International Journal of Engineering & Technology, 7 (4.5) (2018) 79-86

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Distributed Community Detection based on Apache Spark using Multi Label Propagation for Digital Social Networks

Satya Keerthi Gorripati[1]*, Valli Kumari Vatsavayi [2]

[1]Department of IT, GMR Institute of Technology, Rajam, Andhra Pradesh, India
[2]Department of Computer Science and Systems Engineering, Andhra University, Visakhapatnam,Andhra Pradesh, India
**Corresponding author E-mail satyakeerthi.gsk@gmail.com*

Abstract

Organization, Government and Individual (OGI) have popularized the use of Digital Social Networks (DSN) that reduces the processing time of social-aware tasks. To accomplish a community-based communication, each social-aware task should identify its community group. The identified group uses a task to avail all the DSN benefits to their customers / citizens. As a result, the community-based detection algorithm has played a significant role in literature. However, the existing algorithms have had several challenging issues, such as performance and scalability. Thus, a distributed community detection algorithm is presented using Apache Spark's Resilient Distributed Data Set (RDD) framework based on the Scala programming language. The Apache Spark framework provides an ideal solution that offers ease of coding, performance, interactive mode and disk Input-Output bottlenecks in Hadoop /Map Reduce. Besides, it presents a platform of distributed community detection that reduces the computational computation by applying transformations, aggregations and joins. The experimental results show that the proposed framework achieves high accuracy for both real-world and synthetic networks.

Keywords: *Apache Spark ;Community detection, Distributed; RDD ; Social graphs.*

1. Introduction

E-government uses Information Communication and Technologies (ICTs) to satisfy the citizen goals. For public trustworthy, the government has provided the technology, transparency that enhances accountability and citizen empowerment [1]. Social media service is employed to provide the interactive communication, content-sharing and opinion expression. It helps to collect the necessary information about the communities and the citizen backgrounds. As a result, the government publicizes the social media as a direct-communicator to disseminate the public welfare schemes. Since it is massively considered across various e-government agencies namely government-to-citizens (G2C), government-to-business (G2B) and government-to-government (G2G), the improvement of online participation and classification is highly demanded.
A plausible indicator shows that the online government services have reported 39% in 2005 and 57% in 2011 [2]. The creative roles are proven in various countries in terms of political aspects such as Iran, Egypt, Tunisia and Arab spring. Among these countries, the Arab spring completely changes the shape of political discourse by the influence of social media. It offers a tactical breakthrough to manage and engage the individuals, businesses and public organization cautiously [3]. Moreover, the social media benefits the government in terms of transparency to ensure the citizen with more innovative services and information access to open an interactive communication. Complex social networks (Facebook and Twitter), biological networks (transcriptional regulatory networks and virus-host networks) and technology networks (electric grids) exhibit the features namely high clustering coefficient, a heavy-tail degree distribution, hierarchical structure and presence of communities. A high clustering coefficient is a measure of the degree to which nodes in a graph tend to be clustered (or community) together. A heavy tail degree distribution is also known as scale-free or power-law distribution that has a distribution "heavier" than the exponential distribution. Communities or groups in a network hold denser connections (or relationships) inside the groups and have the sparser connections outside the groups. However, there is no formal community definition accepted universally. In most of the community detection algorithms, the communities or groups are represented as the final products.

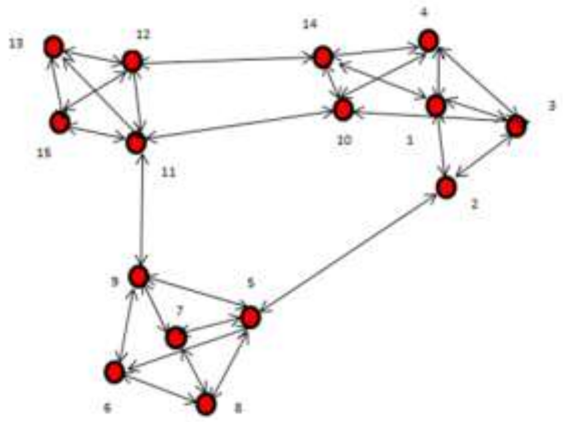


Fig.1: A Network with 3 communities

A network or graph is an abstract representation of nodes and their connections respectively. A basic network arrives when the similar nodes are connected. A social network is defined as a network that has a node connector as an actor in order to provide a relationship such as co-authorship, friend, ownership, relationship of beliefs, etc. From Figure 1, it is observed that the network has three communities with the nodes such as $C_1 \rightarrow \{5,6,7,8,9\}$, $C_2 \rightarrow \{11,12,13,15\}$ and $C_3 \rightarrow \{1,2,3,4,10,14\}$. For the given graph, dense and sparse connection is used to find the communities. In C_1, there are eight edges inside the community whereas only two edges outside the community. While separating the outside edges, the network is partitioned into two sub-networks. Similarly, while continuing the partition process, the network is bifurcated into three sub-networks or communities. Girvan-Newman algorithm [4] is used to find the sparse edges with high betweenness that is removed to partition a network whereas Louvian technique[5] uses a benefit function (modularity) to search the best possible partition of the network and Clique based methods[6] searches strongly connected edge components to bifurcate a network.

