Knowl Inf Syst
https://doi.org/10.1007/s10115-018-1156-3

REGULAR PAPER

DFuzzy: a deep learning-based fuzzy clustering model for large graphs

Vandana Bhatia[1] · **Rinkle Rani**[1]

Received: 1 May 2017 / Revised: 26 September 2017 / Accepted: 25 December 2017
© Springer-Verlag London Ltd., part of Springer Nature 2018

Abstract Graph clustering is successfully applied in various applications for finding similar patterns. Recently, deep learning- based autoencoder has been used efficiently for detecting disjoint clusters. However, in real-world graphs, vertices may belong to multiple clusters. Thus, it is obligatory to analyze the membership of vertices toward clusters. Furthermore, existing approaches are centralized and are inefficient in handling large graphs. In this paper, a deep learning-based model 'DFuzzy' is proposed for finding fuzzy clusters from large graphs in distributed environment. It performs clustering in three phases. In first phase, pre-training is performed by initializing the candidate cluster centers. Then, fine tuning is performed to learn the latent representations by mining the local information and capturing the structure using PageRank. Further, modularity is used to redefine clusters. In last phase, reconstruction error is minimized and final cluster centers are updated. Experiments are performed over real-life graph data, and the performance of DFuzzy is compared with four state-of-the-art clustering algorithms. Results show that DFuzzy scales up linearly to handle large graphs and produces better quality of clusters when compared to state-of-the-art clustering algorithms. It is also observed that deep structures can help in getting better graph representations and provide improved clustering performance.

Keywords Fuzzy clustering · PageRank · Deep learning · Large graphs · Pregel

1 Introduction

Deep learning has been graciously embraced by many big companies such as Facebook, Apple and Google that take benefits from the huge amount of digital data [7]. These big

✉ Vandana Bhatia
vbhatia91@gmail.com; vandana.bhatia@thapar.edu

Rinkle Rani
raggarwal@thapar.edu

1 Department of Computer Science and Engineering, Thapar University, Patiala, India

contributors of big data often demarcate the relationships among points in the form of graphs which let us mine more business value [34]. The recent research shows that deep learning-based autoencoders can be undertaken for the mapping of complex graph data into lower-dimensional spaces [29,42]. An autoencoder is a deep neural network comprising of multiple layers of sparse autoencoder used for performing unsupervised learning in iterative manner. The output of each layer is considered as the input to the successive layer. Graph algorithms are also iterative and are benefitted by the use of autoencoder [29,37]. Autoencoder can provide a better graph representation and has been used in various mining tasks such as data classification [8], pattern recognition and clustering [4,37,42].

Clustering is an efficient unsupervised learning technique to mine useful patterns from graphs [36]. Graph clustering is helpful in many real-life applications such as community detection [48,51], image segmentation [3] and protein–protein interaction [15]. In the literature, many graph clustering algorithms were proposed which have taken benefits from deep autoencoder layers [4]. A layer-wise pertaining scheme named ‘GraphEncoder’ has been proposed to map the input graph similarity matrix to the output graph embedding using spectral clustering and K-means [42]. The authors have shown that deep structures can help in getting better clustering results. Another approach named deep embedded clustering (DEC) has been proposed which iteratively optimizes a clustering objective based on Kullback–Leibler (KL) divergence with a self-training target distribution. DEC has been proved to gain benefit from the deep autoencoder [46]. Yang et al. [49] proposed a nonlinear model in deep neural networks to gain representation power for large complex networks for the task of community detection based on modularity.

However, all existing deep learning-based clustering algorithms find disjoint clusters only and are unable to scale up to large graphs [37,42]. But, in many real-world applications like social networks, biological networks, it is necessary to allow overlapping among the clusters because data points may inherently belong to more than one cluster [19]. Fuzzy C-mean (FCM) is the most popular fuzzy clustering algorithm introduced by Bezdek et al. [5] and has been used widely in many applications in the literature. But, FCM is sensitive to the selection of initial cluster centers. However, if the number of clusters is pre-defined in real-world large graphs, the actual clusters might break up or get merged. It may also result in the creation of clusters even when actually the data cannot be clustered [13,27]. Furthermore, many fuzzy clustering algorithms have been proposed in the literature [6,19,24,48]. But, most of them do not consider graph structure while clustering. However, for big data graphs, the structure of the graph plays an important role in determining the size of the clusters. PageRank algorithm can efficiently analyze the structure of graph by performing random walks iteratively and is highly scalable [22,23].

Motivated to investigate fuzzy graph clustering problem in large graphs by considering the structure of the graph, we proposed a distributed autoencoder-based layered model named DFuzzy. DFuzzy learns the structure of graph vertices by modeling sequence of random walks using PageRank algorithm. The proposed model first pre-trains the model by finding initial cluster centers on the basis of their importance in network. Further, it fine-tunes the network to learn the latent representation using personalized PageRank and by optimizing modularity. In precision enhancement phase, the cluster centers are updated using intra-cluster distance among vertices.

PageRank helps in reducing the number of parameters in autoencoders for graph clustering, without compromising with the quality of extracted information. However, PageRank-based clustering may ignore those vertices which do not fall in any random walk. Autoencoder attempts to reconstruct those lost vertices in the final output to minimize the information loss. It is important to note that autoencoder only reconstructs original graph by linear recon-

Table 1 Comparison of characteristics of clustering algorithms

Approach	Analyze network structure	Finds Fuzzy clusters	Compute fuzzy membership	Handle graphs	Scalable
MR-FCM [24]	×	✓	✓	×	✓
OLP [12]	✓	×	×	✓	×
BIGCLAM [47]	✓	✓	×	✓	✓
PFM [43]	×	✓	✓	✓	×
Semi-clustering [25]	×	×	×	✓	✓
GraphEncoder [42]	✓	×	×	✓	×
DLC [37]	×	×	×	✓	×
PGFC [6]	×	✓	✓	✓	✓
SIMPLE [45]	×	×	×	✓	✓
Proposed DFuzzy	✓	✓	✓	✓	✓

struction. Thus, DFuzzy exploits both the approaches so that the shortcomings of either approach can be compensated by the advantages of the other. The experimental results show the efficiency of the proposed approach in comparison with the existing approaches.

To the best of our knowledge, this is the first attempt to explore the use of deep learning-based autoencoder for fuzzy clustering of large graphs in distributed environment. Main contributions of the proposed DFuzzy are:

1. A scalable fuzzy clustering model is proposed that leverages the idea from stacked autoencoder pipelines to identify overlapping and non-overlapping clusters in real-life graphs efficiently.
2. Unlike traditional clustering algorithm, it is not mandatory to have prior knowledge about the number of clusters. DFuzzy finds the number of clusters by analyzing the structure of the graph and fine-tunes it by optimizing modularity.
3. DFuzzy is efficient in identifying both crisp and fuzzy areas of the clusters. DFuzzy saves time by calculating the fuzzy membership for only those vertices which lie in fuzzy partition.
4. DFuzzy handles large graphs of any size efficiently using parallel processing framework Pregel which runs on the top of Hadoop.

A comparative summary of the characteristics of DFuzzy with the relative existing approaches is given in Table 1. DFuzzy has all the desired characteristics, while the other algorithms lack in one or more. The clustering problem can also be solved efficiently using granular computing [21,32]. Solving graph problems with granular structures provide a good visual representation by incorporating various merits such as multi-view and multilevel representation [50]. Benefits of granular computing can also be utilized in solving problems related to uncertainty [9], fuzzy clustering [35], rough sets [39], soft computing [1,41], optimization problems [44], etc., efficiently.

The rest of the paper is structured as follows. Section 2 explains the background and related work. The proposed model ’DFuzzy’ is described in Sect. 3. The results of the performed experiments are shown in Sect. 4. Section 5 precisely presents conclusion with recommendation for further research.

2 Background and related work

2.1 PageRank

Many applications like social networks, collaboration networks. are based on the concept of distance between the two vertices of network for estimating the connection between them. Likewise, there is a profound concept of connected walks on networks suggesting that the relationship among two vertices can be predicted by computing the estimated number of steps taken to transit from one vertex to another following a random walk. A random walk is an ordinary stochastic process on graphs like Markov chains. Given a graph G and a starting vertex v_i, we select an adjacent vertex v_j at random and make a transition, after which we continue the random walk V' from the newly chosen v_j. The probability associated with jumping from node v_i to node v_j is $P_{ij} = \frac{a_{ij}}{d_i}$ where $d_i = \sum_{j=1}^{N} a_{ij}$ is the degree of v_i.

