Efficient Pattern Mining of Big Data using Graphs

A Thesis
submitted in partial fulfillment of the requirements
for the award of degree of

Doctor of Philosophy

by
Vandana Bhatia
(Roll No. : 901403001)

Under the Guidance of
Dr. Rinkle Rani
(Associate Professor, Computer Science and Engineering Department,
Thapar Institute of Engineering and Technology, Patiala)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA – 147004

Feburary 2018

Table of Contents

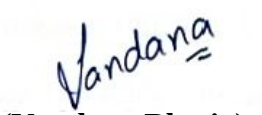
List of Figures

List of Tables

Certificate

I hereby certify that the work which is presented in this thesis entitled **"Efficient Pattern Mining of Big Data using Graphs"**, in partial fulfillment of the requirement for the award of degree of "**Doctor of Philosophy**" submitted in Computer Science and Engineering Department of Thapar Institute of Engineering & Technology, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. Rinkle Rani** and refers other research works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.



(**Vandana Bhatia**)
Regn. No. 901403001

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(**Dr. Rinkle Rani**)

Associate Professor

Computer Science & Engineering Department

Thapar Institute of Engineering and Technology,

Patiala, India

Acknowledgement

I would like to express sincere appreciation to my supervisor **Dr. Rinkle Rani** for being a pillar of support and encouragement throughout my research work. Her experience, strength, tenderness and willfulness, have taught me valuable lessons of life, that are going to be of immense help to me in taking decisions throughout my life. My sincere thanks are due to **Dr. Maninder Singh**, Professor and Head, CSED, Thapar Institute of Engineering and Technology for providing me the necessary administrative assistance and infrastructure that helped me in completion of my research work. I am thankful to my doctoral committee members, **Dr. Prateek Bhatia**, Associate Professor, CSED, **Dr. Ashima Singh**, Assistant Professor, CSED, and **Dr. Kulbir Singh**, Professor, ECED, Thapar Institute of Engineering and Technology, Patiala for their helpful suggestions and regularly ensuring the progress of my research work. I am thankful to all the faculty, staff members and research scholars of CSED, Thapar Institute of Engineering and Technology for their support.

I offer my deepest gratitude to my father, **Dr. Avnish Kumar Bhatia** whose love and constant encouragement has been a major source of inspiration in turning my vision into a reality. I am also thankful to my mother **Mrs. Neelam Rani** and my brother **Atal Bhatia** for love, encouragement, motivation and confidence in me. I also offer my gratitude towards my late grandparents **Smt Ishwar Devi**, **Dr. Hans Raj Bhatia** and Aunt **Sudesh Bhatia** for showering their blessings on me.

I also acknowledge the cooperation and encouragement by my friends for providing support and motivation in this journey.

The chain of gratitude would be definitely incomplete without thanking the supreme power, the **Almighty** for showering blessings in all of my endeavors.

Patiala (Vandana Bhatia)

February, 2018

Abstract

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 among the entities is of much more importance than the information itself. Big data has many such connections which can be mined efficiently using graphs. However, it is very challenging to obtain ample profits from this complex data. To overcome these challenges, graph mining approaches such as clustering and subgraph mining are used. In recent times, these approaches have become an indispensable tool for analyzing graphs in various domains.

This thesis presents research work undertaken in the field of pattern mining approaches for large graphs. The main objective of this research is to investigate the benefits of using scalable approaches for mining large graphs. Two fuzzy clustering algorithms namely 'PGFC' and 'PFCA' are proposed for large graphs using different concepts of graph analysis. Furthermore, a scalable deep learning based fuzzy clustering model named 'DFuzzy' is proposed that leverages the idea from stacked autoencoder pipelines to identify overlapping and non-overlapping clusters in large graphs efficiently. Our proposed clustering approaches are proved to be effective for small and large graph dataset, and generate high quality clusters. For mining frequent subgraphs, a scalable frequent subgraph mining algorithm named 'PaGro' is proposed for a single large graph using pattern-growth based approach. In PaGro, a two-step hybrid approach is developed for optimization of subgraph isomorphism and subgraph pruning task at both local and global levels to avoid the excess communication overhead. Additionally, an approximate frequent subgraph mining algorithm named 'Ap-FSM' is proposed which exploits PaGro using sampling for faster processing. A novel

sampling approach is proposed in ‘Ap-FSM’ for the selection of an approximate subgraph while capturing the original graph properties for convenient and relatively easy analysis. The results of PaGro and Ap-FSM show that both outperform the competent algorithms in various aspects of processing Time, no. of iterations and memory overhead.

It is suggested that the utilization of graph clustering and frequent subgraph mining generate discriminate and significant patterns, which can help in many tasks such as classification and indexing of big data.