As social networks such as Facebook, Twitter, LinkedIn etc. have had several edges have challenging issues namely convergence problems and sparse matrix representations are addressed. Besides, finding triangles, cliques and other related properties (clustering coefficient) are relatively expensive and time consuming as well for the use of billion-scale graphs. Zeng et al [8] introduced a distributed computation that improves the performance of sequential approaches. Thus, this paper proposes a Parallel Label Propagation Algorithm (PLPA) that is based on parallel community detection using the Spark RDD framework.

The paper is rationalized into four sections. Second section deals with the review of existing traditional and parallel community algorithms; Next section introduces our proposed PLPA approach in detail. Last section presents output results followed by conclusion.

2. Background

The existing community detection methods suffer from the following problems. The first one is that the algorithms work sequentially on small data sets and the efficiency of the algorithm decreases when the size of datasets increases and has time complexity O(n), where n is the number of edges (or links) in the graph. The second one is that in many algorithms graph was modeled as an adjacency matrix (or edge list, sparse matrix) and during computation at every phase the adjacency entries may become zero and leaves most of the entries in the matrix as sparse. Sequential algorithm cannot skip these sparse entries in the matrix. Therefore, to handle the above mentioned problems parallel algorithms are suggested in this paper.

2.1 Traditional Community Algorithms

Community detection algorithms are classified into model-based, local optimized and label propagation algorithms. Numerous problems were reported related to time-efficiency in the first two categories of algorithms and the third is the most cited one i.e., label propagation methods such as LPA[9], SLPA [10,11], BMLPA[12] MLPA[13], DMLPA[14], LabelRank[15], LabelRankT[16] provides properties like low computational complexity, ease of implementation and accuracy in both disjoint and overlapping community detection. The mutual characteristic feature in the label propagation family is the process of exchanging community labels during propagation process between graph nodes in order to send the updated messages to neighbor nodes.

The following are the stages in the label propagation method:

- Load graph
- Speaker Approach: draw l labels
- Listener Approach: store the most frequent l labels
- Post-processing

It is worth noticing that at every stage each node has a memory (entry in a matrix) to keep the account information observed in the past iteration. For loading graph the algorithm requires O(n), for the speaker and listener approaches it requires O(Tm) and for the post-processing it requires the complexity O(n). Therefore the overall complexity is O(Tm) of the entire algorithm.

2.2 Parallel Community Algorithms

Parallel computation has become a desirable and important feature of community detection due to increasing size of graphs. Known algorithms to increase scalability and performance include: LPA using Hadoop MapReduce[17] ,SLPA using OpenMP API[18,19] the Louvian method using Apache Giraph [20], Scalable Community Detection[21],Graph processing using Graphx[22] EgoLP[23] and many others.

3. Proposed PLPA Framework

The Label propagation community algorithms iterates over list of nodes and picks one of its neighbor as a listener and each neighbor of the selected node sends a frequent occurred label list by following speaking rule. Listener accepts one label from the list of labels from neighbors by following listening rule. This process iterates for all the nodes in the graph. After all iterations are completed, post processing is carried out to extract the communities.

It is obvious that the computation of community labels is a sequence of iterations makes the algorithm sequential and updated labels are accountable for subsequent iterations. Therefore, the nodes cannot be processed independent of each other. Parallel (or distributed) approach supports interactive and repetitive applications with the help of uniform data abstraction called RDD and it is a replacement of MapReduce approaches [24].

Label Propagation community algorithm (SLPA) performed well in constant networks to find the disjoint/overlapping communities. In SLPA, at each phase, speaker sends a label that has a high probability and listener listens to all labels send by the speaker nodes but accepts only one label that has maximum number of occurrences. However, due to random selection of labels at listener stage the algorithm produce different communities in different runs. Many revisions were made to resolve this randomness issue. Our proposed PMLPA a parallel version of community detection methods (based on SLPA, MLPA) is developed using Apache Spark framework. It alters speaker, listener rules by passing multiple labels during propagation process and the inflation[25] operator in post processing phase in order to detect the communities quickly.

3.1 Implementation

3.1.1 Load Graph

The edge list data set containing edges of a graph from a TEXT FILE is loaded in Spark RDD to make use of fault tolerance in cluster and also to provide efficient use of cluster memory.

Efficiency is achieved through parallelization of processing a file on multiple nodes in the cluster. The file is broadcasted to every worker node to make it available to each worker. Once the data is loaded into an RDD we can carry out transformations and actions. flatMap transformation is applied on the dataset to read the edges, which converts each edge into (source, destination) i.e., (key, value) pairs. A function countByKey is applied to get the degree of every node and groupByKey function is performed to get the neighbors of each node. Finally mapByValues() is applied on RDD to get the probability distribution matrix.