Following a random walk from v_i to v_j, some vertices have more incident links and thus visited more frequently than others. For identifying such vertices, Brin and Page [30] introduced a web search algorithm ‘PageRank’ to find the most referred web pages. It can be very well fitted in various forms of graph also and is pretty much effectual for capturing relationships among vertices of graphs. PageRank uses a real value ‘α’ where $\alpha \in [0, 1)$; instead of counting steps, the random walk is taking. In a graph G with vertices $1, \ldots, N$, let the adjacency matrix $M \in G^{NxN}$ and the diagonal matrix having degrees on the diagonal be $D \in G^{NxN}$. Let $P = GD^{-1}$ be the random walk transition matrix. The PageRank vector p is equal to:

$$p = (1 - \alpha)(I_n - \alpha P)^{-1} s \tag{1}$$

where I_n is the identity matrix and $s \in G^N$ is a vector referred as teleportation vector. An effective method for performing random walk with restart is personalized PageRank [30]. It is demarcated as a stationary distribution of the random walk that takes an outgoing edge with the probability α and jumps back to the center node with probability $(1-\alpha)$. The personalized PageRank transition matrix P_s is given by following equation:

$$P_s = (1 - \alpha)c + \alpha M \tag{2}$$

Personalized PageRank has been used in many applications to find communities. Bahamani et al. [2] designed a Map-Reduce framework for efficient processing of personalized PageRank. DFuzzy uses personalized PageRank, to learn the structure of the graph. It finds the fuzzy clusters by analyzing the connections between the vertices. The return probability in personalized PageRank may differ from cluster to cluster depending upon the size of the cluster. Let π_s be the stationary distribution of P_s. We allocate vertices to clusters using cluster $(v_i) = argmax_{s \in K} \pi_s(v)$.

2.2 Fuzzy C-means clustering

Fuzzy C-mean (FCM) algorithm was initially developed by Dunn in 1973 and further upgraded by Bezdek [5] in 1981.

The fuzzy C-mean is an overlapping data clustering algorithm which attempts to partition a finite collection of vertices $V = (v_1, v_2, v_3, \ldots, v_N)$ into fuzzy clusters $C = (c_1, c_2, c_3, \ldots, c_k)$ by assigning some degree specified by a membership value with in the interval [0, 1]. FCM is based on minimizing the squared error objective function J in accordance with the distance and the membership value.

$$J = \sum_{i=1}^{N} \sum_{j=1}^{C} (U_{ij})^m ||v_i - c_j||^2 \tag{3}$$

where m is the fuzziness index of value greater than 1 and U_{ij} is the membership of v_i in the jth cluster. Fuzzy clusters are obtained through optimization of objective function in Eq. (6) iteratively by updating membership matrix U_{ij} and cluster center c_j. Convergence is reached when the edge cut is less than the threshold ϵ.

First initialize the membership matrix $U_{ij} = \mu_{ij}^k$. Then, calculate the fuzzy membership for all the vertices using equation:

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{c} \{\frac{x_i - c_j}{x_i - c_k}\}^{\frac{2}{m-1}}} \tag{4}$$

After that, compute the fuzzy clusters c_j for all the clusters using:

$$c_j = \frac{\sum_{i=1}^{n} (\mu_{ij}^k)^m * v_i}{\sum_{i=1}^{n} (\mu_{ij}^k)^m} \tag{5}$$

repeat the above two steps until the value of J in Eq. (3) is minimized or if $\forall i; j$: $max_{ij}\{\mu_{ij}^{k+1} - \mu_{ij}^k\} \leq \epsilon$.

2.3 Modularity optimization

Modularity of any Graph G is a measure of efficiency of the clusters formed within graph [14]. A graph with large modularity has more intra-cluster edges and less inter-cluster edges among various clusters. For calculating the modularity of fuzzy clusters, Nepusz [27] introduced a measure for cluster quality. Membership value μ_{ki} can be considered as the probability that vertex v_i is a member of cluster k. The probability of the case that vertex v and u belong to the same cluster is the dot product of their membership values, resulting in a similarity measure s_{uv}. For fuzzy clusters, the modularity is given by:

$$Q_{fuzzy} = \frac{1}{2m_e} \sum_{(u,v)\in V^2} (a_{uv} - \frac{deg(u) * deg(v)}{2m_e}) s_{u,v}. \tag{6}$$

where a_{uv} is the element in adjacency matrix of graph G and m_e is the total number of edges, $s_{uv} = \sum_{k=1}^{C} \mu_{kv}\mu_{ku}$.

2.4 Deep learning for graph clustering

Deep learning is not something new. It originated way back in 1960s [16]. But in recent times, the constantly reducing cost of computing resulted in a sudden surge for deep learning. Also due to rise in data volume and improvements in technology, deep learning has become a captivating method toward the growing business glitches [7,26].

For performing unsupervised learning in deep architectures, autoencoder plays a vital role in transfer learning and various other tasks. Autoencoder in deep neural networks is responsible for extraction and embedding of features in an unsupervised manner. Figure 1 shows the high-level abstraction of an autoencoder based on multiple hidden layers.

Autoencoder is trained to reconstruct their own input to find a more informative version generally using clustering. Deng et al. [11] designed deep stacking network (DSN), a

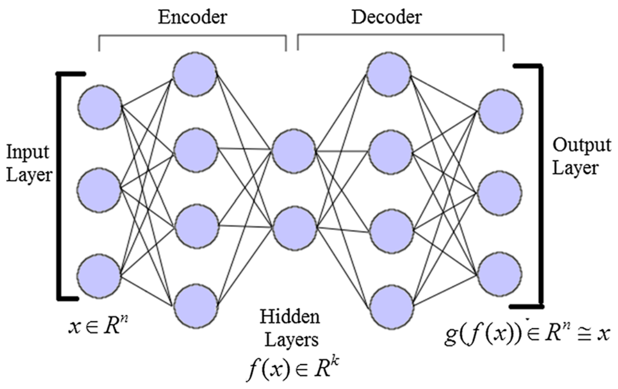

Fig. 1 Autoencoder in deep neural network

parallelized deep architecture in batch mode which involves numerous specialized neural networks (modules) with only one hidden layer. Instead of input units, the raw data vector was concentrated with the output layer in lower modules. Hutchinson et al. [17] proposed tensor deep stacking network (T-DSN), a deep architecture consisting multiple stacked blocks with two hidden layers. A GPU-based framework [33] is designed for hugely parallelizing unsupervised learning models including deep belief networks.

Some deep methods were proposed for clustering recently [37,40,42]. Deep learning-based autoencoder is used for finding clusters by optimizing modularity [49]. Due to rapid growth of data, deep learning has also been used in certain distributed platforms. Jhang and Chen [52] presented a distributed deep belief network based on Map-Reduce framework by exploiting data-level parallelism. Several levels of distributed restricted Boltzmann machines were stacked, and then, distributed back-propagation algorithm was used for the fine tuning.

2.5 Framework for learning from large graphs

Although deep learning has been adopted by many applications and provided the efficient outcomes, its training is still a trivial task. It is tremendously difficult to parallelize the iterative computations of deep learning algorithms. But, in recent years, with the unprecedented growth of digital data in various domains, the deep models can be trained well. To handle such large amount of data and to perform the computations on it, there is a surge for designing parallel, efficient and scalable algorithms for training the deep learning models. Most of the existing work in the area of distributed computing considers Map-Reduce framework for performing computations [10,18]. However, Map-Reduce framework is not efficient in performing computations on graphs.

For handling large graphs, Pregel [25] performs computations efficiently. It is a distributed framework which performs in-memory vertex-centric computations using bulk synchronous parallel (BSP) model by dividing the processing into supersteps and substeps. It exploits fine-grained parallelism at node level and enables fast random access and the reuse of intermediate results for iterative graph algorithms.

