Expert Systems With Applications 78 (2017) 135–144

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A parallel fuzzy clustering algorithm for large graphs using Pregel

Vandana Bhatia, Rinkle Rani*

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

ARTICLE INFO

Article history:
Received 12 December 2015
Revised 2 February 2017
Accepted 3 February 2017
Available online 8 February 2017

Keywords:
Clustering
Graphs
Big data mining
Fuzzy C-Mean
Pregel

ABSTRACT

Large graphs are scale free and ubiquitous having irregular relationships. Clustering is used to find existent similar patterns in graphs and thus help in getting useful insights. In real-world, nodes may belong to more than one cluster thus, it is essential to analyze fuzzy cluster membership of nodes. Traditional centralized fuzzy clustering algorithms incur high communication cost and produce poor quality of clusters when used for large graphs. Thus, scalable solutions are obligatory to handle huge amount of data in less computational time with minimum disk access. In this paper, we proposed a parallel fuzzy clustering algorithm named 'PGFC' for handling scalable graph data. It will be advantageous from the viewpoint of expert systems to develop a clustering algorithm that can assure scalability along with better quality of clusters for handling large graphs.The algorithm is parallelized using bulk synchronous parallel (BSP) based Pregel model. The cluster centers are initialized using degree centrality measure, resulting in lesser number of iterations. The performance of PGFC is compared with other state of art clustering algorithms using synthetic graphs and real world networks. The experimental results reveal that the proposed PGFC scales up linearly to handle large graphs and produces better quality of clusters when compared to other graph clustering counterparts.

1. Introduction

Big data has great amount of hidden knowledge and many insights which have raised remarkable challenges in knowledge discovery and data mining. For certain types of data, the relationship between the entities is of much more importance then the information itself. Big data has many such connections which can be mined efficiently using graphs (Kang & Faloutsos, 2013). However, it is very challenging to obtain ample profits from this ostensibly frenzied data. Valuable hidden information and patterns cannot be discovered easily from graphs without using sophisticated methods (Cook & Holder, 2006).

To overcome the challenging problems, clustering techniques are used based on certain similarity to produce a fine quality of clusters. A set of vertices which have some mutual properties or play roles similar in nature within the whole network are collected to form a cluster (Schaeffer, 2007). Sometimes, a graph cluster is also called as community, particularly in social networks (Fortunato, 2010). Clustering can be done among several transactional graphs or within one large graph having more links with in cluster and fewer links between clusters. In this paper clustering within a single large graph is considered.

In general, clustering can be hard or soft. With hard clustering, each vertex is assigned to a single cluster, whereas with soft or fuzzy clustering, vertices may belong to more than one cluster. K-Means is the most popular hard clustering algorithm which partitions the graph in pre-specified number of clusters by optimizing the objective function. To analyze clusters in large graphs, the detection of fuzzy clusters is of high concern. Many such cases exist in social networks where users may belong to several communities (Fortunato, 2010). In biological networks, a large segment of proteins may belong to numerous protein complexes at the same time. Also genes may also have various coding functions and may participate in more than one metabolic pathway (Nepusz, Yu, & Paccanaro, 2012). In information retrieval and text mining, documents (Kurland & Lee, 2004), news articles and web pages can belong to many different categories.

For finding overlapping clusters, a soft variant of K-Means was introduced by Bezdek, Ehrlich, and Full (1984) known as Fuzzy C-Means (FCM) where vertices may belong to different clusters with certain membership degrees. FCM is an unsupervised clustering algorithm that is realistic to widespread problems concerned with image processing (Rahimi, Zargham, Thakre, & Chhillar, 2004; Son, 2015), time series forecasting (Egrioglu, Aladag, & Yolcu, 2013), Social Networks (Golsefid, Zarandi, & Bastani, 2015; Su, Wang, & Yu, 2014), Geo-demographic analysis (Son, Cuong, Lanzi, & Thong, 2012), Intrusion Detection (Wang, Hao, Ma, & Huang, 2010), Network Optimization (Wang, Ma, Lao, & Wang, 2014), etc.

* Corresponding author.
E-mail addresses: vbhatia91@gmail.com (V. Bhatia), raggarwal@thapar.edu (R. Rani).

http://dx.doi.org/10.1016/j.eswa.2017.02.005
0957-4174/

But, FCM is sensitive to selection of initial centers and selects them randomly (Bezdek et al., 1984). The clustering results produced could be erroneous as random initialization generally leads to convergence at local minima. Moreover, vertices belonging to more than one cluster are usually located at the boundary of clusters, while the vertices with high degree take the central positions in clusters and are imperative for the stability of the clusters (Opsahl, Agneessens, & Skvoretz, 2010). The initial cluster centers computed using degree centrality measure are found to be very close to the desired cluster centers for iterative clustering algorithms like FCM.

Most of the existing fuzzy clustering algorithm are for centralized systems (Fortunato, 2010; Kesemen, Tezel, & Özkul, 2016; Király, Vathy-fogarassy, & Abonyi, 2015; Özbay, Ceylan, & Karlik, 2006). However, for large graphs from big data, it is highly desirable to be distributed instead of centrally stored. Furthermost existing research focused on the small data and use traditional graph mining approaches which are not trivial to paralyze. Among the existing graph algorithms, some of them can be used to deal with the large data (Fahad et al., 2014; Malek, Golsefid, Hossien, & Zarandi, 2015; Nepusz, Petrczi, Ngyessy, & Bazs, 2008) but they are not able to deal with large graphs and the related scalability issues.

In this paper, a parallel fuzzy clustering algorithm PGFC is proposed by amending the structure of the classical Fuzzy C-Means algorithm for large graph data. Degree measure has been taken for initialization of cluster center. BSP based Pregel model is followed to design the distributed version of the algorithm. The efficiency of the proposed approach is validated through experimental results.

The rest of the paper is structured as follows. Section 2 explains clustering task on graphs, the problem definition and the introduction of programming models used. In Section 2, traditional Fuzzy C-Means and K-Means algorithms are also explained. In Section 3, literature survey is given. The Pregel based K-Means and Semi-clustering algorithm are discussed in Section 4. The proposed distributed fuzzy graph clustering algorithm PGFC is described in Section 5. The results of the performed experiments are shown in Section 6. Section 7 precisely presents conclusion with recommendations for further research.

2. Preliminaries and background

In this section, an intuitive explanation of the clustering of graph data has been provided. A brief introduction of Pregel for graph computations is given. At the end of this section, the K-Means and Fuzzy C-Means algorithms are discussed.

2.1. Clustering in graphs

A graph G = {*V, E*} is a finite set of objects called vertices *V* and the relationship among vertices is represented by the set of edges $E \subseteq V \times V$. Set *V* contains the vertices and set *E* defines the structure of graph. For getting better insights from a graph, clustering techniques are used. A graph cluster is generally anticipated as a connected component having similar vertices within the cluster and few edges between clusters. Similarity function is defined on the basis of edge structure. A partition based clustering algorithm partitions a large graph into clusters. The partition may be hard or soft.

Definition 1. A hard clustering of a graph $G = \{v_1, v_2, \ldots, v_n\}$ is an assignment of vertices v_i to k cluster sets C_k such that $\cup_{j=1}^{k} |C_k| = |C|$ and $C_i \cap C_j = \phi$ for each cluster pair i and j. Therefore:

$$\sum_{j=1}^{k} |C_j| = |C| \tag{1}$$

A partition is denoted as C = {C_j}.