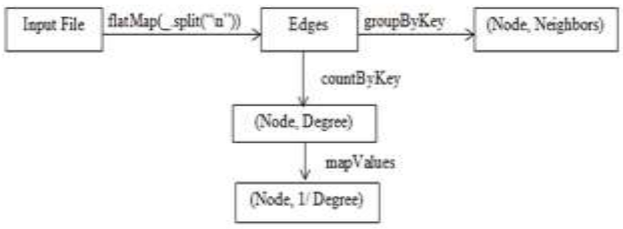
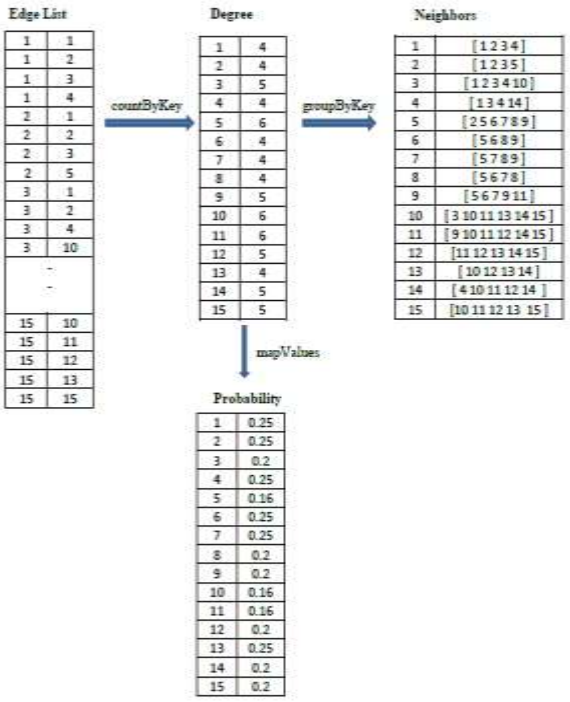


Fig 2:. Subgraph-centric Large-Scale Graph Analytics on Spark lineage graph for RDD of First Phase [26]

Algorithm

```
1.    Foreach edge E in $R_i$
2.       flatMap ( lineoffset , T )
3.           Foreach edge  e  in  E
4.               Yield (source , destination) edge pair
5.           End foreach
6.        End flatMap
7.      End foreach
8.       groupBykey(K,V)
9.         Foreach(K,V) pair
10.            Yield(K,Iteratable<V> ) pairs
11.                End foreach
12.        End groupBy
13.        countByKey(K,V)
14.                Foreach (K,V) pair
15.                  Yield(K,Count) pair
16.            End foreach
17.        mapValues(K,Count)
18.            Foreach(K)
19.                Yield(K,K=> 1.0 / V)
20.                    End foreach
21.    End mapValues
```



Edge List

1	1
1	2
1	3
1	4
2	1
2	2
2	3
2	5
3	1
3	2
3	4
3	10
-	
-	
15	10
15	11
15	12
15	13
15	15

Degree

1	4
2	4
3	5
4	4
5	6
6	4
7	4
8	4
9	5
10	6
11	6
12	5
13	4
14	5
15	5

Neighbors

1	[1 2 3 4]
2	[1 2 3 5]
3	[1 2 3 4 10]
4	[1 3 4 14]
5	[2 5 6 7 8 9]
6	[5 6 8 9]
7	[5 7 8 9]
8	[5 6 7 8]
9	[5 6 7 9 11]
10	[3 10 11 13 14 15]
11	[9 10 11 12 14 15]
12	[11 12 13 14 15]
13	[10 12 13 14]
14	[4 10 11 12 14]
15	[10 11 12 13 15]

Probability

1	0.25
2	0.25
3	0.2
4	0.25
5	0.16
6	0.25
7	0.25
8	0.2
9	0.2
10	0.16
11	0.16
12	0.2
13	0.25
14	0.2
15	0.2

Fig 3:. An example of the spark dataflow operators needed to load a graph (pre-processing). Input is an edge list; a countByKey operation can be performed to get the degree of every node. Then a groupByKey to expand the neighbors of every node finally mapValues to get the probability of every node in the network

3.1.2 Speaker Approach

Each node in the network starts off as its own community label (i.e., node's id). This means that initially each node belongs to a different community. The following steps repeated until the exit condition is satisfied.

a. Select any node as the listener.

b. The neighbor nodes of the listener become speakers, usually more than one speaker exists and their roles may get exchanged depend on whether a node obliges as information consumer or provider.

c. Apply the speaker rule and get the labels list for propagation.

The major functionality of speaker is to decide which labels list should be forwarded to the listener for propagation. The following are the rules proposed for this purpose at speaker side.

Rule 1: For a node *i* if there are $l_1,l_2,l_3,\ldots..l_n$ speakers exist and if all labels have the same frequency, then all the labels are sent to the listener.

Rule 2: For a node *i* if there are $l_1,l_2,l_2,l_3,l_3\ldots..l_n$ speakers exist and the labels l_2,l_3 have the same frequency of occurrence, then the label list containing the labels l_2,l_3 is sent to the listener.

