based on Network Flow," *Social Networks*, vol. 13, no. 2, pp. 141-154, 1991. https://doi.org/10.1016/0378-8733(91)90017-N

[7] Gregory S., "Finding Overlapping Communities in Networks by Label Propagation," *New Journal of Physics*, vol. 12, no. 10, 2010. DOI:10.1088/1367-2630/12/10/103018

[8] Gupta S. and Singh D., "Seed Community Identification Framework for Community Detection over Social Media," *Arabian Journal of Science and Engineering*, vol. 48, pp. 1829-1843, 2023. https://doi.org/10.1007/s13369-022-07020-z

[9] Giraph API Manual. Available online: (https://giraph.apache.org/, Last Visited, 2023.

[10] Khang T., Vuong N., Tran M., and Fowler M., "Fuzzy C-Means Clustering Algorithm with Multiple Fuzzification Coefficients," *Algorithms*, vol. 13, no. 7, pp. 158, 2020. https://doi.org/10.3390/a13070158

[11] Lancichinetti A. and Fortunato S., "Limits of Modularity Maximization in Community Detection," *Physics Review E*, vol. 84, no. 6, pp. 1-8, 2011. https://doi.org/10.48550/arXiv.1107.1155

[12] Marwa M., Saber B., Laid K., and Okba K., "A Personalized Recommendation for Web API," *The International Arab Journal of Information Technology*, vol. 8, no. 3A, pp. 58-65, 2021. 2021 https://doi.org/10.34028/iajit/18/3A/7

[13] Mohamed M. and Latha K., "An Effective Community based Link Prediction Model for Improving Accuracy in Social Networks," *Journal of Intelligent and Fuzzy Systems*, vol. 42, no. 3, pp. 2695-2711, 2022. 10.3233/JIFS-211821

[14] Newman M. and Girvan M., "Finding and Evaluating Community Structure in Networks," *Physical Review E*, vol. 69, no. 2, 2004. https://doi.org/10.48550/arXiv.cond-mat/0308217

[15] Newman M., "Communities, Modules and Large-Scale Structure in Networks," *Nature Physics*, vol. 8, no. 1, pp. 25-31, 2012. https://doi.org/10.1038/nphys2162

[16] Palla G., Derenyi I., Farkas I., and Vicsek T., "Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society," *Nature*, vol. 435, no. 7043, pp. 814-818, 2005. https://doi.org/10.1038/nature03607

[17] Raghavan U., Albert R., and Kumara S., "Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks," *Physics Review E*, vol. 76, no. 3, 2007. https://doi.org/10.48550/arXiv.0709.2938

[18] Rusinowska A., Berghammer R., De Swart H., and Grabisch M., "Social Networks: Prestige, Centrality, and Influence," *in Proceedings of the International Conference on Relational and Algebraic Methods in Computer Science*, Rotterdam, pp. 22-39, 2011. https://doi.org/10.1007/978-3-642-21070-9_2

[19] Schaeffer S., "Graph Clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27-64, 2007. https://doi.org/10.1016/j.cosrev.2007.05.001

[20] Sun P., Gao L., and Yang Y., "Maximizing Modularity Intensity for Community Partition and Evolution," *Information Sciences*, vol. 236, pp. 83-92, 2013. https://doi.org/10.1016/j.ins.2013.02.032

[21] Xie J. and Szymanski B., "Towards Linear Time Overlapping Community Detection in Social Networks," *in Proceedings of the Pacific-Asia Conference on Knowledge Discovery Data Mining*, Kuala Lumpur, pp. 25-36, 2012. https://doi.org/10.48550/arXiv.1202.2465

Mohamed Iqbal completed his Ph.D. Degree at Anna University, India. He received B. Tech degree in Information Technology from Karpaga Vinayaga College of Engineering and Technology affiliated to Anna University, Chennai, Tamilnadu in 2009. M.E Degree in CSE from Anna University, BIT Campus, Tiruchirappalli, India in 2011. He is currently working as Assistant Professor (Senior Grade1) in the School of Computer Science and Engineering (SCOPE) at VIT-AP University, Andhra Pradesh, India. His research interests include Social Networking, Big Data, Machine Learning, Deep Learning and Natural Language Processing.

Kesavarao Latha currently working as Assistant Professor (Sr. Grade), Department of computer science and engineering in University College of Engineering, Anna University (B.I.T Campus), Trichirappalli-620024. She Published papers in various Scopus indexed journals, National and International conferences and Journals. She is specialization in Information Retrieval, Information Extraction, Data Mining and Cloud Computing.