Pregel works with two basic operations, communication via sending messages and barrier synchronization as shown in Fig. 2. The vertices of a large graph are distributed among the worker nodes for performing computations independently. Communication among the workers occurs by passing messages which are generally small in comparison with data and

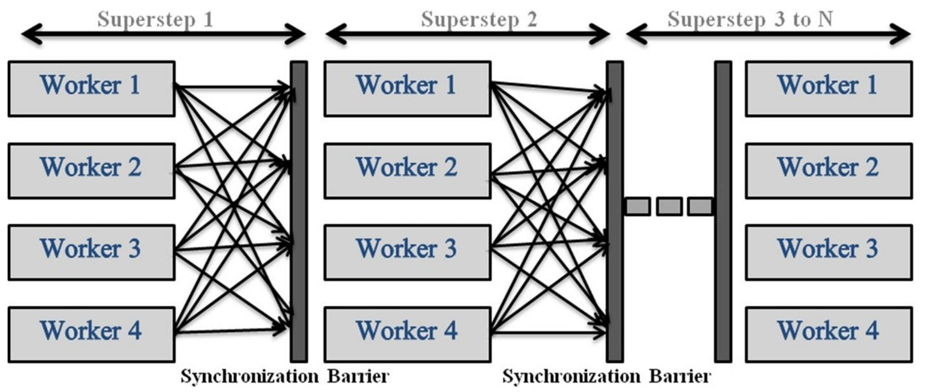

Fig. 2 Working model of Giraph

thus more efficient for transmission. Messages from the previous superstep are available in next superstep. The algorithm terminates when messages transfer does not occur during iteration or until each worker node votes to halt.

In this paper, iterative BSP-based graph processing framework Giraph is used. It works as a sequence of supersteps and performs iterative calculations on the top of the Hadoop cluster. It avoids costly disk and network operations by using out of core in-memory execution. Only intermediate values in the form of messages are sent across the network. In DFuzzy, Hadoop distributed file system (HDFS) is used for distributed storage and calculation. The disk access takes place while fetching input from HDFS, storing output back in HDFS and for storing checkpoints only. Each cycle of an iterative Giraph calculation on Hadoop runs a full Map-Reduce job with only mappers.

Pregel is appropriate for transfer learning as it is performing computations in multiple iterations by using output of one superstep as the input for next superstep. The vertices of the graph can be viewed as neurons, and the message communication between vertices can be interpreted as synapsis between neurons.

3 Proposed DFuzzy model

The proposed DFuzzy model is a parallel and scalable fuzzy clustering model especially designed for large graphs using BSP- based Pregel framework. Deep neural network with sparse autoencoder is the building blocks of the DFuzzy. The model works in three phases. In the first phase, for a given graph $G = \{V, E\}$ the cluster heads are selected based on their importance in the graph. Further, the clusters are formed using personalized PageRank starting from each of the selected cluster heads. All operations are performed using stacked autoencoder by providing greedy layer-wise training. The use of autoencoder for graph clustering is shown in Fig. 3.

Consider the autoencoder module for DFuzzy be D_A comprising of cluster head selection, detection of fuzzy and crisp clusters, optimization of the formed clusters and assignment of vertices to clusters. Let p be the desired number of clusters for each layer l_i of D_A and k be the final number of clusters. Let the layers of D_A be $l_1, l_2, \ldots, l_x$. For each layer, input X_l is computed using:

$$X_l = D_{A,l}(X_{l-1} - m_l) \tag{7}$$

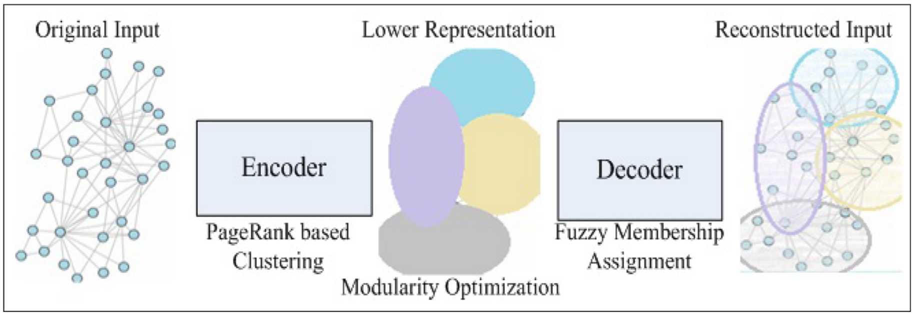

Fig. 3 Graph clustering using autoencoder

Algorithm 1: DFuzzy with Deep learning based Autoencoders

```
Data: Graph $G$ with edges($v_i$, $v_j$)
Result: $k$ clusters
1  Initialize autoencoder module $D_A$
2  Initialize the vertices of graph $G$
3  for each $v_i \in G$ do
4      Compute the PageRank value $p_i$          // using Algorithm (2)
5      Update $X_l$= $p_i$                       // Input to autoencoder
6  end
7  Select $m$ cluster centers
8  for each $m_i \in G$ do
9      //Train autoencoder using Personalized PageRank
10     Obtain Lower Representations            // using Algorithm (3)
11     Optimize clusters using Modularity      // Latent space of trained autoencoder
12     Update Final Cluster $C[k]$
13     Minimize the reconstruction error $L(i)$   // using equation (9)
14 end
15 Precision Enhancement                        // Using Algorithm (5)
```

where $m_l = \frac{\text{Number of Connections}}{2}$. Let x_i be the input vector of input layer, h_i be the hidden vector of hidden layer and f_i be the output vector of output layer. Consider H_o and F_o as the activations of the hidden and output layer, respectively. The hidden vector h_i and output vector f_i are computed using:

$$h_i = H_o(Wx_i + a) \quad \text{and} \quad f_i = F_o(Xh_i + b) \tag{8}$$

The set of parameters $\{\theta_1, \theta_2\}= \{W, a, X, b\}$ is to be learned in the layers. The goal is to minimize the difference between the reconstructed f_i and input x_i passing through the hidden embedding h_i. The reconstruction error is computed as:

$$L(i) = min_\theta = \sum_{i=1}^{N} ||f_i - x_i||_2 + \beta(\rho) \tag{9}$$

where β controls the weight of the sparse penalty term and ρ is average activation of hidden unit H_o. The working of DFuzzy using autoencoders is summarized in Algorithm 1.

The whole training process in DFuzzy is iterative, and during each iteration, two tasks are performed, vertex value update and merge update. During the vertex value update task, each worker first updates the value of vertices in the local graph according to the received

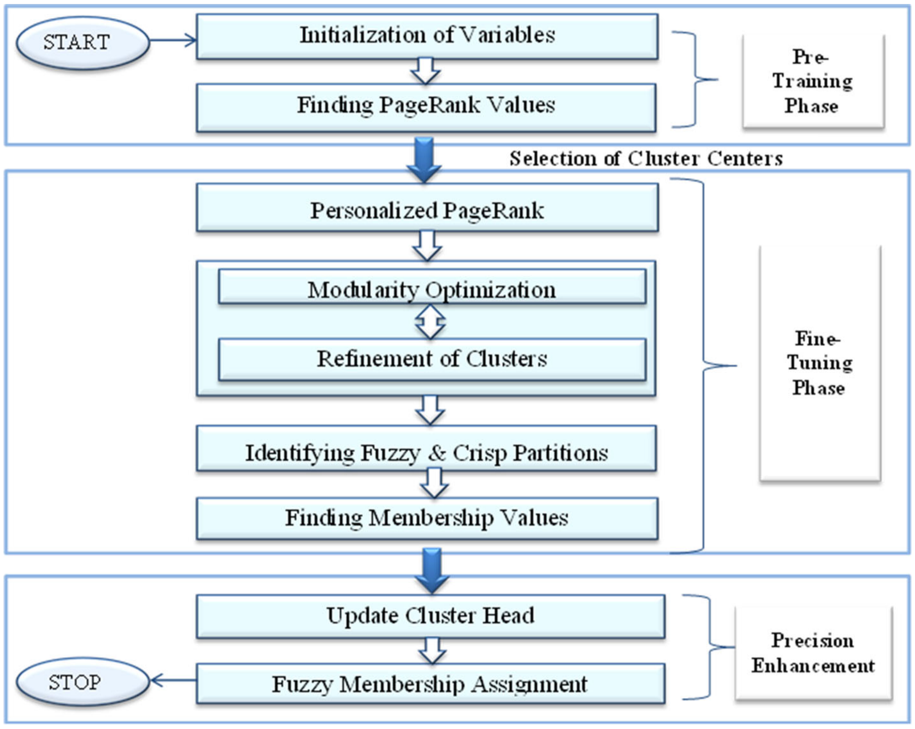

Fig. 4 Block diagram of DFuzzy model

messages from the master node. Then, the local computation is performed to alter the vertex values and labels in the assigned data partitions and finally send the updated vertex values to the master node.