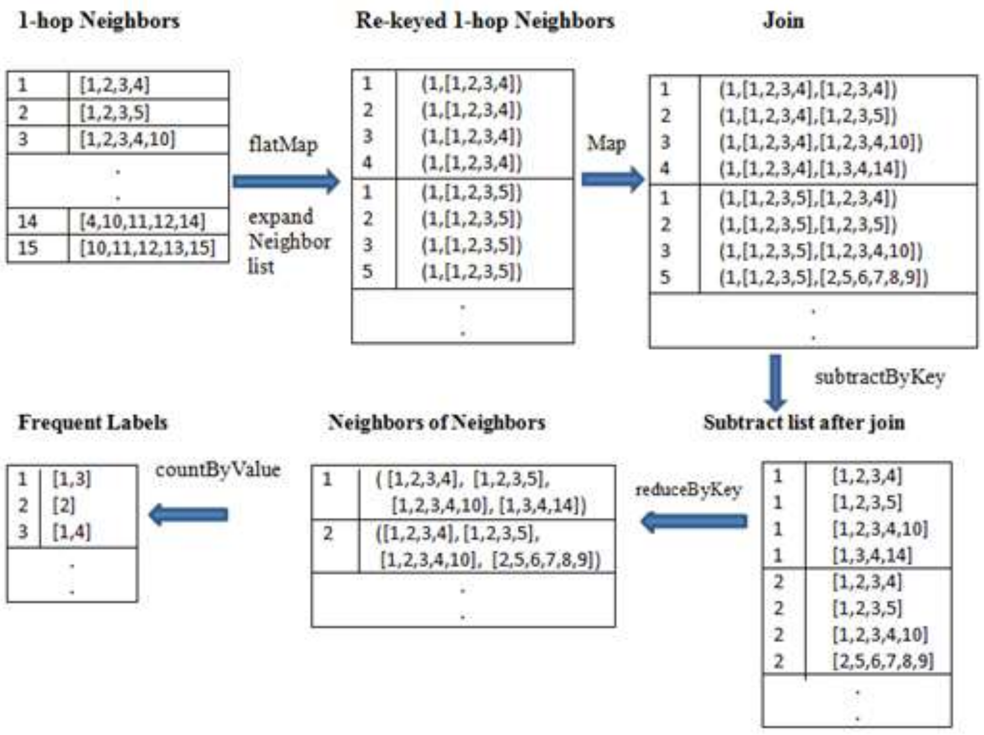


Fig 4.: An example of the spark dataflow operators needed to apply to propagate the labels to the listener. Assuming node 1 as listener, the speaker rules applied on each neighbor of the first node for sending label list to listener.

3.1.3 Listener Approach

Label propagation always starts with a listener. The principal task performed by the listener is to decide which labels are to be processed and which labels to be omitted? If a node *i* contains *J* speakers and if the number of labels sends by *J* nodes to node *i* is stored in label list *l* then, to determine what labels should be considered is based on following rules:

Rule 1: If the label names in the list *l* are unique, i.e. all labels have equal occurrence, then, the label list *l* should sent for updating process.

Rule 2: If the label names in the list *l* are not unique, i.e. all labels have differences of occurrences, then, the following rules are applied

i. If one label has maximum frequency than the other labels, then only the label which has maximum frequency should be considered for updating process and all others should be ignored.

ii. Assuming l_2,l_3 are the labels in the label set *l*, and if both has equal number of occurrences than the other labels, both labels are considered

After identifying the labels that have maximum number of occurrences, listener calculates the probabilities of accepted labels and passes the labels for the next phase and updates the label distribution matrix accordingly

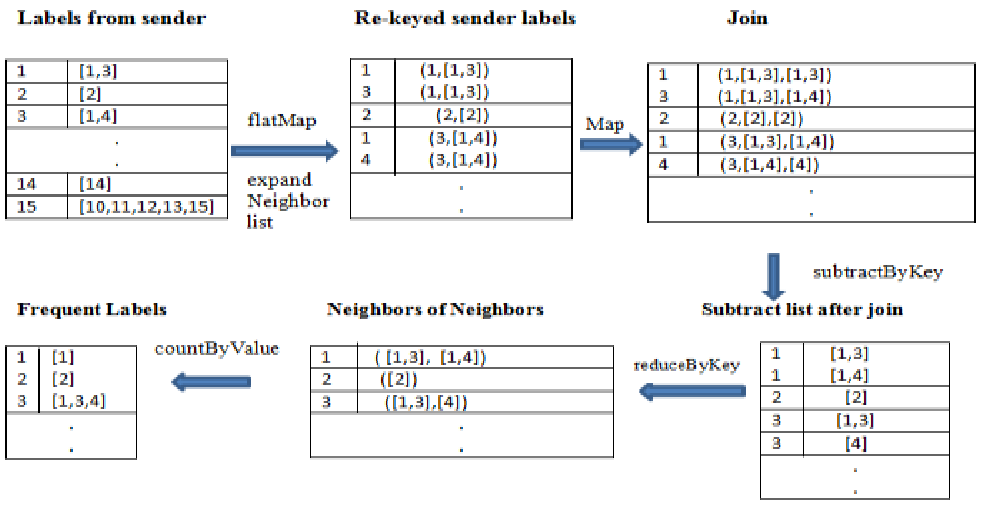


Fig 5:. An example of the spark dataflow operators needed to get the frequent observed label/s of every listener in the current step