Definition 2. A soft clustering of a graph $G = \{g_1, g_2, \ldots, g_n\}$ is an assignment of vertices v_i to k cluster sets C_k where the fuzzy membership operator ε_m quantifies the degree of vertex v_i towards cluster C_j such that $v_i \, \varepsilon_m \, C_j = \mu_{i,j} \in [0,1]$. A fuzzy partition is denoted as $C = \{C_j\}_m$ and fuzzy partitioning creates subgraphs which have pairwise overlapping. Therefore:

$$\sum_{i=1}^{n} \sum_{j=1}^{k} \mu_{i,j} = |C| \tag{2}$$

The actual overlapping depends on the number of clusters and on the selected membership function. And the appropriate selection of initial cluster centers reduces the time complexity. In a graph, each node has a varying degree. The nodes with high degree have relatively more connections than other nodes. So, nodes with high degree are appropriate candidates for cluster heads.

Definition 3. The degree of a node $v_i \in V$ is defined as the cardinality of its adjacent vertices which are connected to node v_i through an edge such that $deg(v_i) = |\, N(v_i)\, |$.

2.2. Problem definition

In this paper, Pregel based Giraph is used for dealing with large graphs in which the major cost is of communication among the nodes. Formally:

Definition 4. (*Communication Cost for cluster* C_j) For a cluster *C* of graph *G(V,E)*, the communication cost for cluster C_j is given by $cost(C_j) = \sum_{v \in V} cost(v)$, where *cost(v)* is the number of clusters having adjacent vertices to vertex *v*.

A fuzzy clustering algorithm for large graphs is introduced in this paper which is formalized as: To partition a graph into *k* overlapping clusters such that the edge cut size between clusters or the communication cost is minimized in the distributed framework.

Problem. Definition (*Graph Fuzzy Clustering*) Given a graph *G(V,E)* and a positive integer *k*, we partition *V* into a set of overlapping clusters $C = (C_1, C_2, \ldots, C_k)$ such that $cost(C_i)$ is minimized.

2.3. Cluster quality measures

There are some good measures for measuring the quality of clusters. Following clustering measures are used in this paper to analyze the quality of fuzzy clusters.

Partition Coefficient: It measures the amount of overlapping between clusters and is given as:

$$PC = \frac{1}{N} \sum_{i=1}^{k} \sum_{j=1}^{N} {\mu_{ij}}^2 \tag{3}$$

where, $\frac{1}{k} \le PC \le 1$. Greater is the value of *PC*, lesser is the overlapping among clusters. Valid clusters are generated for $k = (2,3,\ldots,N\text{-}1)$ by solving $\max_k\{\max(PC)\}$for the graph *G*.

Conductance: It can be defined as the ratio of number of inter cluster edges to the minimum number of edges incident on either cluster C_k or $\bar{C}_k$:

$$Cond(C_k) = \frac{\sum_{i \in C_k, j \in \bar{C}_k} A_{ij}}{\min(A(C_k), A(\bar{C}_k))} \tag{4}$$

where, A_{ij} is the adjacency matrix of graph *G*, $C_k \in V$ and $A(C_k)$is the number of edges incident on C_k:

$$A(C_k) = \sum_{i \in C_k} \sum_{j \in V} A_{ij} - \sum_{i \in C_k} \sum_{j \in C_k} A_{ij}$$

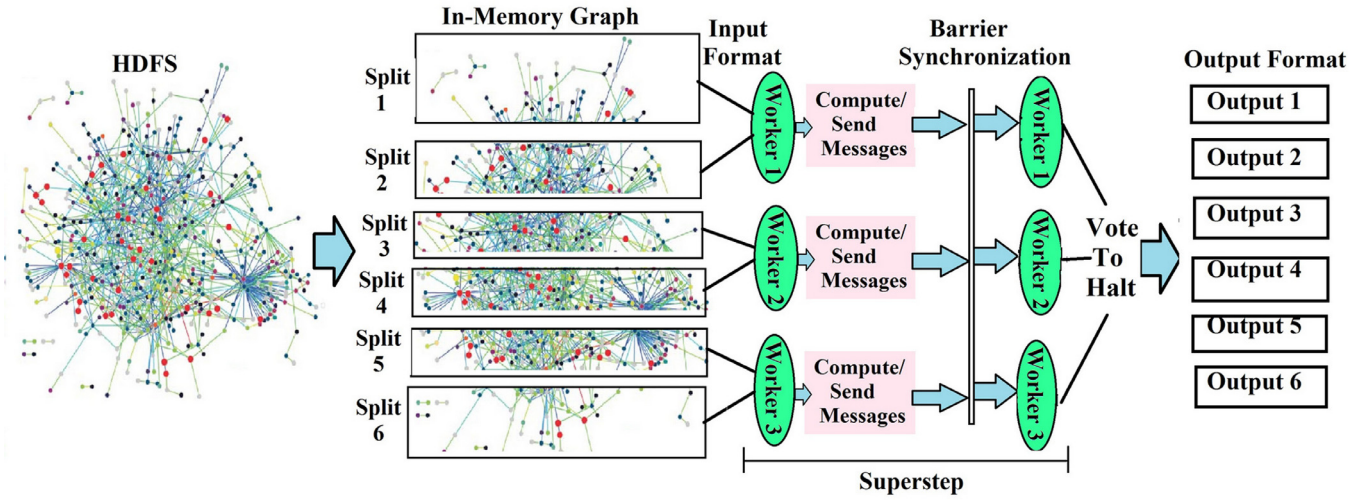

Fig. 1. Graph processing in Giraph.

2.4. Pregel for graph computing

The most popular framework for handling big data is Map-Reduce which runs on the top of Hadoop. Map-Reduce is a programming paradigm for dealing with huge data in a distributed cluster of commodity machines. It refers to two distinct tasks map and reduce processes. The mapper reads the input file and produces key-value pairs as output. The shuffling stage sorts the output to group the pairs with the same key and distribute them to reducers. The reducer processes the key-values with same key and produces other key-value pairs as output (Dean & Ghemawat, 2008). Several Map-Reduce jobs can be performed iteratively for processing of the large graphs. But, in Map-Reduce paradigm, intermediate results of every iteration needs to be materialized as entire graph structure need to be sent over network in each iteration which causes huge unnecessary data movement.

For handling such distributed graph, Pregel (Malewicz et al., 2010) based structures are considered which performs vertex based computation by dividing the processing in supersteps and substeps. It is a graph analytic environment designed for big graphs around bulk synchronous parallel(BSP) model. It works in a distributed manner by dividing vertices of a large graph among the workers in the cluster and performing computations individually at each worker node. Communication among the workers occurs by passing messages which are generally small in comparison to data and thus more efficient for transmission. The algorithm terminates when no messages are produced during iteration or until each worker node votes to halt.

Bulk synchronous parallel (BSP) is a paradigm to design parallel algorithms with two basic operations Communication via sending messages and barrier Synchronization. A BSP computation consists of a sequence of supersteps. Messages from the previous superstep are available in next superstep. In this paper, iterative BSP based graph processing framework Giraph is used to perform offline batch processing of semi-structured graph data. Giraph performs iterative calculations on the top of the Hadoop cluster by avoiding costly disk and network operations mandatory in map-reduce framework using out of core in-memory execution. Only intermediate values in the form of messages are sent across the network. The disk access takes place only while fetching input from Hadoop distributed file system (HDFS), storing output back in HDFS and for storing checkpoints. Each cycle of an iterative Giraph calculation on Hadoop runs a full map-reduce job with only mappers.