In the merge update task, the master node updates the vertex's values and labels according to the messages received from the worker nodes. Then, it distributes the updated data to all the worker nodes. Both tasks are repeated alternatively until the specified numbers of iterations are performed or any of the termination criteria is met. During each iteration, computations are performed on each worker node, and then, the final aggregation is done on master node. The updated model is coordinated with each worker node. In Pregel, the major cost is of communication among the nodes. Therefore, rather than minimizing the inter-cluster edges, the communication cost needed to be minimized. Hence, we combine individual messages from each worker node into a single message, which incurs through the most costly network communication.

The block diagram constituting the steps of involved in three phases of proposed DFuzzy is shown in Fig. 4. The functioning of three phases of DFuzzy in parallel graph distributed environment Pregel is given below:

3.1 Unsupervised pre-training

The pre-training is performed before the fine-tuning phase to optimize the number of iterations used in fuzzy clustering. To initialize the traditional FCM algorithm, cluster centers are selected randomly, and then, further optimization is performed which takes ample time in

Algorithm 2: Pre-Training: Identifying Important Vertices

```
Data: Graph $G$ with edges($v_i, v_j$)
Result: Vertices PageRank Value in sorted form
Function Compute(Msg, superstep)
if $superstep = 0$ then
    $V_i = 1/N$;
end
else
    while $superstep \leq 25$ do
        superstep++ ;
        while $v \leftarrow Msg_i$ do
            $Sum = Sum + Msg_i.V_i()$;
            $V_i = 0.15 * V_i + 0.85 * Sum$ ;
        end
        sendMsg(value)
    end
end
while $superstep < 30$ do
    while $v \leftarrow Msg_i$ do
        // Sort vertices according to PageRank value. sort( $v.list, Msg_i$);
    end
    Send Message to all the neighbors(v.list);
end
```

performing computations. To overcome this time overhead, PageRank algorithm is used in DFuzzy to determine the importance of each vertex in the graph. The detailed algorithm for pre-training phase is shown in Algorithm 2.

3.1.1 Finding PageRank

The vertices of the graph are initialized with a value $1/N$ at superstep 0. Then, the vertices try to learn their importance in the graph by analyzing the topology of the graph. Here, basic autoencoders are used by setting the target values V' to be equivalent to the input values V. The autoencoder tries to learn PageRank values of the vertices using Eq. (4). Each vertex passes its tentative PageRank value divided by the number of incident edges to each outgoing edge.

The learning process starts from superstep 1 where each vertex adds the values arriving via messages into Sum and forward its own tentative PageRank value to adjacent vertices. The PageRank value is computed using $0.15/N + 0.85 * Sum$. After reaching the maximum limit of supersteps or when the PageRank values stop changing, all vertices stop passing messages and vote to halt. A new representation is obtained consisting of normalized PageRank matrix.

The vertex with higher PageRank value has more number of edges incident on it, so it is more important in comparison with other vertices of the network. Thus, vertices are sorted in the decreasing order of their PageRank value in the final list and top 10% vertices from this sorted list of vertex values are selected as candidate cluster heads. These 10% vertices have very high influence in the network. It is taken into consideration that if two adjacent vertices are in the list, then the vertex with high influence is considered and the other is simply discarded from the candidate cluster head list.

Algorithm 3: Fine Tuning: Fuzzy Clustering in DFuzzy

```
Data: Graph $G$ and selected candidate cluster head set C[$p$]
Result: Fuzzy clusters
while $Superstep \leq 60$ do
    for each $p_i \in C[p]$ do
        $m = 0$
        $c(m_i)$ = PersonalizedPageRank($C_m$)    // using Algorithm (4)
    end
    for each $c(m_i)$ do
        while $Modularity(c(p_i)) \leq \theta$ do
            Allocate vertices of $c(p_i)$ to neighboring clusters $c(p_j)$
            Calculate $\Delta Q_{fuzzy}$ of $c(p_j)$
        end
    end
    Aggregate($vertices, ClusterHead$);
    $C[k]$= Remaining clusters
    while $v \leftarrow Msg_i$ do
        for each $c(k_i) \in C[k]$ do
            for each $v_i \in c(k_i)$ do
                membership($v_i$) $\rightarrow c[k_i]$    // using equation (4)
            end
            Aggregate( fuzzy-vertices, Membership) ;
        end
    end
end
```

3.2 Fine tuning

The pre-training step initializes the value of vertices and then updates them using the PageRank algorithm to reach closer toward the desired solution. Being an unsupervised learning model for large graphs having millions of vertices and edges, DFuzzy skips the costly back-propagation function during all the stages of fine-tuning phase and simply passes the output of one layer to the next layer. For processing large-scale data, using back-propagation is a costly affair [38]. Thus in DFuzzy, back-propagation function of autoencoder is only used while performing modularity optimization. For the rest of the stages, the value of vertices is updated during iterations to minimize the reconstruction error.

The vertices with high PageRank values have been selected as the candidate cluster heads C[p] in pre-training phase. Here, p is the desired number of clusters at corresponding layer. The other vertices are added to clusters by starting personalizing PageRank from each candidate cluster heads p_i. Vertices that lie in the path of random walk starting from the candidate cluster head are aggregated to form cluster. The formed clusters are the latent representations of the input which are further fine-tuned by maximizing modularity and thus minimize the error function. The clusters with higher modularity than threshold θ are then stored in final cluster set $C[k]$. Further, the fuzzy and crisp clusters are identified to compute the membership of fuzzy vertices toward respective clusters. The steps of fine-tuning phase of DFuzzy are summarized in Algorithm (3). The detail of each step in fine-tuning phase is given below:

3.2.1 Fuzzy personalized PageRank

Once a list of cluster heads is ready, we can start the process of node assignment. The node assignment for the expansion of the cluster is done by using personalized PageRank algorithm

Algorithm 4: Adding Vertices to Clusters (PersonalizedPageRank(C_m))

```
Data: Selected CandidateClusterHeads
Result: Formed Clusters
Function Compute(Msg,superstep)
while $superstep \leq 40$ do
    superstep++;
    while $v \leftarrow Msg_i$ do
        if $v_i == c_i$ then
            $Sum_i = 0$
            $value = 0.15 * v_i + 0.85 * Sum_i$ ;
        end
        else
            value = $0.85 v_i * Sum_i$ ;
        end
        $v.label = label_i$
    end
    sendMsg(value)
end
while $superstep \geq 40$ do
    while $v \leftarrow Msg_i$ do
        voteToHalt() ;
        sendMsg( Clusters);
    end
end
```

starting from the selected centers. The algorithm starts lazy random walk from cluster head c_i and visits the neighbor vertex v_i randomly with probability $\frac{1}{2}(1-\alpha)$ or go back to c_i with probability α. The detailed steps are explained in algorithm 4.

A cluster includes the center having high PageRank value with several other vertices. The prominent center spread the influence to its neighbor vertices in multiple iterations. The closer the vertex is to the center, more influence it has. The cluster center c_i transmits its impact with label information to its adjacent neighbors. The neighbors pass their impacts to their adjacent neighbors in further iterations. During each iteration, vertices classify the received messages according to the label. In further iterations, each vertex creates new messages and sends them to its neighbors. The process continues until the specified numbers of supersteps are executed. Finally, sum of messages having the same label is calculated by adding the number of edges in a random walk $(C_a, v_{a1}, v_{a2}, \ldots, v_{ap})$ using $Sum_a = \sum_{i=1}^{msg_a} Edge_a$, where msg_a is the number of messages with label a.

The clusters formed in this stage form the latent representations and are used as input for the next stage. The dimensions of hidden layer are lower than that of input layer in autoencoders. All the edges and vertices of original graph are not necessarily part of the clusters. The relationships among vertices are thus captured to cluster the vertices.