3.1.4 Inflation

The inflation operator is used to increase the importance of labels that has maximum probability and to decrease the importance of labels that has least probability at the end of every iteration.

$$\Gamma_{in} P_i(C) = \frac{P_i(C)^{in}}{\sum_{j \in C} P_i(j)^{in}} \quad (1)$$

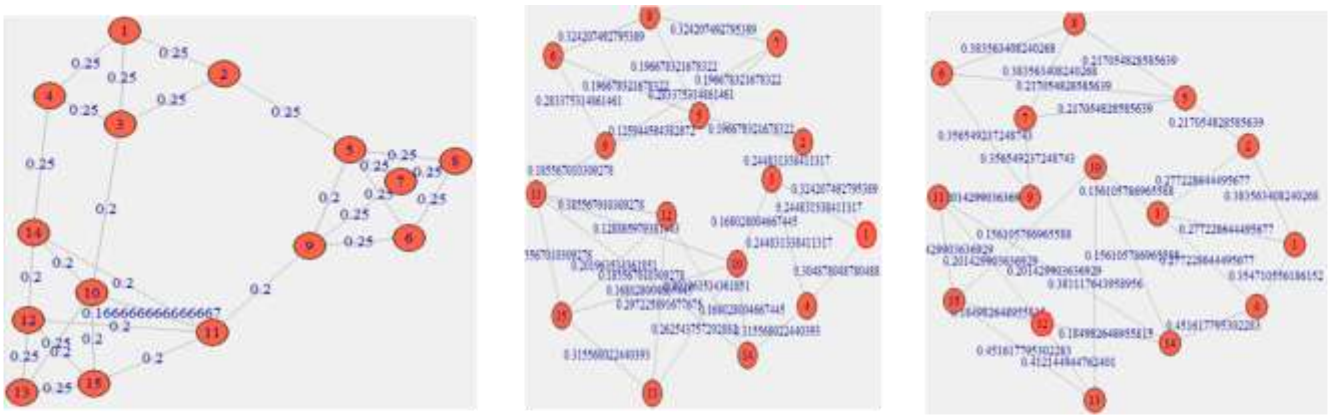
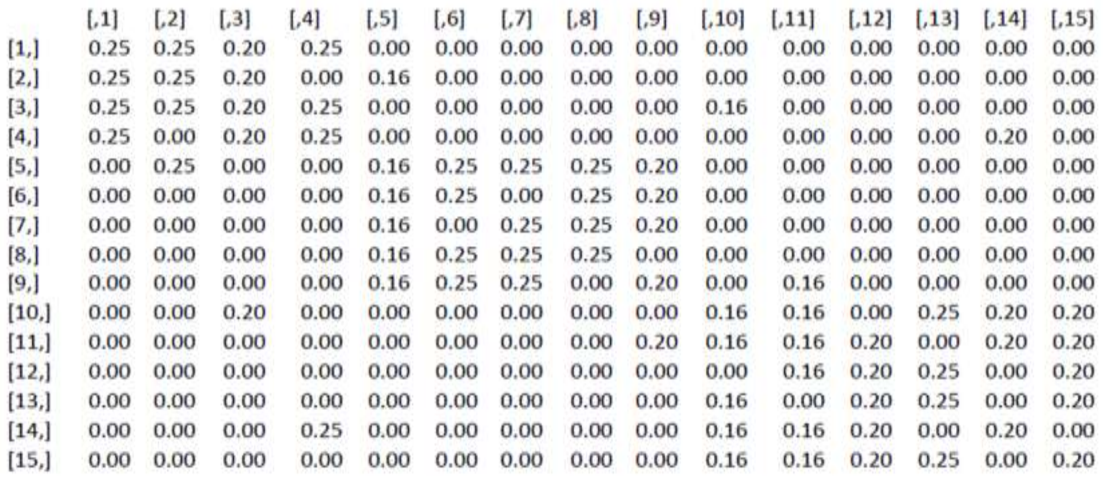

(a) (b) $\Gamma_{in} = 2$ (c) $\Gamma_{in} = 4$

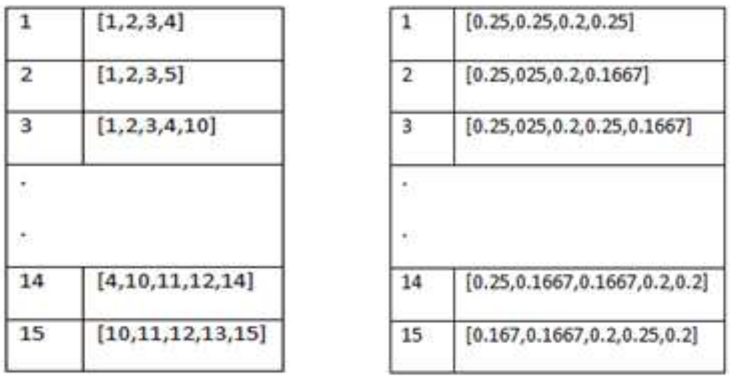
Fig 6: (a) sample network with the probability values as edge labels. (b) Network when applied inflation of 2. (c) Network when applied inflation of 4.