The input graph is partitioned among the worker nodes as shown in Fig. 1. Vertices are loaded from HDFS using a user specified input format. Workers call the compute() function on the active vertices and collect the messages sent in previous superstep. Every vertex executes in each superstep, recomputes its values and sends massages to all other vertices. The message transfer takes place over a superstep barrier. Workers then compute aggregators, collect statistics and wait at the synchronization barrier for more vertices and messages to be processed until all workers votes to halt. At last vertices are offloaded to HDFS through an output format. Using checkpoints, fault tolerance is achieved. All workers save the state of their partition in HDFS after getting the instruction from the master at the beginning of each superstep. Worker nodes save their vertex values, edge values and the values of incoming messages which are then aggregated by master node.

2.5. K-Means algorithm

K-Means is a partitioning based clustering method that attempts to find pre-defined number of clusters (k), represented by their centroids, by minimizing the square error function defined as:

$$F = \sum_{i=1}^{n} \sum_{j=1}^{k} x_{ij} d(v_i, c_k) \tag{5}$$

Given a graph G with a set of vertices $V=(v_1,v_2,v_3, \ldots\ldots, v_n)$ and a positive integer k, the K-Means algorithm partitions vertices V into k clusters $(c_1,c_2,c_3, \ldots,c_k)$, represented by their centers or means. The K-Means algorithm starts by initializing k cluster centers. The vertices of input graph are then assigned to one of the current clusters according to the square of the Euclidean distance from the clusters, selecting the closest. The center of each cluster is then calculated as the mean of all the vertices belonging to that cluster.

$$u_k = \frac{1}{N_k} \sum_{q=1}^{N_k} v_q \tag{6}$$

where, N_k is the number of vertices belonging to cluster k and μ_k is the mean of the cluster k. The centroid of each cluster is then computed in order to update the cluster center. The cluster centers are updated as a result of the change in the membership value for each cluster. The process of re-assigning the vertices and updating the cluster centers repeats until the value of all the cluster centers stops changing.

Algorithm 1 Fuzzy C-Means algorithm.

Input: Fuzziness index m; No. of clusters C ($2 < C < N$); termination criteria $\varepsilon > 0$.
Step 1: Initialize the Membership matrix $U_{ij} = \{\mu^k{}_{ij}\}$
Step 2: Calculate the fuzzy Membership using:

$$\mu_{ij}^{k} = \frac{1}{\sum_{k=1}^{c} \left\{ \frac{x_i - c_j}{x_i - c_k} \right\}^{\frac{2}{m-1}}} \tag{8}$$

Step 3: Compute the Fuzzy cluster center c_j for all the cluster such that $1 \le j \le k$ using:

$$c_j = \frac{\sum_{i=1}^{n} (\mu_{ij}^{k})^m . v_i}{\sum_{i=1}^{n} (\mu_{ij}^{k})^m} \tag{9}$$

Step 4: Repeat Step 2 and Step 3 until the minimum J value is achieved OR if

$$\forall i, j : \max_{ij} \left\{ \mu_{ij}^{k+1} - \mu_{ij}^{k} \right\} < \epsilon \tag{10}$$

Output: Final membership Matrix U_{ij} and final Cluster centers.

2.6. Fuzzy C-Means algorithm

One of the most prevalent fuzzy clustering algorithms is Fuzzy-C-Mean algorithm. It attempts to group/partition a graph such that a set of vertices forming similar patterns are assigned to same cluster. It was initially developed by Dunn in 1973 and further upgraded by Bezdek, Ehrlich, and Full (1984).

The Fuzzy C-Means is an overlapping data clustering algorithm which attempts to partition a finite collection of vertices $V = \{v_1, v_2, v_3, \ldots\ldots\ldots, v_N\}$ into $C = \{c_1, c_2, c_3, \ldots, c_k\}$ fuzzy clusters 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 in accordance to the distance and the membership value.

$$J = \sum_{i=1}^{N} \sum_{j=1}^{C} \left(U_{ij}\right)^m \|v_i - c_j\|^2 \qquad 1 \le m \le \infty \tag{7}$$

where m is the fuzziness index of value greater than 1 and U_{ij} is the membership of v_i in the jth cluster in D-dimensional data. Fuzzy clusters are obtained through optimization of objective function in (7) iteratively by updating membership matrix U_{ij} and cluster center c_j as described in Algorithm 1. Convergence is reached when the edge cut is less than threshold ε.

3. Related work

Clustering techniques can be categorized into partition based, hierarchical, density based and grid based (Fahad et al., 2014). The most popular partition based clustering algorithm is K-Means. Many modifications have been done on K-Means to improve its performance (Arthur, 2007; Bahmani, Moseley, Vattani, Kumar, & Vassilvitskii, 2012; Kanungo et al., 2002). The distributed improved version of K-Means was also introduced in literature (Xu et al., 2014; Zhao, Ma, & He, 2009). However, K-Means finds the non-overlapping clusters only.

Fuzzy C-Means (FCM) algorithm is among the most widely used algorithm for finding overlapping among clusters. It was first introduced by Dunn (1973) and then improved by Bezdek et al., 1984. Although, the traditional FCM works well in most of the applications, but the random initialization of cluster centers leads the iterative process to converge at local optimum solution. For solving this issue, many improvements to FCM have been proposed (Timón, Soto, Pérez-sánchez, & Cecilia, 2016). Some improvements implicates the alteration of objective function (Wikaisuksakul, 2014). The approaches such as neural networks were used along with FCM in some applications such as forecasting using time-series data (Egrioglu et al., 2013) and classifying ECG signals (Özbay et al., 2006). Certain similarity measures too have been used in literature such as vertex similarity (Nepusz et al., 2008), Euclidean distance (Khalilia, Bezdek, Popescu, & Keller, 2014). Several other overlapping cluster detection algorithms were also proposed in literature (Chen, 2015; Vehlow, Member, Reinhardt, Weiskopf, & Society, 2013).

Most of the existing algorithms work on the centralized system only. However, in this era of big data, distributed fuzzy clustering algorithms are required. A few researches have been done on fuzzy clustering using distributed systems. An incremental multiple medoids-based fuzzy clustering approach was suggested for handling complex patterns that are not well separated and compact using distributed network (Zhou, Member, Chen, & Chen, 2014). A map- reduce based implementation of FCM was proposed by Ludwig (2015). Some parallel methods were also proposed such as Parallel Fuzzy Minimal (PFM), a parallel implementation of the fuzzy minimals clustering algorithm (Timón et al., 2016). But they all have high communication cost and require heavy disk access.

Although, many fuzzy clustering algorithms were developed and used widely in many applications, little research has been found on fuzzy graph clustering in distributed environment. Fuzzy clustering in graphs was quite a concerning area in the last decade. Some alternatives were also proposed. For Complex networks, a relation based approach was introduced instead of graphs for find overlapping as well as non-overlapping communities (Sun, Gao & Han, 2011). But for dealing with large graphs, distributed fuzzy clustering algorithms on frameworks like Pregel are required. A Pregel based Semi-clustering (Malewicz, Austern, Bik, Dehnert, & Horn, 2010) was introduced for handling large graphs. But, it requires complex operations and do not find the extent of belongingness towards the candidate cluster centers. The comparison of various fuzzy clustering algorithms for graphs with the proposed PGFC, on the basis of advantages and limitations given by the respective authors is given in Table 1.