3.2.2 Modularity optimization

The formed clusters are assessed by the fuzzyfication of the modularity function Q_{fuzzy} from Eq. (6). The cluster with low modularity tends to have more edges outside the cluster in comparison with the edges within the cluster. The conflict of the membership of vertices in clusters with very low modularity toward the neighbor cluster is solved by allocating such vertices to that neighbor cluster in which they have an outgoing edge. It is accomplished in an iterative manner by arranging the generated clusters according to their Fuzzy modularity value Q_{fuzzy}.

The master iterates through the clusters and adds the clusters with high fuzzy modularity to the final list of clusters. When a cluster with low fuzzy modularity appears to master, it instructs worker nodes to allocate the vertices of such clusters to other neighbor cluster in which they have an outward edge.

Here, the parameters $\{W, a\}$ of autoencoder can be learned from the change in modularity value ΔQ_{fuzzy} of the clusters to which vertices are added. The back-propagation feature of autoencoders is exploited at this stage by optimizing the overall modularity value. The change in modularity is given as:

$$\Delta Q_{fuzzy} = \frac{1}{m}\Bigg[\Bigg(\sum_{j\in p} M_{vj} - degree(v)\frac{\sum_{j\in p} degree(j)}{2m}\Bigg) - \Bigg(\sum_{j\in q} M_{vj} - degree(v)\frac{\sum_{j\in q} degree(j)}{2m}\Bigg) \tag{10}$$

Worker re-assigns all such vertices and reports the master with the new updated list of clusters. Clusters with lower modularity are thus vanished and removed from the cluster list.

3.2.3 Identifying fuzzy and crisp partitions

From the clusters formed in last autoencoder layer l_i, a vertex v_i can be a part of more than one random walks initiated from different cluster centers as shown in Fig. 5. Vertices with label v_8 and v_9 from random walk $(C_1, v_1, v_2, \ldots, v_9)$ and vertices with label u_7 and u_8 from random walk $(C_2, u_1, u_2, \ldots, u_9)$ are same. Thus, two different clusters with cluster heads C_1 and C_2 overlap and form fuzzy clusters with two fuzzy vertices v_9/u_8 and v_8/u_7, respectively.

The vertices found in more than one cluster are identified as fuzzy and which lies only in one cluster are identified as crisp. In next stage, the membership value is assigned to these identified fuzzy vertices.

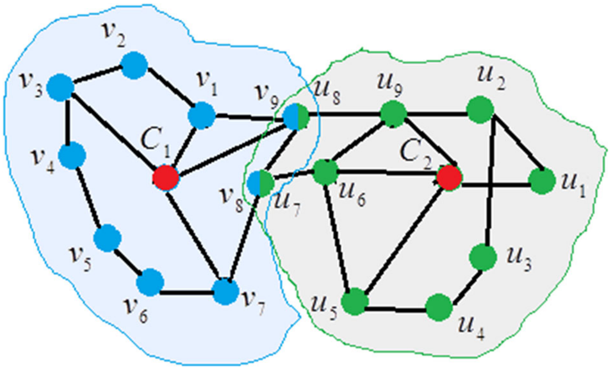

Fig. 5 Fuzzy personalized PageRank clustering

Algorithm 5: Precision Enhancement: Final Cluster centers

```
Data: Formed Clusters $C_k$
Result: Fuzzy clusters with cluster heads
1 while $C_k \neq 0$ do
2     Foreach $v_i \in C_k$ minDistance=min(message)
3     sendMessage($C_k$,minDistance)
4     $newC_k = C_j$        // from equation (5)
5 end
6 Aggregate($C_j$, $Vertices$)
7 Aggregate( Vertices, Membership) ;
```

3.2.4 Fuzzy membership

Fuzzy membership of a fuzzy vertex v_f is the membership value μ_{fk} toward all the entailing clusters. In DFuzzy, workers compute the membership value of only those vertices in corresponding autoencoder layer X_{l_x} which lies in more than one cluster using Eq. (4). The membership value 1 is assigned to tho vertices lying in only one cluster. Thus, a node assignment in DFuzzy can be crisp as well as fuzzy. Master node identifies such vertices while performing merge update and instructs workers to calculate the membership of fuzzy vertices toward the connected clusters. During each iteration, workers calculate the membership value μ_{fk} of fuzzy vertices v_f toward the connected clusters c_k using Eq. (7). Manhattan distance is used to compute the distance because it is not a squared function and is less sensitive to noise. It is given as:

$$d(v_i, v_j) = \sum_{k=1}^{C} |v_{ik} - v_{jk}| \tag{11}$$

3.3 Precision enhancement

In this phase, the cluster head of the formed cluster is updated based on the distance from the rest of the nodes in the cluster to obtain the final results in the last layer. The final output f_i is obtained be aggregating the vertices with the membership values toward the respective clusters. The working of this phase is summarized in algorithm 5.

3.3.1 Updation of cluster head

The selected candidate cluster heads have high importance in network, and their impacts do not change if we walk to any other neighbor vertex in the cluster. Up to now, the distance of vertex v_i from center c_k is not considered. This may not lead to better results, as some vertices in the cluster could be far away from the cluster head. The accurate membership of these vertices toward the corresponding cluster cannot be calculated. To deal with aforementioned problems, the distance should be taken into consideration. Farthest the vertex v_i from the center c_k, lesser is the influence of c_k on v_i.

The master node has the aggregated list of all the clusters with the assigned vertices. In the next superstep, worker nodes receive a list from the master and are instructed to calculate the distance of each vertex v_i from v_j in the cluster c_v. Vertices within the same cluster communicate by passing messages telling others about their distance from the cluster head. Worker nodes use combiners to minimize the number of messages. So, each vertex needs to

Table 2 Dataset description

	Number of vertices	Number of edges
Data source		
Facebook	4039	88,234
Twitter	81,306	1,768,149
GPlus	107,614	13,673,453
LiveJournal1	4,847,571	68,993,773
Ground truth data		
DBLP	317,080	1,049,866
Amazon	334,863	925,872
YouTube	1,134,890	2,987,624
LiveJournal2	3,997,962	34,681,189

evaluate only some part of the received message. In further iterations, worker nodes calculate the new cluster heads for all the clusters using Eq. (5).

3.3.2 Membership assignment

After reaching the convergence, all the vertices vote to halt. The workers then pass the information to the master node for merge update. The master aggregates the list of new cluster centers with the membership values of assigned vertices correspondingly which yields the reconstructed output f_i. The final output f_i is in the form $\langle v_i \ \mu_1, \mu_2, \ldots, \mu_j \rangle$ containing vertex with vertex id v_1 followed by membership values of the belonging clusters.

4 Experimental evaluation

The experiments were performed to evaluate the efficiency of the algorithm on 8 dell machines forming a Hadoop cluster each with 8GB of RAM, 1024 GB hard disk, having Ubuntu Linux OS installed. All the machines were connected via a gigabyte network. One of the node was considered as master node, and the rest were considered as slave nodes. We used Hadoop 2.6.0 and Giraph 1.1.0 for our experiments. Hadoop Distributed File System is used for storage.

The experiments were performed on four real-life datasets to illustrate the strengths and weakness of DFuzzy. Table 2 lists various datasets of large graphs used in experiments. All the datasets are taken from Stanford Large Network Dataset Collection [20]. Most of the considered graph datasets are from social networks. In social networks, social interactions and the diversity of people's interests suggest that the overlapping community structures exist. Also, in explicit groups present in social media Web sites, users are permitted to join more than one group depending on their personal interests and preferences. These overlaps are generally fuzzy as a person cannot be entirely involved with all the belonging communities due to narrow resources and time.

DFuzzy model can be executed on any commodity system cluster. Following are the parameters on the basis of which efficiency of DFuzzy is analyzed for large graphs. We consider the number of layers for fine tuning as 5 and the maximum number of iterations to be performed on each layer as 30. The clusters with low modularity value tend to have more inter-cluster edges than intra-cluster edges; thus, modularity threshold θ is taken as

0.2 [28]. The vertices of such cluster can be easily merged with other neighboring clusters. For computation of fuzzy membership values, fuzzyfication index m is considered as 2 [31].

The performance of the proposed DFuzzy is compared with four state-of-the-art clustering algorithms, namely fuzzy C-mean (FCM), label propagation, semi-clustering and BigClam. The native implementation of both label propagation and FCM has been modified for better performance in distributed environment. Also, we used the label propagation-based overlapping clustering approach for the fair comparison. For fair evaluation, number of clusters for FCM, semi-clustering, overlapping LPA and BigClam is taken as same as detected by the proposed DFuzzy. In FCM, the value of epsilon ϵ is considered to be $le - 3$ [31].