1. INTRODUCTION

Big data has a good potential to transfigure most of the aspects of the society. But, harvesting the valuable knowledge from the huge amount of data from various sources is still a remarkable challenge in the area of knowledge discovery and data mining [1]. The connections among various entities of big data collectively form complex networks. A complex network is a graph having significant topological features that are common in real-life networks of various domains such as biological networks [2], social networks [3]–[5], collaboration networks [6], etc..

1.1 Mining Big Data

Big data mining is an important research area and has attracted considerable amount of attention in last decade [7]. It allows to process, analyze, and extract meaningful information from large amounts of data. Big data has great amount of hidden knowledge and many insights which have raised remarkable challenges in knowledge discovery and data mining [8], [9] . Big data mining has many applications including health care [10], [11], social networks [12], cloud services [13], chemical compounds [14], business analytics [15], [16], sensor networks [17], etc.. But mining big data possess ample challenges [18]. This huge amount of data is distributed and cannot fit into memory of a single system [19].

Many approaches have been designed for distributed data management [20]. It includes clustering [21], classification [22] and frequent itemset mining [23]. For certain types of data, the relationship among the entities is of much more importance then the information itself. Big data has many such connections which can be mined efficiently using graphs [21].

1.2 Mining Data using Graphs

A graph is a structure that represents abstract entities (vertices) and their relationships (edges). Big data can be represented in a diverse variety of graphs such as social networks, biological networks [24], web, chemical networks etc. With the help of graphs, understanding the relationships (links) and the content (text or images) will be quite easy in comparison to the traditional relational databases. Graphs are well suited for analyzing interconnections. Thus, there is a lot of interest in analyzing data from domains like social media in the form of graphs. Graphs are useful in many other domains also like business that involve complex and dynamic relationships, communication networks, analyzing stock market data [25] and for recommendations like "customers who bought this also looked at".

Mining graphs to get the useful information has been the most evolving and explored area in the field of knowledge discovery and data mining [21]. Graphs are broadly used in modelling of complex structures [26] in several commercial and scientific applications such as social networks [27], [28], chemical compounds [29], [30], images, biological networks etc..

Real-life graphs usually contain subgraphs where certain vertices have more connections with each other than the rest of the vertices in graph [4]. The process of mining such subgraphs is known as graph clustering. Likewise, there may exist certain subgraphs that happen to occur frequently in the graph. Mining such subgraphs can help in wide range of applications. Mining large graphs poses so many challenges as their processing upsurges storage and time overhead. Valuable hidden information and patterns cannot be discovered easily from this ostensibly frenzied data without using sophisticated methods, making it is so difficult to obtain ample profits using that data.

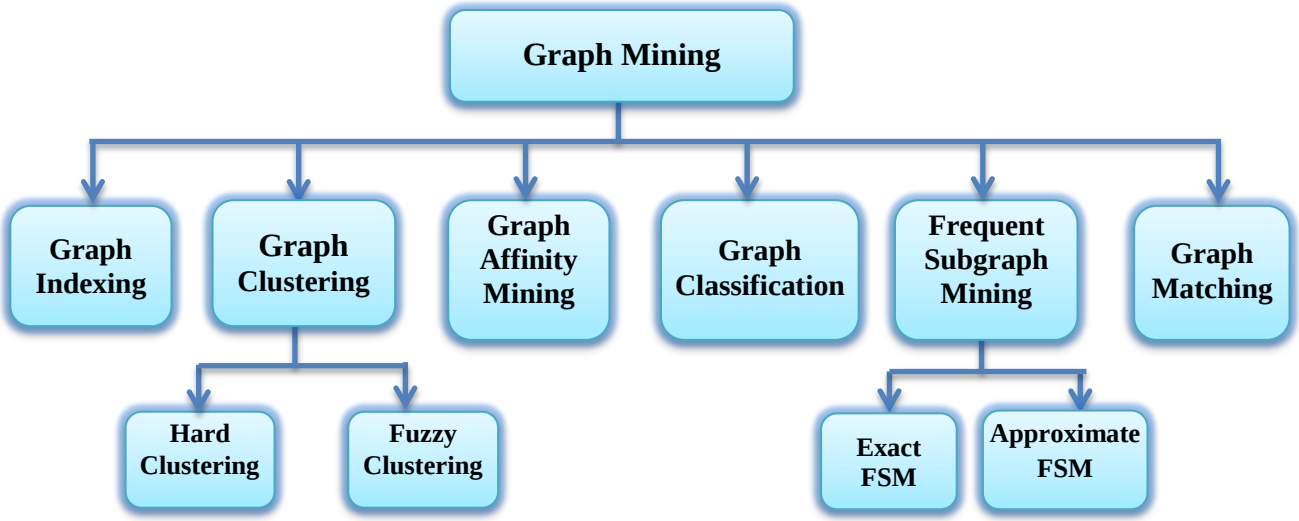


Fig. 1.1 Graph Mining Approaches

One way to overcome these challenging problems is to have large graphs clustered in certain dense arrangement which still is an informative version of the whole data. Among other approaches graph indexing, graph affinity mining, graph classification, frequent subgraph mining (FSM) and graph matching are amongst the most popular graph mining algorithms. The classification of graph mining approaches is shown in Fig. 1.1. The main focus of this research is on graph clustering and frequent subgraph mining as both lays foundation of other approaches for graph mining such as graph classification, graph compression, graph indexing, graph matching and correlation analysis.