4. Pregel based clustering algorithms

This section explains two Pregel based cluster analysis methods, namely K-Means clustering and semi-clustering algorithm. K-Means is not a native graph clustering application. But due to the iterative nature, K-Means can easily maps into Pregel model where vertices are taken as input vectors and every centroid calculation can be done by aggregators. Master handles all the aggregators and shares the values to all the workers.

4.1. K-Means algorithm in Pregel

K-Means clustering algorithm is the simplest partitioning algorithm which partitions the vertices of a graph into K clusters in an iterative manner. A preSuperstep() method is executed on every worker node before the input vertices are processed to prepare them for computation. Algorithm 2 shows the details of the computation for K-Means.

The description about the compute() function running on master node, called at the beginning of each superstep is given in Algorithm 2. It is used to perform the centralized computation between supersteps with the help of aggregators. It starts with the random selection of the cluster centers and passing the information about selected centers to all the workers. The cluster centers are updated in further iterations by calculating the arithmetic mean of the assigned points until the convergence is reached. The

Table 1
Comparison of fuzzy clustering algorithms for graphs.

Algorithm	Scalability	Graph data handling	Advantages	Limitations
MR-FCM	Scalable	Yes, but it incurs high communication cost	Finds both fuzzy clusters and membership	Convergence at local minima
E. Egrioglu et al.	Not scalable	Yes, suitable for time-series graphs only	Difficult matrix operations are avoided	Appropriate for only Time-Series data
iRFCM	Not Scalable	Yes, but it involve expensive matrix calculations	Improved Euclidean distance matrix	High Time Complexity
Semi-clustering	Scalable	Yes, it is made for handling large graphs in Pregel	Suitable for graph fuzzy clustering	Do not find membership of the vertices towards clusters
P.G. Sun et al.	Not Scalable	Yes, in the form of relation model	Finds both overlapping and non-overlapping clusters	Involves bulky matrix calculations
PFM	Scalable (Parallel implementation)	Yes, in the form of matrix only	Prior knowledge of the number of clusters is not required	Correlations are not uniformly distributed among data partitions
Nepusz et al.	Not Scalable	Yes, but it is efficient for small datasets only	Finds both fuzzy clusters and membership	Appropriate for small datasets only
Proposed PGFC	Scalable	Yes, it handles large graphs efficiently	Finds fuzzy clusters and node membership in distributed environment	Prior Knowledge about the number of clusters is required.

Algorithm 2 Pregel K-Means algorithm.

```
Input: An undirected graph G
1.  Function Compute (msg,superstep)
2.    if (this.getSuperstep() == 0) {
3.      // initialization using randomly selected cluster centers
4.      getValue(Initial_Centers)
5.    else
6.      VertexCompute() // Algorithm 3
7.    if (this.getSuperstep == 1)
8.      Initial_Centers= getaggregatedvalue(Initial_Centers)
9.      InitialCenters.get(cluster_count)
10.   else
11.     For each Message: msg do
12.       // compute the new clusters centers
13.       newClusterCenters[K]
14.  if (getSuperstep() > maxIteration | | (cluster_Position_Difference =0)
15.     printFinalClusters_Centers()
16.     haltComputation()
17.   else
18.     // update the aggregator with the new cluster centers
19.   setAggregatedValue(newCluster_Centers)
20.   Send msg(newCluster_Centers) to all the neighbours
21.  end function
Output: List of Vertices with the assigned cluster head
```

Algorithm 3 K-Means VertexCompute function.

```
1.  Function VertexCompute(Vertex<Long,Double,Float>, Message <Double>)
2.    ReadClusterCenter()
3.      For each ClusterCenter do
4.        Distance(ClusterCenter, vertices)
5.        If (Distance < minDistance)
6.          minDistance(clusterID)=Distance(ClusterID)
7.  // Asign vertices to cluster centers with minimum Distance
8.    Aggregate(Vertices, CenterID)
```

algorithm will run until the positions of the cluster centers stop varying.

The details of the computations performed by each worker node are shown in Algorithm 3. The VertexCompute() method is called on each worker. It receives the aggregated values of centroids from master node in the starting of the superstep. In each superstep, all the vertices assigned to workers recalculate their distance from all the centroids using the messages they received from neighbors in the previous superstep. Messages carry the minimum distance in this algorithm. The vertices are then aggregated with the centroid having the minimum distance from it.

4.2. Pregel based semi-clustering algorithm

It is a greedy algorithm commonly used for social graph processing representing users as vertices and the relationships or interactions among users as edges. A weighted undirected graph is taken as an input. The goal of the algorithm is to find at most C_{max} semi-clusters having V_{max} vertices. C_{max} and V_{max} are defined by user in the beginning. A score is assigned to semi-cluster c based on the following formula:

$$S_c = \frac{I_c - f_B B_c}{V_c(V_c - 1)/2} \tag{11}$$

where, I_c is the sum of the weights of all internal edges, B_c is the sum of the weights of boundary edges which are connected to the vertices of different semi-cluster, V_c is the number of vertices in the semi-cluster, and f_B is a user-defined boundary edge score factor whose value is between 0 and 1. Each vertex V keeps a score wise sorted list comprising at most C_{max} semi-clusters. Vertex V adds itself to the first semi-cluster of size 1 in the list with score 1 in the superstep 0. Then, the information is transmitted to all of its adjacent vertices.

In the next superstep, V iterates through all the semi-clusters in the list. If V is not there in any semi-cluster c_i, then V is added to c_i to form c'_i. The semi-clusters are then sorted according to scores, and the one's with best scores are sent to the adjacent vertices of V. Vertex V updates the list of all the semi-clusters containing V. The convergence is reached when there is no change in the semi-clusters or the maximum limit of supersteps is reached. The list of the top most semi-cluster candidates on each vertex are gathered and added to the global list of semi-clusters.

5. Design and implementation of the proposed PGFC algorithm

In this section, the key design of the proposed parallel and scalable fuzzy graph clustering algorithm 'PGFC' is given. The parallel and serial parts of Fuzzy C-Means (FCM) are formalized as vertex-Centric PGFC to perform in-memory computation in Giraph. PGFC assigns vertices partially to multiple clusters for finding overlapped partitions. The number of clusters should be pre-defined. The degree of membership of vertices for fuzzy clusters relies on the closeness of the vertex to the cluster heads. The membership value lies in the range [0–1].

The flowchart of the PGFC algorithm is shown in Fig. 2. The input graph is partitioned among worker nodes using hash practitioner of Giraph. In superstep 0, the cluster center and membership lists are initialized. All the vertices are now active with initial value zero.