4.1 Run time

Run time of an algorithm is the time taken by a system to execute that algorithm. The total run time to execute the proposed model includes both the time spent on loading the graph to Giraph and the time to run DFuzzy. The communication overhead is reduced by using combiners of Giraph. Combiners aggregate the messages prior to transfer in each superstep. Combiners in DFuzzy aggregate all the PageRank values received from a worker in the

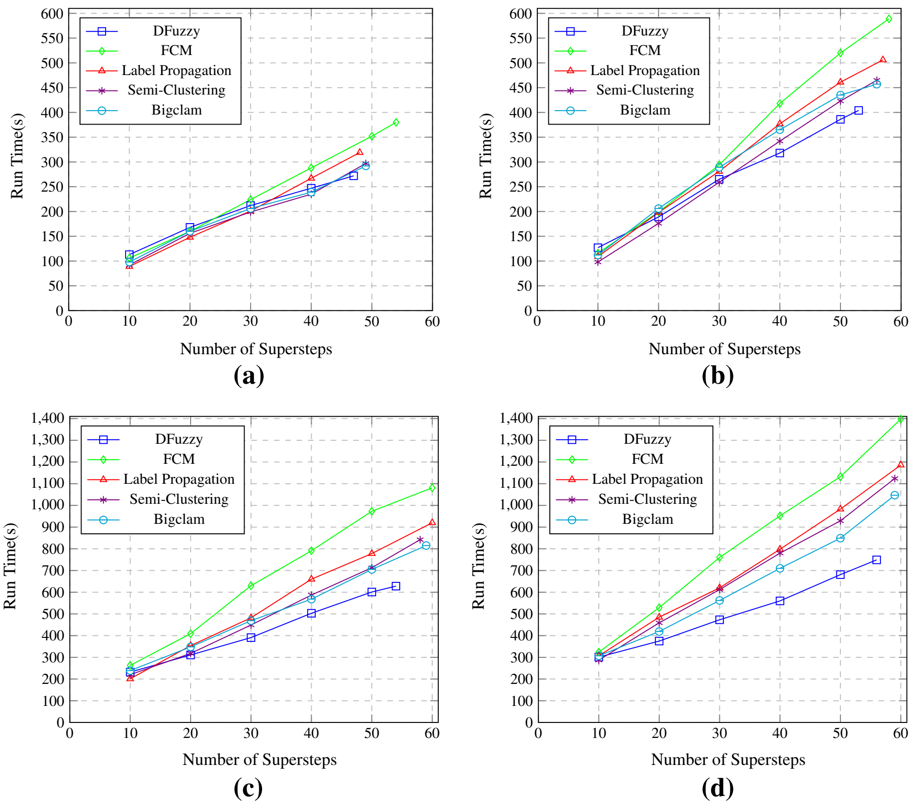

Fig. 6 Number of supersteps versus run time on graph datasets: **a** Facebook, **b** Twitter, **c** GooglePlus and **d** LiveJournal

superstep and forward it as a single grouped message. The lesser communication overhead here results in faster computation as shown in Fig. 6. The difference in supersteps is more noticeable for large graph datasets such as GooglePlus and LiveJournal.

It should be noted that computations involved in FCM, label propagation, semi-clustering and BigClam do not involve finding the number of clusters. Despite analyzing the structure of the network to find the number of clusters, DFuzzy achieves noticeable time efficiency in comparison with the other state-of-the-art clustering algorithms. As shown in Fig. 6, DFuzzy finishes its execution in lesser number of supersteps and is computationally less expensive than other algorithms for all the considered datasets. DFuzzy is efficient in terms of running time by an order of magnitude than the native FCM algorithm. The proficiency of proposed DFuzzy over all the algorithms is more noticeable for large graph datasets such as GooglePlus and LiveJournal.

4.2 Scalability

To analyze the scalability of DFuzzy, we varied the number of available workers P in Giraph job. In Fig. 7, the scalability of DFuzzy, label propagation, FCM, semi-clustering and Big-Clam algorithm is shown using up to eight nodes. As anticipated, increase in the number of processors results in decreased run time.

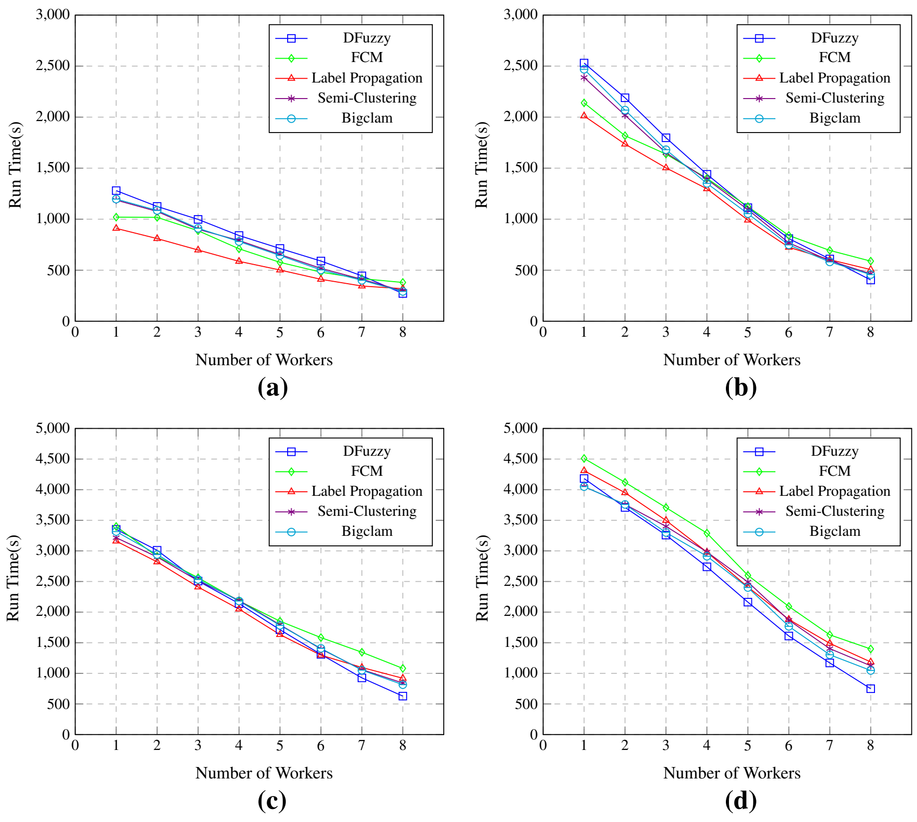

Fig. 7 Scalability: number of workers versus run time on **a** Facebook, **b** Twitter, **c** GooglePlus, **d** LiveJournal

After a certain number of processors, the synchronization and communication costs start to take over the computation overhead, and there is not much difference in computation time by adding further processors. Semi-clustering is highly scalable on Pregel framework. Label propagation and FCM perform calculations iteratively and thus can be scaled. But, FCM incurs high communication cost in distributed environment and hence is not computationally efficient. BigClam is also scalable in its native implementation. With added worker nodes, DFuzzy performs well in terms of run time and achieves a linear speedup as shown in Fig. 7.

The scalability of DFuzzy is tested using eight worker nodes only. However, if the number of worker nodes is further increased, DFuzzy will result in higher speedup values and will perform computations much faster than all the other competent algorithms.

4.3 Accuracy corresponding to ground truth data

The accuracy of DFuzzy for predicting the number of clusters, average cluster size and average membership of nodes toward various clusters is measured. The datasets mentioned in Table 2 with ground truth communities taken from [20] are considered for accuracy measurement. We find out the relevance between the clusters detected by DFuzzy against the ground truth data. As per the results shown in Fig. 8a, DFuzzy finds the number of clusters close to the ground truth communities for all the considered real-life networks. For the largest network LiveJournal, DFuzzy finds the number of clusters with 91.49% accuracy.

However, for Amazon, where the size of cluster is very small, DFuzzy results in finding more number of clusters in comparison with ground truth and thus provides lesser accurate results. Further, for data sources like YouTube, where the size of cluster is very large, DFuzzy is able to find almost accurate results corresponding to ground truth data.