Traditional algorithms represent network structure P in the form of matrix, but when the network grows the matrix size rapidly increases and most of the entries become sparse as shown in Figure 7. Apache Spark RDD is a solution to overcome for the above said problem, in the proposed framework the label distribution matrix is represented as a RDD shown in Figure 8. Every node is a key and its speakers' probabilities are values (i.e., key with multiple values) inflation is applied to label distribution P, which initially holds the probabilities of all nodes and it continually updates the probabilities of accepted labels at listener phase. RDD shown in Figure 8 has a same resemblance as of a matrix as in Figure 7, which avoids the sparse entries.



	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]	[,11]	[,12]	[,13]	[,14]	[,15]
[1,]	0.25	0.25	0.20	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
[2,]	0.25	0.25	0.20	0.00	0.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
[3,]	0.25	0.25	0.20	0.25	0.00	0.00	0.00	0.00	0.00	0.16	0.00	0.00	0.00	0.00	0.00
[4,]	0.25	0.00	0.20	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.00
[5,]	0.00	0.25	0.00	0.00	0.16	0.25	0.25	0.25	0.20	0.00	0.00	0.00	0.00	0.00	0.00
[6,]	0.00	0.00	0.00	0.00	0.16	0.25	0.00	0.25	0.20	0.00	0.00	0.00	0.00	0.00	0.00
[7,]	0.00	0.00	0.00	0.00	0.16	0.00	0.25	0.25	0.20	0.00	0.00	0.00	0.00	0.00	0.00
[8,]	0.00	0.00	0.00	0.00	0.16	0.25	0.25	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00
[9,]	0.00	0.00	0.00	0.00	0.16	0.25	0.25	0.00	0.20	0.00	0.16	0.00	0.00	0.00	0.00
[10,]	0.00	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.16	0.00	0.25	0.20	0.20
[11,]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.16	0.16	0.20	0.00	0.20	0.20
[12,]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.20	0.25	0.00	0.20
[13,]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.00	0.20	0.25	0.00	0.20
[14,]	0.00	0.00	0.00	0.25	0.00	0.00	0.00	0.00	0.00	0.16	0.16	0.20	0.00	0.20	0.00
[15,]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.16	0.20	0.25	0.00	0.20

Fig7: The label distribution matrix of network size 15



1	[1,2,3,4]
2	[1,2,3,5]
3	[1,2,3,4,10]
.	
.	
14	[4,10,11,12,14]
15	[10,11,12,13,15]

1	[0.25,0.25,0.2,0.25]
2	[0.25,025,0.2,0.1667]
3	[0.25,025,0.2,0.25,0.1667]
.	
.	
14	[0.25,0.1667,0.1667,0.2,0.2]
15	[0.167,0.1667,0.2,0.25,0.2]

Fig8: The Lineage graph for RDDs of neighbors and label distribution of network size 15

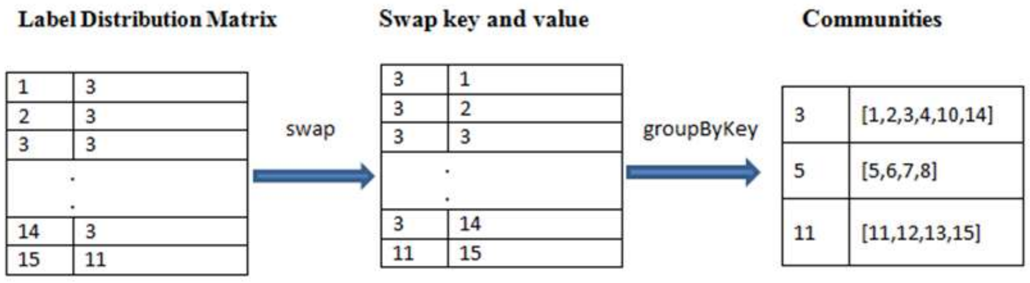
3.1.5 Post Processing

Finally post processing is applied to all the labels in the memories of nodes to output the communities. A node may contain varying probabilities for one/more labels, each label's probability are verified with threshold value in order to omit the labels that have less probability. If a node i contains k labels, the probability of each label verified using the following eqn.(2).

$$\overset{k}{\underset{j=1}{\forall}} P_{ij} < r, then \qquad (2)$$

$$P_{ij} = 0$$

After post processing, the labels with low probability becomes zero at the end of iteration every node left with a single label, the nodes with similar label are grouped into one community as shown in Figure 9.