An empty list of size K is created where K is the predefined number of cluster. The vertex with higher degree value has more number of edges incident on it, so is more important in comparison to other vertices of the network. All the vertices are sorted

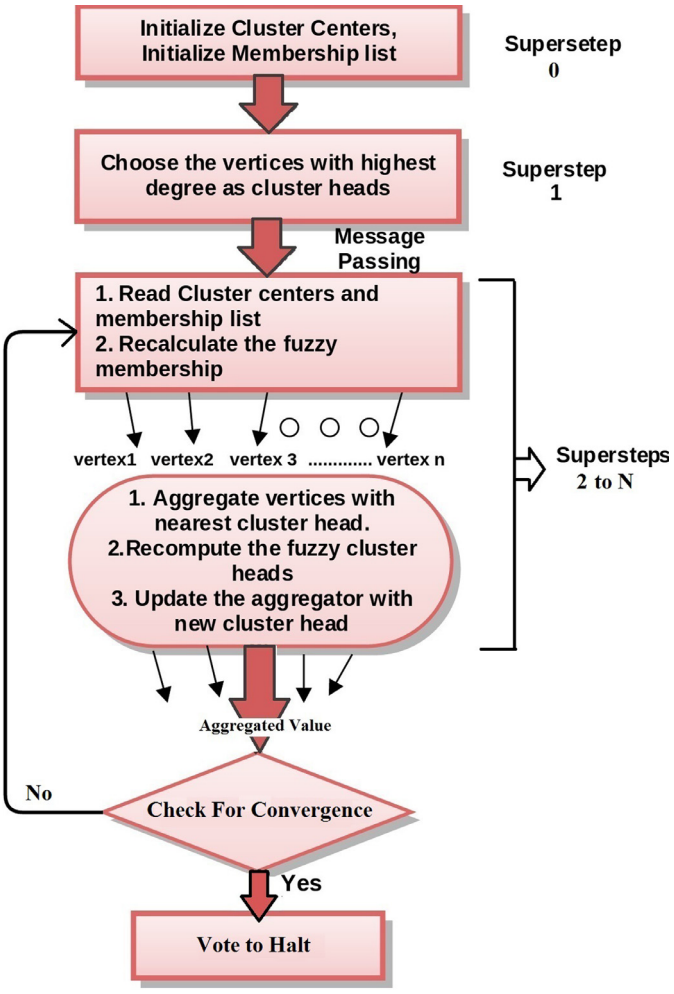

Fig. 2. Flow chart of PGFC algorithm.

Algorithm 4 PGFC algorithm.

```
Input: Graph G
1.  Function Compute(msg,superstep)
2.    If (this.getSuperstep() == 0) {
3.      initialize the cluster centers
4.      initialize the Membership_Matrix
5.    If (this.getSuperstep() ==1){
6.      initial clusters=vertexValue[k] // From Algorithm 5
7.        getValue(initial_centers)
8.        getValue(membership_matrix)
9.  Else
10.       VertexCompute() // From Algorithm 6
11.       For each Message: msg do
12.       // compute the new clusters centers using Eq. (9)
13.       newClusterCenters[K]
14. If (getSuperstep() > maxIteration | | J=min ||max_ij{μ_ij^(k+1) − μ_ij^k} <∈
15.       printFinalClusters_Centers()
16.       haltComputation()
17.  else
18.       // update the aggregator with the new cluster centers
19.       setAggregatedValue(newCluster_Centers)
20.       Send msg(newCluster_Centers, updatedMembershipMatrix) to all the
          neighbors
21. end function
Output: List of Vertices with the final membership values corresponding to final
        cluster centers
```

Algorithm 5 Node degree calculation.

```
1.  Function Compute ((MessageIterator* msgs)
2.       If (this.getSuperstep() == 0) {
3.         Edges=getEdges()
4.         For (edges->Next()){
5.           SendMessage(egde.getTargetVertexID()
6.       }
7.    If (this.getSuperstep() > 1) {
8.         Long sum =0;
9.         For (msgs->Next()){
10.          Sum++
11.        }
12.       vertexValue=getValue()
13.       vertexValue.set(sum)
14.       setValue(vertexValue)
15.       Mergesort(vertexValue, msgs)
16.    voteToHalt()}
17. End Function
```

in the decreasing order of their degree. FCM is sensitive to initialization of cluster centers. It selects cluster heads randomly during initialization which leads to uncertainty in the results obtained. Therefore, in PGFC, instead of choosing cluster heads randomly in beginning, the vertices with high degrees are selected as cluster heads. Top K vertices are selected from the degree list in Superstep 1 and are stored in the cluster head list. In further supersteps, master and worker communicates through messages to update cluster centers and membership values of each vertex corresponding to clusters iteratively, until the stopping criteria is met.

In distributed algorithms for graphs, the major cost is of network communication. One solution to minimize the communication cost is to minimize the inter cluster edges. But it will result in the incorrect results as a vertex can be part of more than one cluster. PGFC maintains the native graph structure and minimize communication overhead by aggregating individual messages into a single message from each node. The utility functions of Giraph like combiners and aggregators are used in order to reduce the message passing and global communication between the vertices.

The detailed process of the compute() method of PGFC is given in Algorithm 4. In the first superstep, master node initializes the cluster centers by selecting top K nodes with high degrees in the graph keeping in view that selected nodes are not adjacent. The membership list is also initialized. In further supersteps, the cluster centers are recomputed using Eq. (9) until the value of J mentioned in Eq. (7) is minimized or the convergence criteria mentioned in Eq. (10) is fulfilled.

5.1. Selection of initial clusters centers

Both FCM and K-means are sensitive to the selection of initial centers. The most simple centrality measure is the degree of the vertices in the graph. The degree of a node is the number of edges inclined on that node. In the proposed approach, we first calculate the degrees of all the vertices using a standard in-degree and out-degree algorithm in Giraph. Then, all the vertices are sorted in the descending order of their degree. The steps used in the calculation of degree of the vertices in graph using Pregel are shown in Algorithm 5. K vertices with highest range of degree are considered as the initial cluster heads. In our approach, we have taken an assumption that two cluster heads must not be the direct neighbor of each other. Thus, if two adjacent nodes are chosen as cluster head on the basis of degree, then the node with larger degree is considered as cluster head and the second one is simply ignored.

5.2. Finding membership values

Algorithm 6 shows the computation of fuzzy membership value at each worker node. In each superstep, workers receive the cluster center list from the master node, calculates the membership value of each vertex towards each cluster center using Eq. (8). The calculation is performed based on the updated potential minimum distance from the cluster center using the messages they receive from the neighbors. All the vertices pass their membership values

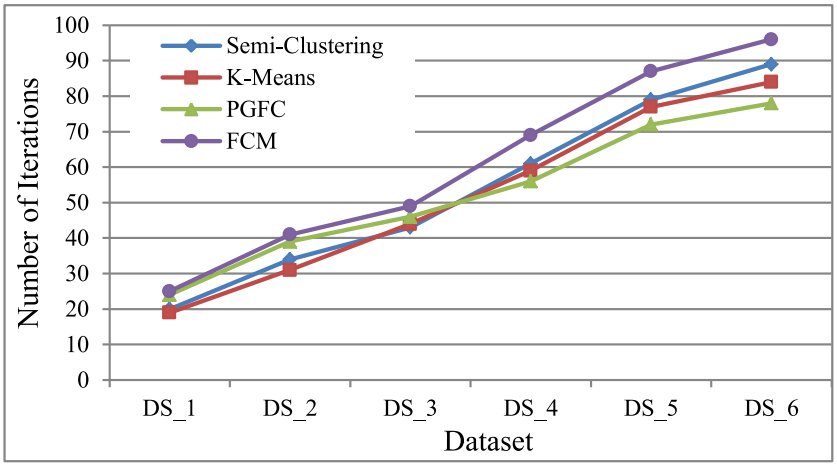

Fig. 3. Comparison in terms of number of iterations.