The average cluster size and average membership obtained by DFuzzy are also close to the ground truth data as shown in Fig. 8b, c. For LiveJournal, DFuzzy is 97.3 and 96.1% more accurate in terms of average cluster size and average membership of vertices, respectively, corresponding to ground truth data. Most of the clusters are crisp for DBLP network. DFuzzy finds both crisp and fuzzy clusters accurately for DBLP as shown in Fig. 8b. The proposed model is 2.1 times accurate than native FCM when compared with ground truth data in terms of membership of nodes toward clusters.

4.4 Cluster quality

The efficiency of the DFuzzy is measured on the basis of modularity, conductance and partition coefficient. Modularity is explained in Sect. 2.3 in detail. Conductance can be calculated over a cut or cluster boundary, and partition coefficient measures the extent of overlapping between two or more clusters. Partition coefficient measures the amount of overlapping between the clusters. DFuzzy optimizes the clusters using modularity in the precision enhancement phase. Higher is the value of modularity and conductance, better is the quality of cluster. Here, we considered the intra-cluster conductance. For partition coefficient, lower is the value, better are the clustering results.

Figure 9a displays the composite performance of DFuzzy along with the competent approaches over the four graph datasets. On average, the composite modularity of DFuzzy is 0.695, which is 33, 78, 39 and 63% higher than that of overlapping label propagation (0.520), FCM (0.39), semi-clustering (0.496) and BigClam (0.425), respectively. The high modular-

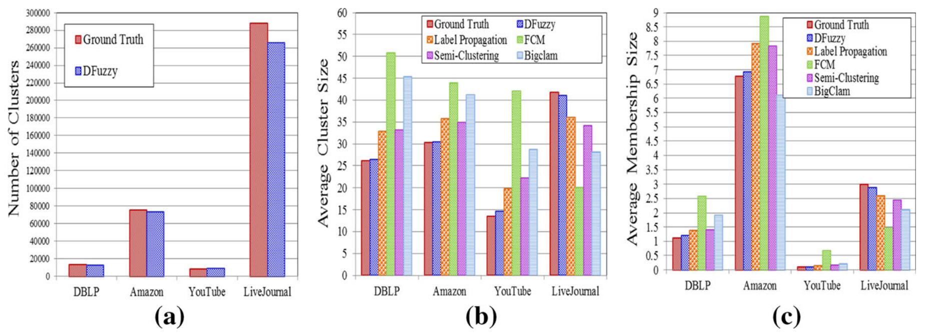

Fig. 8 Accuracy corresponding to ground truth data. **a** Number of clusters, **b** average cluster size, **c** average membership size

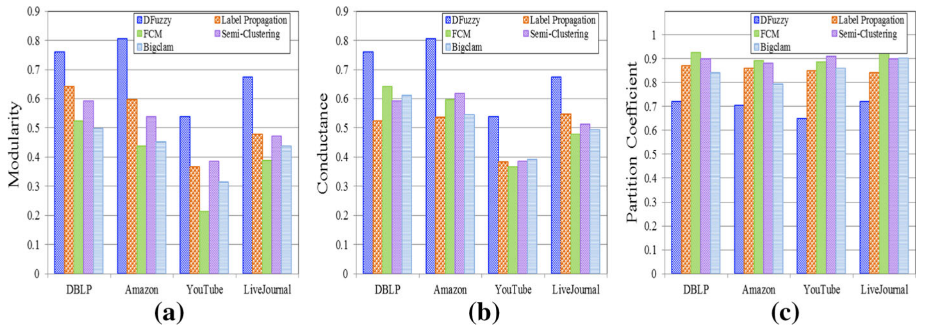

Fig. 9 Quality measurement in terms of **a** modularity, **b** conductance and **c** partition coefficient

ity value of DFuzzy for all the considered graphs indicates that it discovers the overlapping communities efficiently.

Figure 9b exhibits the composite performance of DFuzzy on the basis of conductance. The average conductance of DFuzzy is 0.645, which is 39, 33, 31 and 36% higher than that of overlapping label propagation (0.497), FCM (0.520), semi-clustering (0.526) and BigClam (0.509), respectively. The high conductance value of DFuzzy indicates that it has lesser inter-cluster links in comparison with other algorithms.

The value of partition coefficient over the considered four graph datasets is shown in Fig. 9. The absolute average value for partition coefficient of DFuzzy is 0.696, while for overlapping label propagation is 0.855, FCM is 0.91, semi-clustering is 0.89, and BigClam is 0.848. Thus, DFuzzy is 22, 30, 28 and 21% more efficient than overlapping label propagation, FCM, semi-clustering and BigClam, respectively.

DFuzzy based on autoencoder is proved to gain a significant advantage over the other competent algorithms in terms of quality. Hence, deep structures provide better graph representations and improved clustering results.

4.5 Effect of deep layers on quality

For showing the effect of stacked autoencoders on processing time and accuracy, we varied the number of layers of the autoencoder pipeline in DFuzzy. The modularity of DFuzzy is shown for the shallowest 2-layer model to the deepest 6-layer model in Fig. 10. The effect of

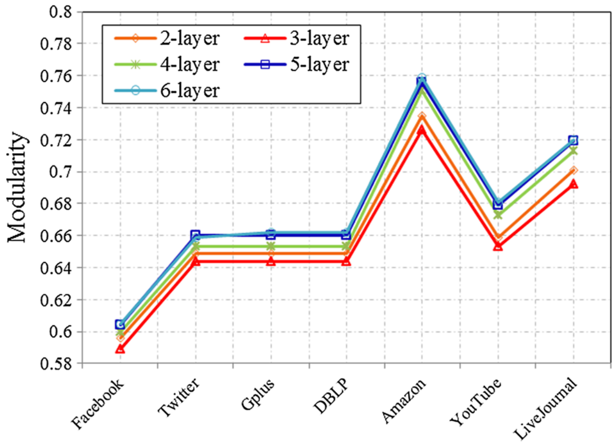

Fig. 10 Effect of multiple layers: improvement in quality in terms of modularity

the number of autoencoder layers can be observed from the modularity gain when the layers become deeper.

However surprisingly, 3-layer model is less efficient than 2-layer model for all the considered datasets. But for all the further cases, DFuzzy performs later while reaching to deep layers from shallow layers. Also, it can be noticed that the difference between modularity values of 5-layer model and 6-layer model is minimal. Therefore, we used 5-layer DFuzzy model for all the computations performed on DFuzzy.

From the results, it can be observed that the deep structures play a vital role for the proposed model in generating communities of good quality and make the model efficient. The clustering results became more accurate by using autoencoders on reaching to deep layers from shallow layers.

5 Conclusion

In this paper, a fuzzy graph clustering model named ‘DFuzzy’ is proposed, which performs clustering by leveraging the idea from deep learning pipelines. A layer-wise pertaining order is used for finding cluster centers and for adding vertices to the clusters based on the analysis of graph structure. Hadoop and Pregel frameworks are used for the scalable implementation of DFuzzy which make the whole learning process adaptable. Experimental results on various real-world graph datasets have shown that DFuzzy finds the fuzzy clusters efficiently and takes less computational time, when compared with the state-of-the-art algorithms such as label propagation, FCM, semi-clustering and BigClam. DFuzzy saves time by computing fuzzy membership for only those vertices which lies in fuzzy partition. Accuracy of the proposed DFuzzy is measured using the ground truth data. It is demonstrated that DFuzzy finds the clusters of varying sizes with high accuracy and achieves better clustering results in terms of modularity, conductance and partition coefficient. It is also proved that DFuzzy takes significant gain in accuracy by using deep autoencoder layers.

The proposed model considers the local clusters only using PageRank. In future, the model can be enhanced considering the clustering results globally in distributed environment. Also, the model can be tested with graphs having billions of vertices using more number of machines of higher configurations and can be optimized to reduce the computational cost.