1	3
2	3
3	3
.	
.	
14	3
15	11

3	1
3	2
3	3
.	
.	
3	14
11	15

3	[1,2,3,4,10,14]
5	[5,6,7,8]
11	[11,12,13,15]

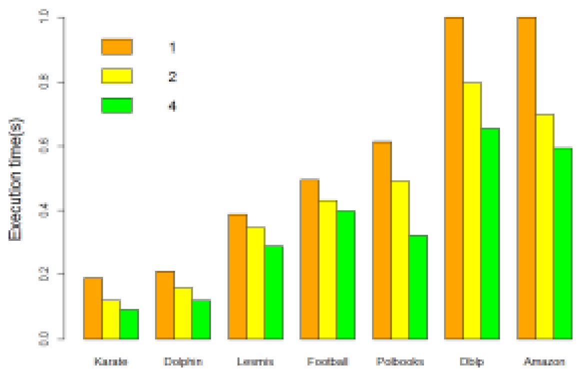
Fig 9: The Lineage graph for RDDs of post processing phase of network size 15

4. Tests in Synthetic Networks

We tested our framework on large social networks available at UCI Network Data Repository for which we know the exact groups and compared its performance with traditional algorithm SLPA in terms of speedup and efficiency. We performed speedup and efficiency using the following equations.

$$\text{Speedup} = \frac{T_1}{T_p} \qquad \text{Efficiency} = \frac{Speedup}{p}$$

Where P is the number of partitions.



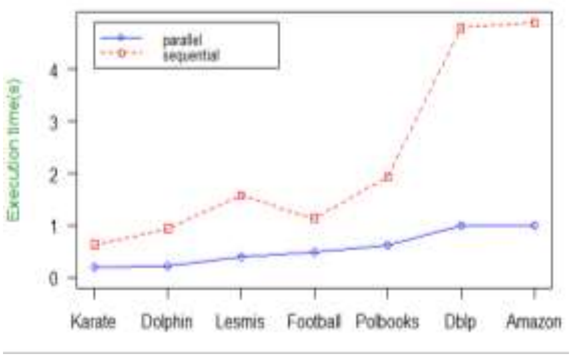
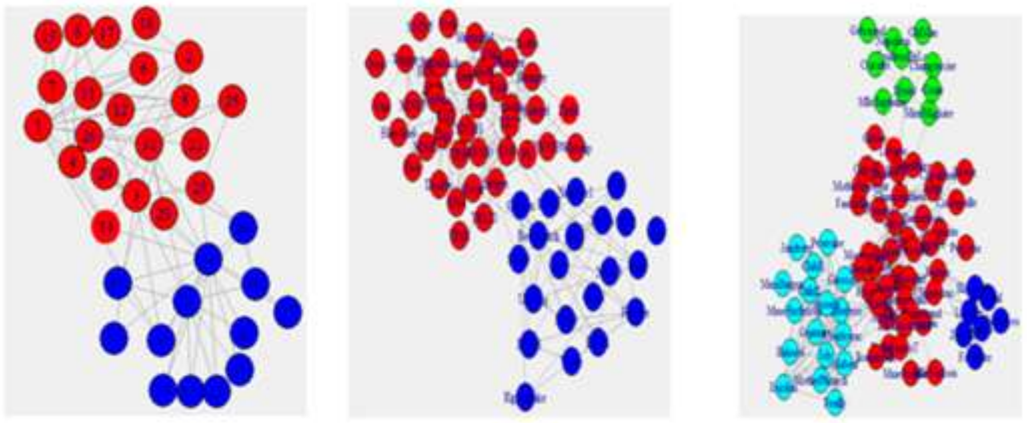
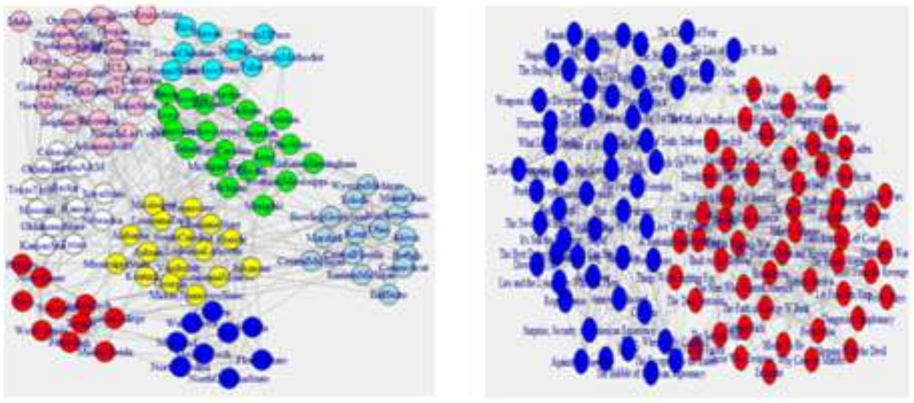


Fig 10: Execution time for different number of partitions



(a) Karate Network (b) Dolphins Networks (c) Lesmis Network



(d) Football Network (e) PolBooks Networks

Fig 11:: Community results of 5 networks of PLPA, where colours represent communities.

5. Conclusion

In this paper, a distributed community detection algorithm has been implemented for large datasets that supports to find the communities in various digital social networking applications, such as personalized e-government services, citizen-centered smart cities. The proposed algorithm overcomes the weaknesses of processing time and space in comparison with traditional community detection algorithms. Besides, this algorithm reduces the computational time of speaker and listener phases through the addition of inflation operator. In order to improve the performance of PLPA several times with respect to large dataset, PLPA is implemented on Spark, which provides an extremely distributed computational environment in relation with other platforms. The theoretical and empirical comparison of PLPA shows the outperformance of community detection algorithms as compare to other existing approaches.