Algorithm 6 VertexCompute.

```
1. Function VertexCompute(Vertex<Long,Double,Float>, Message <Double>)
2.       ReadClusterCenter()
3.     For each ClusterCenter do
4.       // Calculate the Fuzzy Membership using Eq. (8)
5.         Membershipfunction (Distance, fuzzinessIndex)
6.     Membership = max (Membership[K])
7. Aggregate (Vertices, Membership)
```

Table 2
Description of data set used.

Data Code	Data Set Type	|V|	|E|	Size
DS_1	Synthetic	68,209	253,330	8MB
DS_2	Synthetic	165,220	502,872	20MB
DS_3	Synthetic	281,341	1,128,850	44MB
DS_4	LiveJournal	3,997,962	34,681,189	478MB
DS_5	Orkut	3,072,441	117,185,083	1.68GB
DS_6	Synthetic	10,412,300	124,876,078	2.7GB

to the aggregators, which are further used by compute() function at master node to compute the new cluster centers. After reaching the convergence, all the vertices vote to halt. Messages in this algorithm carry the potential shorter distances between the vertices and the cluster center.

6. Experimental evaluation

To evaluate the efficiency of the proposed algorithm experiments were performed on 12 dell machines, forming a Hadoop cluster each with 8GB of RAM, 1024GB hard disk, having Ubuntu Linux OS installed. All the machines were connected via a gigabyte network. One of the node was selected as master node, and rest were considered as slave nodes. We used Hadoop 2.6.0 and Giraph 1.1.0 for experiments.

We used six datasets of different sizes. Out of six, four datasets were generated synthetically using Graphgen tool. Also, two real world datasets taken from Stanford Large Network Dataset Collection (Leskovec & Andrej, 2014) namely LiveJournal and Orkut were used. The LiveJournal graph models the friendship relationships among online communities of users whereas, Orkut was a popular online social networking site. We named the six datasets as DS_1, DS_2, DS_3, DS_4, DS_5 and DS_6. The details are shown in Table 2.

The proposed PGFC algorithm is compared with the existing state of art algorithms namely K-Means and Semi-Clustering in Pregel. For evaluation, the number of cluster K is considered as 7, fuzzyfication index m is taken as 2 and the termination criteria ε as *le-3*.

Table 3
Cluster quality comparison in terms of partition coefficient and conductance.

Algorithm	Semi-Clustering		K-Means		FCM		PGFC	
Measure	*PC*	C_k	*PC*	C_k	*PC*	C_k	*PC*	C_k
DS_1	0.95	0.341	1.0	0.432	0.82	0.304	0.82	0.291
DS_2	0.96	0.393	1.0	0.491	0.82	0.243	0.79	0.207
DS_3	0.94	0.343	1.0	0.471	0.79	0.261	0.75	0.214
DS_4	0.90	0.411	1.0	0.572	0.78	0.292	0.71	0.269
DS_5	0.89	0.352	1.0	0.492	0.76	0.247	0.68	0.212
DS_6	0.88	0.374	1.0	0.433	0.79	0.283	0.71	0.248

6.1. Cluster quality

To measure the accuracy of the clustering decisions, we compared the Partition coefficient (*PC*) and conductance (C_k) of PGFC with the other state of art algorithms. The comparative values of the considered algorithms in terms of partition coefficient and conductance for various datasets are given in Table 3. For both quality measures, lower is the value better are the clustering results.

For K-Means, the value of partition coefficient has been always 1.0 because, it forms hard clusters. For DS_5 with around 3 million vertices, the Partition coefficient of PGFC is 0.68 in comparison to semi-clustering which has a value of 0.89. Thus, in terms of Partition Coefficient, the accuracy of PGFC is 23.5%, 32% and 10.6% higher than semi-clustering, K-Means and FCM respectively.

The proposed PGFC also performs better in terms of conductance in comparison to semi-clustering, K-Means and FCM algorithm. For largest dataset DS_6, the accuracy of PGFC in conductance is 33%, 42% and 12% higher than semi-clustering, K-Means and FCM respectively. Thus, the quality of the clusters produced by the proposed PGFC algorithm is better in comparison to semi-clustering and K-Means for all the datasets.

6.2. Number of iterations

The number of iterations before convergence of PGFC along with state of art algorithms is shown in Fig. 3. In both PGFC and FCM, the membership of each vertex is computed corresponding to each cluster, therefore more calculations are performed during each iteration. The results show that despite of having large

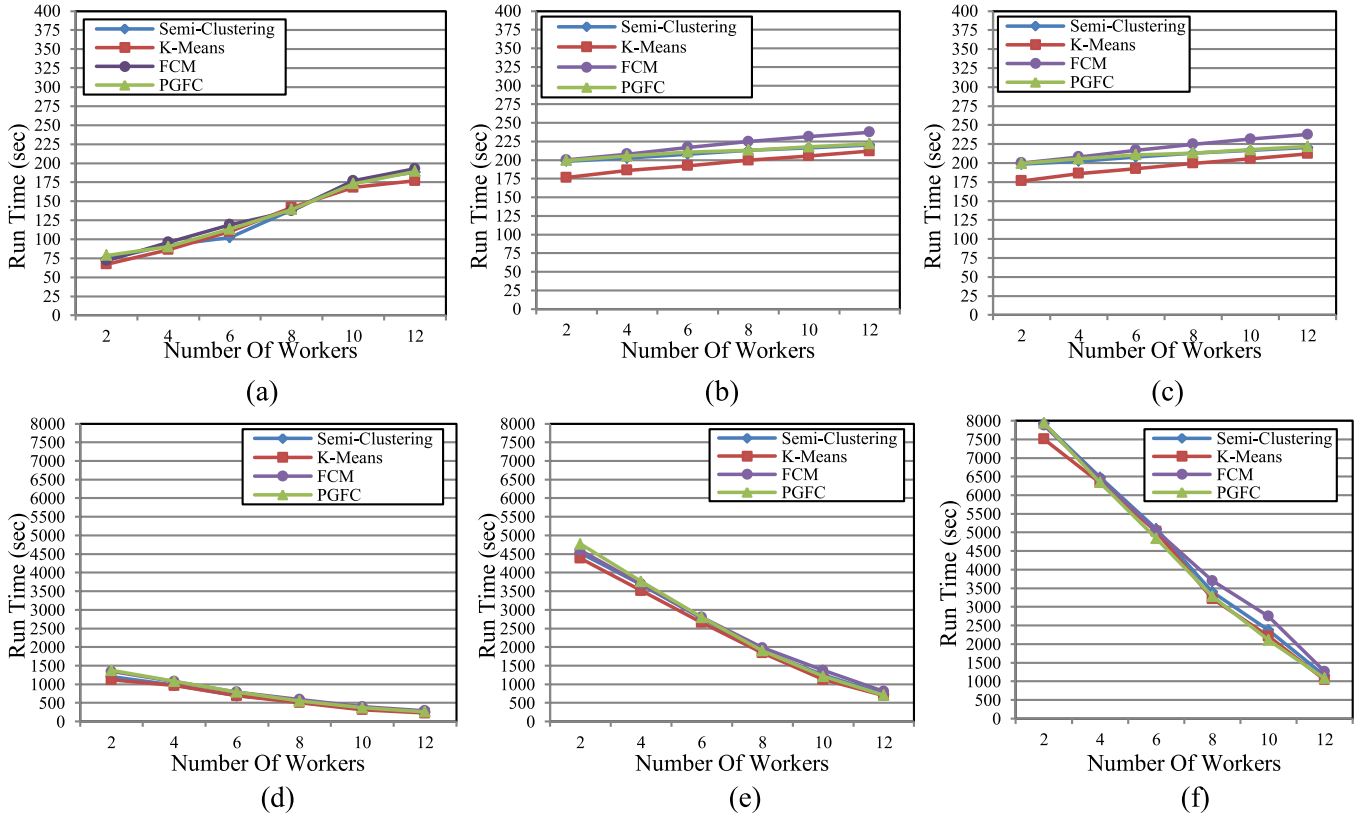

Fig. 4. Scalability: run time of clustering algorithms with different number of workers (a) DS_1, (b DS_2, (c) DS_3, (d) DS_4, (e) DS_5, (f) DS_6.

calculations, the proposed approach terminates in lesser number of iterations in comparison to the other algorithms for large datasets.