References

1. Apolloni B, Bassis S, Rota J, Galliani GL, Gioia M, Ferrari L (2016) A neurofuzzy algorithm for learning from complex granules. Granul Comput 1(4):225–246
2. Bahmani B, Chakrabarti K, Xin D (2011) Fast personalized pagerank on mapreduce. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data. ACM, pp 973–984
3. Bampis CG, Maragos P, Bovik AC (2017) Graph-driven diffusion and random walk schemes for image segmentation. IEEE Trans Image Process 26(1):35–50
4. Banijamali E, Ghodsi A (2017) Fast spectral clustering using autoencoders and landmarks. arXiv preprint arXiv:1704.02345
5. Bezdek JC, Ehrlich R, Full W (1984) FCM: the fuzzy c-means clustering algorithm. Comput Geosci 10(2):191–203
6. Bhatia V, Rani R (2017) A parallel fuzzy clustering algorithm for large graphs using pregel. Expert Syst Appl 78:135–144
7. Chen XW, Lin X (2014) Big data deep learning: challenges and perspectives. IEEE Access 2:514–525
8. Ciresan D, Meier U, Schmidhuber J (2012) Multi-column deep neural networks for image classification. In: Proceedings of IEEE conference on computer vision and pattern recognition (CVPR). IEEE, pp 3642–3649
9. Ciucci D (2016) Orthopairs and granular computing. Granul Comput 1(3):159–170
10. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
11. Deng L, Yu D, Platt J (2012) Scalable stacking and learning for building deep architectures. In: Proceedings of IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp. 2133–2136
12. Gregory S (2010) Finding overlapping communities in networks by label propagation. New J Phys 12(10):103,018
13. Havens TC, Bezdek JC, Leckie C, Hall LO, Palaniswami M (2012) Fuzzy c-means algorithms for very large data. IEEE Trans Fuzzy Syst 20(6):1130–1146
14. Havens TC, Bezdek JC, Leckie C, Ramamohanarao K, Palaniswami M (2013) A soft modularity function for detecting fuzzy communities in social networks. IEEE Trans Fuzzy Syst 21(6):1170–1175
15. He T, Chan KC (2016) Evolutionary graph clustering for protein complex identification. IEEE/ACM Trans Comput Biol Bioinform. https://doi.org/10.1109/TCBB.2016.2642107
16. Hubel DH, Wiesel TN (1962) Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J Physiol 160(1):106–154
17. Hutchinson B, Deng L, Yu D (2013) Tensor deep stacking networks. IEEE Trans Pattern Anal Mach Intell 35(8):1944–1957
18. Kang U, Tsourakakis CE, Faloutsos C (2011) Pegasus: mining peta-scale graphs. Knowl Inf Syst 27(2):303–325
19. Kianmehr K, Alshalalfa M, Alhajj R (2010) Fuzzy clustering-based discretization for gene expression classification. Knowl Inf Syst 24(3):441–465
20. Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data. Accessed 15 Feb 2017
21. Lingras P, Haider F, Triff M (2016) Granular meta-clustering based on hierarchical, network, and temporal connections. Granular Comput 1(1):71–92
22. Liu L, Chen X, Liu M, Jia Y, Zhong J, Gao R, Zhao Y (2016) An influence power-based clustering approach with pagerank-like model. Appl Soft Comput 40:17–32
23. Liu L, Sun L, Chen S, Liu M, Zhong J (2016) K-prscan: a clustering method based on pagerank. Neurocomputing 175:65–80
24. Ludwig SA (2015) Mapreduce-based fuzzy c-means clustering algorithm: implementation and scalability. Int J Mach Learn Cybern 6(6):923–934
25. Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 135–146
26. Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E (2015) Deep learning applications and challenges in big data analytics. J Big Data 2(1):1–21
27. Nepusz T, Petróczi A, Négyessy L, Bazsó F (2008) Fuzzy communities and the concept of bridgeness in complex networks. Phys Rev E 77(1):016,107
28. Newman ME (2006) Modularity and community structure in networks. Proc Natl Acad Sci 103(23):8577–8582

29. Niepert M, Ahmed M, Kutzkov K (2016) Learning convolutional neural networks for graphs. In: International conference on machine learning, pp 2014–2023
30. Page L, Brin S, Motwani R, Winograd T (1999) The pagerank citation ranking: bringing order to the web. Stanford InfoLab, Stanford
31. Pal NR, Bezdek JC (1995) On cluster validity for the fuzzy c-means model. IEEE Trans Fuzzy Syst 3(3):370–379
32. Peters G, Weber R (2016) Dcc: a framework for dynamic granular clustering. Granul Comput 1(1):1–11
33. Raina R, Madhavan A, Ng AY (2009) Large-scale deep unsupervised learning using graphics processors. In: Proceedings of the 26th annual international conference on machine learning. ACM, pp 873–880
34. Robinson I, Webber J, Eifrem E (2015) Graph databases new opportunities for connected data. O'Reilly Media, Newton
35. Sanchez MA, Castro JR, Castillo O, Mendoza O, Rodriguez-Diaz A, Melin P (2017) Fuzzy higher type information granules from an uncertainty measurement. Granul Comput 2(2):95–103
36. Schaeffer SE (2007) Graph clustering. Comput Sci Rev 1(1):27–64
37. Shao M, Li S, Ding Z, Fu Y (2015) Deep linear coding for fast graph clustering. In: IJCAI, pp 3798–3804
38. Šíma J (1996) Back-propagation is not efficient. Neural Netw 9(6):1017–1023
39. Skowron A, Jankowski A, Dutta S (2016) Interactive granular computing. Granul Comput 1(2):95–113
40. Song C, Liu F, Huang Y, Wang L, Tan T (2013) Auto-encoder based data clustering. In: Progress in pattern recognition, image analysis, computer vision, and applications. Springer, pp 117–124
41. Song M, Wang Y (2016) A study of granular computing in the agenda of growth of artificial neural networks. Granul Comput 1(4):247–257
42. Tian F, Gao B, Cui Q, Chen E, Liu TY (2014) Learning deep representations for graph clustering. In: Proceedings of 28th conference on artificial intelligence (AAAI-14), pp 1293–1299
43. Timón I, Soto J, Pérez-Sánchez H, Cecilia JM (2016) Parallel implementation of fuzzy minimals clustering algorithm. Expert Syst Appl 48:35–41
44. Wang G, Yang J, Xu J (2017) Granular computing: from granularity optimization to multi-granularity joint problem solving. Granul Comput 2(3):105–120
45. Wu Z, Gao G, Bu Z, Cao J (2016) Simple: a simplifying-ensembling framework for parallel community detection from large networks. Cluster Comput 19(1):211–221
46. Xie J, Girshick R, Farhadi A (2016) Unsupervised deep embedding for clustering analysis. In: International conference on machine learning, pp 478–487
47. Yang J, Leskovec J (2013) Overlapping community detection at scale: a nonnegative matrix factorization approach. In: Proceedings of the sixth ACM international conference on Web search and data mining. ACM, pp 587–596
48. Yang JX, Zhang XD (2017) Finding overlapping communities using seed set. Physica A Stat Mech Appl 467:96–106
49. Yang L, Cao X, He D, Wang C, Wang X, Zhang W (2016) Modularity based community detection with deep learning. In: Proceedings of the twenty-fifth international joint conference on artificial intelligence. AAAI Press, pp 2252–2258
50. Yao Y (2016) A triarchic theory of granular computing. Granul Comput 1(2):145–157
51. Yoon SH, Kim KN, Hong J, Kim SW, Park S (2015) A community-based sampling method using dpl for online social networks. Inf Sci 306:53–69
52. Zhang K, Chen XW (2014) Large-scale deep belief nets with mapreduce. IEEE Access 2:395–403

Vandana Bhatia is currently working toward the Ph.D. degree in computer science from Computer Science and Engineering Department at Thapar University, Patiala, India. Her research interests include Big Data analytics, Graph Mining and Soft Computing. She received a Masters in Technology in Computer Engineering from Kurukshetra University. She has contributed 6 research articles in Journals and conferences. She is also a member of ACM and IEEE. Contact her at vbhatia91@gmail.com.

Dr. Rinkle Rani is working as Associate Professor in Computer Science and Engineering Department, Thapar University, Patiala. She has done her postgraduation from BITS, Pilani and Ph.D. from Punjabi University, Patiala. She has more than 20 years of teaching experience. She has supervised 3 Ph.D. and 43 M.Tech. dissertations. She has contributed 54 articles in Conferences and 47 papers in Research Journals. Her areas of interest are Computer Networks, Big data mining and Analysis. She is member of professional bodies such as ACM, IEEE, ISTE and CSI. She may be contacted at raggarwal@thapar.edu.