References

[1] Friis, C. S., Demchak, C., & LaPorte, T. (2000). Webbing goernance: National differences in constructing the face of public organizations. In Handbook of public information susytems. Marcel Dekker Incorporated.

[2] Dutton, W. H., & Blank, G. (2011). Next generation users: the internet in Britain.

[3] Hofmann, S., Beverungen, D., Räckers, M., & Becker, J. (2013). What makes local governments' online communications successful? Insights from a multi-method analysis of Facebook. Government Information Quarterly, 30(4), 387-396.

[4] Girvan, M., and Newman, M. E. (2002), Community structure in social and biological networks. Proceedings of the National Academy of Sciences, Vol. 99, No. 12, pp. 7821-7826.

[5] Blondel, V. D., Guillaume, J. L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment, 2008(10), P10008

[6] Adamcsek, B., Palla, G., Farkas, I. J., Derényi, I., & Vicsek, T. (2006). CFinder: locating cliques and overlapping modules in biological networks. Bioinformatics, 22(8), 1021-1023.

[7] Cao, X., Wang, X., Jin, D., Cao, Y. and He, D., (2013) ‘ Identifying overlapping communities as well as hubs and outliers via nonnegative matrix factorization’, Scientific reports, 3, p.2993

[8] Zeng, J. and Yu, H., (2015), “Parallel Modularity-based Community Detection on Large-scale Graphs”, In Cluster Computing(CLUSTER) IEEE International Conference

[9] Raghavan, U.N., Albert, R. and Kumara, S., (2007), “Near linear time algorithm to detect community structures in large-scale networks”, Physical review E, 76(3), p.036106

[10] Xie,J., Szymanski, B.K. and Liu, X., (2011), “Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process”, In Data Mining Workshops (ICDMW) IEEE 11th International Conference, pp. 344-349

[11] Dickinson, B. and Hu, W., (2015), “The Effects of Centrality Ordering in Label Propagation for CommunityDetection”, SocialNetworking, 4(04), p.103.

[12] Wu, Z.H., Lin, Y.F., Gregory, S., Wan, H.Y. and Tian, S.F., (2012), “ Balanced multi-label propagation for overlapping community detection in social networks”, Journal of Computer Science and Technology, Vol.27, No.3, pp.468-479

[13] Prabavathi, G. T., and V. Thiagarasu., (2014), “Design and development of overlapping community detection algorithm using Multi-Label Propagation”, International Journal of Advance Research in Computer Science and Management Studies, Vol.2 No.2, pp. 195-199.

[14] Angadi, A. and Varma, P.S., (2015), "Overlapping community detection in temporal networks", Indian Journal of Science and Technology, 8(31).

[15] Xie,J., Szymanski B K. ,(2013), "Labelrank: A stabilized label propagation algorithm for community detection in networks", In Network Science Workshop (NSW), pp. 138–143.IEEE

[16] Xie,J., Chen, M. and Szymanski, B.K., (2013), "LabelrankT: Incremental community detection in dynamic networks via label propagation", In Proceedings of the Workshop on Dynamic Networks Management and Mining, pp. 25-32, ACM

[17] Bhat, A.U., (2012), Scalable community detection using label propagation & map-reduce

[18] Kuzmin, K., Shah, S.Y. and Szymanski, B.K., (2013), "Parallel overlapping community detection with SLPA", In Social Computing (SocialCom), International Conference, pp. 204-212, IEEE

[19] Kuzmin, K., Chen, M. and Szymanski, B.K., (2015), "Parallelizing SLPA for scalable overlapping community detection" , Scientific Programming, 2015, pp.4

[20] Sotera (2014) [online] https://sotera.github.io/ distributed-graph-analytics/louvain/

[21] Prat-Pérez, A., Dominguez-Sal, D. and Larriba-Pey, J.L.,(2014), " High quality, scalable and parallel community detection for large real graphs", In Proceedings of the 23rd international conference on World wide web pp. 225-236. ACM.

[22] Xin, R.S., Gonzalez, J.E., Franklin, M.J. and Stoica, I., (2013), "Graphx: A resilient distributed graph system on spark", In First International Workshop on Graph Data Management Experiences and Systems, p. 2, ACM.

[23] Buzun, N., Korshunov, A., Avanesov, V., Filonenko, I., Kozlov, I., Turdakov, D. and Kim, H., (2014), "Egolp: Fast and distributed community detection in billion-node social networks.", In Data Mining Workshop (ICDMW), IEEE International Conference on (pp. 533-540).

[24] Rathee, S., Kaul, M. and Kashyap, A.,(2015), "R-Apriori: an efficient apriori based algorithm on spark" , In Proceedings of the 8th Workshop on Ph. D. Workshop in Information and Knowledge Management , pp. 27-34, ACM.

[25] Dongen, S.M., (2000), Graph clustering by flow simulation.

[26] Koy, Albert.(2015), "Subgraph-centric Large-Scale Graph Analytics on Spark."