K-Means and FCM both are sensitive to cluster center initialization. Selecting cluster heads with high degrees for initialization in PGFC resulted in early convergence and thus, results are achieved in lesser iterations. For DS_5, the number of iterations for PGFC, K-Means, FCM and Semi-clustering are 72, 77, 87 and 79 respectively. Thus, PGFC has 6.4%, 17.2% and 8.8% lesser iterations than K-Means, FCM and Semi-clustering respectively.

6.3. 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 algorithm in this paper includes the time of loading the graph to Giraph and the time to run that algorithm. In Fig. 4, run time of semi-clustering, K-Means, FCM and proposed PGFC algorithm are shown. PGFC, FCM and Semi-clustering are finding the overlapping clusters, whereas K-Means is finding the hard clusters. Even though, K-Means and PGFC are similar in approaches. There is the difference in the way of initialization and in additional calculation of finding weighted associations with more than one cluster. K-Means is just performing distance calculation, whereas PGFC needs to find the degrees of all the vertices and then performing full inverse-distance weighting.

From the Fig. 4,a linear run time is observed for all the algorithms when the numbers of worker nodes are increased. For large graph datasets DS_5 and D_6, the difference between run time of K-Means and PGFC is only 1.6% and 3.2% respectively. Whereas, in comparison to overlapping detection algorithm Semi-Clustering, PGFC is 2.7% and 6.5% faster for DS_5 and DS_6 respectively. Also, PGFC is 12.8% and 14% faster its inherent algorithm FCM for DS_5 and DS_6 respectively.

6.4. Speedup behavior

To test the scalability, Speedup is used which measures the efficiency of the parallel algorithm when the number of processors are increased. It is calculated as:

$$Speedup = \frac{Run\_Time_1}{P * Run\_Time_P} \tag{12}$$

where, P is the number of processors used. For ideal parallelization, $Speedup = 1$. Speedup of PGFC is tested by varying the number of available workers P in Giraph job.Speedup behavior of PGFC along with other three algorithms is shown in Fig. 5 for three large graph datasets DS_4, DS_5 and DS_6. Here, higher bar indicates better scalability.

As anticipated, increase in the number of processors results in decreased run time. As shown, PGFC is highly scalable with speedup of 0.62, whereas K-Means, FCM and Semi-clustering have speedup of 0.59, 0.51 and 0.57 respectively for DS_6.

6.5. CPU utilization

CPU usage is examined to see how much percentage of CPU is used to complete the computation. It is better to keep the system busy by performing additional important tasks rather than keeping it idle. Both PGFC and FCM find the fuzzy clusters as well as membership values of each vertex towards the clusters. In addition, PGFC calculates the degrees of each vertex for initialization of cluster heads. Whereas, K-Means detects the non-overlapping clusters only. PGFC keeps the CPU busy in performing calculations, utilizing it better than the other algorithms as shown in Fig. 6.

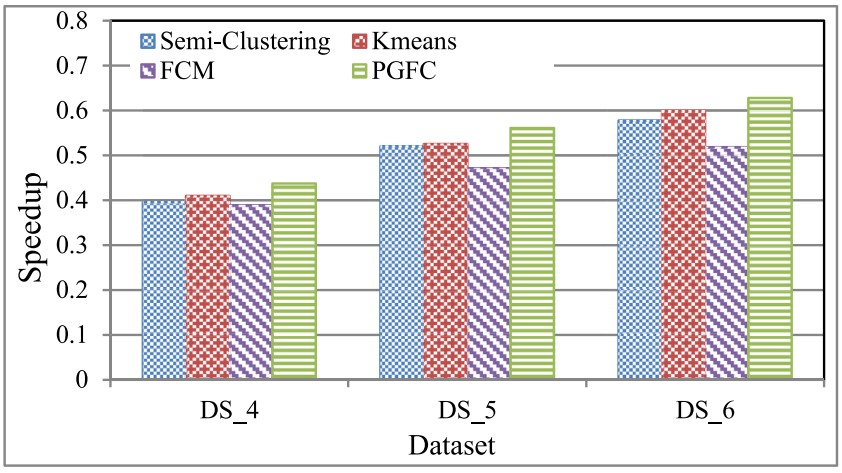

Fig. 5. Speedup behavior of clustering algorithms- higher bar indicates better scalability.

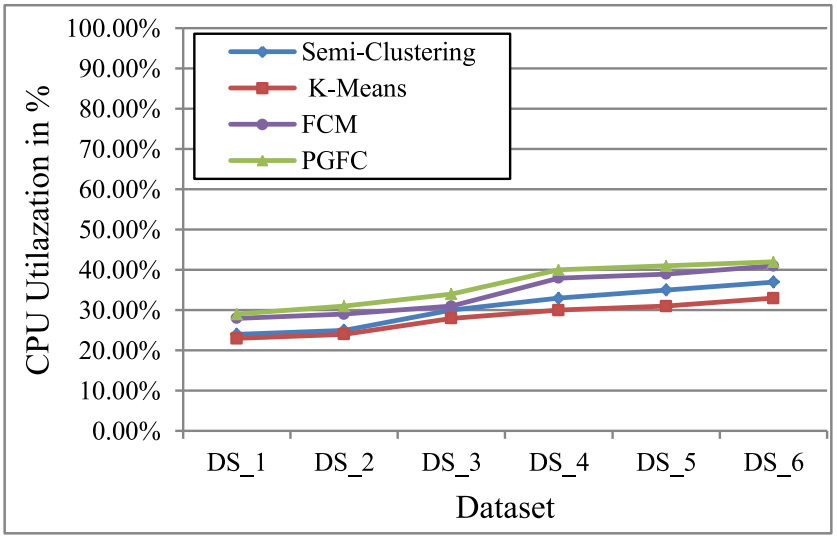

Fig. 6. CPU usage comparison.

7. Conclusion and future work

In this paper, a parallel Fuzzy graph clustering algorithm 'PGFC' is designed which allow overlapping among cluster using Pregel. It is designed especially for large graph data by improving the traditional Fuzzy C-Mean(FCM) clustering algorithm. The scalable and open source implementation of PGFC on Giraph makes it easy to use on any commodity system cluster. The performance of the proposed algorithm is compared with Pregel based semi-clustering, K-Means, FCM by applying it to synthetic and real-life graphs in a distributed environment. It is observed that PGFC show better performance for fuzzy clustering in terms of partition coefficient and conductance. Our algorithm finds the overlapping clusters more accurately in comparison to other state of art algorithms. PGFC takes the considerable benefit from the degree centrality criteria during initialization and converges in lesser number of iterations. Despite of performing additional job of finding membership values, optimum clustering results are achieved by PGFC in minimal time and with high CPU utilization. It is found that in distributed environments, PGFC can scale up well like the other existing algorithms to handle large graphs with high speedup.

In future, the work will be extended for designing a new model able to perform both hard and soft clustering according to the specifications of the input data. The proposed algorithm will be performed using more processors on larger graphs having billions of nodes for better scalability analysis. The total execution time will also be improved by reducing the overhead needed in performing full inverse-distance weighting calculations. Also, a new approach for finding the number of clusters can be incorporated by analyzing the topology of the graph to overcome the requirement of predefining the number of clusters in PGFC.

References

Arthur, D., & Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms* (pp. 1027–1035).

Bahmani, B., Moseley, B., Vattani, A., Kumar, R., & Vassilvitskii, S. (2012). Scalable k-means++. *Proceedings of the VLDB Endowment, 5*(7), 622–633.

Bezdek, J. C., Ehrlich, R., & Full, W. (1984). FCM: The fuzzy C-means clustering algorithm. *Computers & Geosciences, 10*(2), 191–203.

Chen, X. (2015). A new clustering algorithm based on near neighbor influence. *Expert Systems with Applications, 42*, 7746–7758.

Cook, D. J., & Holder, L. B. (2006). *Mining graph data*. John Wiley & Sons.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM, 51*(1), 107–113.

Dunn, J. C. (1973). A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact WellSeparated Clusters. *Cybernetics and Systems, 3*(3), 32–57.

Egrioglu, E., Aladag, C. H., & Yolcu, U. (2013). Fuzzy time series forecasting with a novel hybrid approach combining fuzzy c-means and neural networks. *Expert Systems with Applications, 40*(3), 854–857.

Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A. Y., et al. (2014). A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE Transactions on Emerging Topics in Computing, 2*(3), 267–279.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports, 486*(3), 75–174.

Golsefid, S. M. M., Zarandi, M. H. F., & Bastani, S. (2015). Fuzzy community detection model in social networks. *International Journal of Intelligent Systems, 30*(12), 1227–1244.

Kang, U., & Faloutsos, C. (2013). Big graph mining: Algorithms and discoveries. *ACM SIGKDD Explorations Newsletter, 14*(2), 29–36.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(7), 881–892.

Kesemen, O., Tezel, Ö., & Özkul, E. (2016). Fuzzy c-means clustering algorithm for directional data (FCM4DD). *Expert Systems with Applications, 58*, 76–82.

Khalilia, M. A., Bezdek, J., Popescu, M., & Keller, J. M. (2014). Improvements to the relational fuzzy C-means clustering algorithm. *Pattern Recognition, 47*, 3920–3930.

Király, A., Vathy-fogarassy, Á., & Abonyi, J. (2015). Geodesic distance based fuzzy c-medoid clustering – searching for central points in graphs and high dimensional data. *Fuzzy Sets and Systems, 1*, 1–16.

Kurland, O., & Lee, L. (2004). Corpus structure, language models, and ad hoc information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on research and development in Information retrieval* (pp. 194–201).

Leskovec, J., & Andrej, K. (2014). Stanford Large Network Dataset Collection. https://snap.stanford.edu/data/.

Ludwig, S. A. (2015). MapReduce-based fuzzy c-means clustering algorithm: Implementation and scalability. *International Journal of Machine learning and Cybernetics, 6*(6), 923–934.

Malek, S., Golsefid, M., Hossien, M., & Zarandi, F. (2015). Fuzzy community detection model in social networks. *International Journal of Intelligent Systems, 30*, 1227–1244.

Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., & Horn, I. (2010). Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD international conference on management of data* (pp. 135–146).

Nepusz, T., Petrczi, A., Ngyessy, L., & Bazs, F. (2008). Fuzzy Communities and the concept of Bridgeness in Complex Networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics, 77*(1), 1–13.

Nepusz, T., Yu, H., & Paccanaro, A. (2012). Detecting overlapping protein complexes in protein-protein interaction networks. *Nature methods, 9*(5), 471–472.

Opsahl, T., Agneessens, F., & Skvoretz, J. (2010). Node centrality in weighted networks : Generalizing degree and shortest paths. *Social Networks, 32*, 245–251.

Özbay, Y., Ceylan, R., & Karlik, B. (2006). A fuzzy clustering neural network architecture for classification of ECG arrhythmias. *Computers in Biology and Medicine, 36*(4), 376–388.

Rahimi, S., Zargham, M., Thakre, A., & Chhillar, D. (2004). A parallel fuzzy C-mean algorithm for image segmentation. In *Proceeding of IEEE annual meeting of the fuzzy information processing (NAFIPS'04): 1* (pp. 234–237).

Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review, 1*(1), 27–64.

Son, L. H., Cuong, B. C., Lanzi, P. L., & Thong, N. T. (2012). A novel intuitionistic fuzzy clustering method for geo-demographic analysis. *Expert Systems with Applications, 39*(10), 9848–9859.

Son, L. H. (2015). DPFCM: A novel distributed picture fuzzy clustering method on picture fuzzy sets. *Expert Systems with Applications, 41*, 51–66.

Su, C., Wang, Y. K., & Yu, Y. (2014). Community detection in social networks. *Applied Mechanics and Materials, 496*, 2174–2177.

Sun, P. G., Gao, L., & Han, S. S. (2011). Identification of overlapping and non-overlapping community structure by fuzzy clustering in complex networks. *Information Science, 181*(6), 1060–1071.

Timón, I., Soto, J., Pérez-sánchez, H., & Cecilia, J. M. (2016). Parallel implementation of fuzzy minimals clustering algorithm R. *Expert Systems with Applications, 48*, 35–41.

Vehlow, C., Member, S., Reinhardt, T., Weiskopf, D., & Society, I. C. (2013). Visualizing fuzzy overlapping communities in networks. *IEEE Transactions on Visualization and Computer Graphics, 19*(12), 2486–2495.

Wang, G., Hao, J., Ma, J., & Huang, L. (2010). A new approach to intrusion detection using artificial neural networks and fuzzy clustering. *Expert Systems with Applications, 37*(9), 6225–6232.

Wang, Y., Ma, X., Lao, Y., & Wang, Y. (2014). A fuzzy-based customer clustering approach with hierarchical structure for logistics network optimization. *Expert Systems with Applications, 41*(2), 521–534.

Wikaisuksakul, S. (2014). A multi-objective genetic algorithm with fuzzy c-means for automatic data clustering. *Applied Soft Computing, 24*, 679–691.

Xu, Y., Qu, W., Li, Z., Min, G., Li, K., Member, S., et al. (2014). Efficient K-Means ++ approximation with MapReduce. *IEEE Transactions on Parallel and Distributed Systems, 25*(12), 3135–3144.

Zhao, W., Ma, H., & He, Q. (2009). Parallel k-means clustering based on Mapreduce. *Cloud Computing, Lecture Notes in Computer Science, Chapter IV, 5931*, 674–679.

Zhou, J., Member, S., Chen, C. L. P., & Chen, L. (2014). A collaborative fuzzy clustering algorithm in distributed network environments. *IEEE Transactions on Fuzzy Systems, 22*(6), 1443–1456.