1.3 Pattern Mining using Graph Clustering

Real-life graphs usually contain sub-graphs where certain vertices have more connections with each other than the rest of the vertices [31], [4]. Clustering helps in understanding the patterns present in graphs and thus helps in getting useful insights. Graph clustering is obliging in many real-life applications such as community detection [32], [33], image segmentation [34], protein-protein interaction [35] etc..

In real-life graphs, clusters can be hard [36], fuzzy [37] or nested. K-means is the widely used hard clustering algorithm which finds the pre-specified number of clusters by partitioning the

graph. However, many such applications exist where a node may belong to more than one cluster [38], [39]. In social networks, users may belong to numerous communities [4], [40]. In biological networks, genes may have several coding functions and may also participate in more than one metabolic pathway. In text mining and information retrieval, documents [41], web pages, news articles, etc. may belong to multiple categories. For finding overlapping among clusters, K-Means is extended to Fuzzy C-means (FCM) clustering algorithm by Bezdek [37]. However, both FCM and K-means are sensitive to the selection of initial centers and require the number of clusters to be mentioned in beginning. But, for graphs of real world, the structure of the graph plays an important role in determining the number and size of clusters [42].

Moreover, most of the existing fuzzy clustering algorithm are designed for centralized systems [40], [43]–[45]. But, for large graphs of big data, it is highly desirable for data to be distributed instead of centrally stored. Furthermore, the existing research focused on the small data and used traditional graph mining approaches which are centralized. Among the existing graph algorithms, some of them can be used to deal with the large data [32], [46]–[48] but, they are not able to deal with large graphs and the related scalability issues.

1.4 Pattern Mining using frequent subgraphs

Frequent subgraph mining (FSM) is a popular method in the field of knowledge discovery from graphs. Finding frequent subgraphs from the collection of graphs or from the large graphs is especially useful in characterizing the graph datasets, discerning dissimilar group of networks, clustering [49] and classification [50] of graphs, detecting network motifs [51], and structuring graph indices. The problem of frequent subgraph mining is addressed in many algorithms using various approaches, mainly apriori-based approach [52], [53], pattern growth approach [30], [54] and inductive logic programming.

However, the existing solutions are evaluated over relatively small graph datasets and rely on the centralized computing paradigms. Due to the dramatic increase in the graph size today, most of the existing techniques face the issue of scalability. Consequently, certain distributed and parallel solutions have also been proposed to solve the scalability issue [55]–[58].

Many of them used Map-Reduce [59] to deal with the large data. But, Map-Reduce do not handle the graphs efficiently. It incurs memory as well as communication overhead. To deal with large graph database, a message passing based computational model named 'Pregel' is introduced in which the programs are executed as a sequence of iterations [60].

1.5 Challenges of dealing with large graphs

Handling large graphs in distributed environment for efficient processing also incurs many challenges due to variation of support values for the task of frequent subgraph mining in various systems. In addition, the real-world graphs are dense in nature and traditional graph clustering and FSM algorithms fails in handling dense graphs.

Also, most of the existing algorithms are centralized and can handle small graphs only that fit into memory of a single system. When distributed, these algorithms incur high communication and I/O overhead. In addition, the existing algorithms do not provide scalability, consistency and Precision.

In the recent years, with the unprecedented growth of digital data in various domains, there is a surge for designing scalable, consistent and precise algorithms. Thus, distributed solutions which can deal with large graphs with accuracy are in high demand.

1.6 General Problem Definition

In general pattern mining from big data using graphs can be stated as: Given a graph $G = \{V, E\}$, where V defines a finite set of vertices and $E \subseteq V \times V$ denotes a set of edges, find the set of subgraphs k that have certain inter-subgraph or intra-subgraph similarity.

The general problem is further divided into two sub-problems.

(a) In a graph G (V,E), find k subgraphs such that the vertices inside a subgraph are more connected to each other than the vertices of other subgraphs.

(b) Given a graph G (V,E) and a threshold value τ, find all the subgraphs that occurs at least τ times in the graph G.

The first problem is of graph clustering and the second belongs to frequent subgraph mining. The thesis focuses on finding graph clusters in which vertices may belong to more than one cluster. The research emphasize on minimizing the pre-defined parameters.

For the second problem, the focus is on finding exact as well as approximate frequent subgraphs from a single large graph.

1.7 Research Contribution

The main focus of research in this thesis is designing and development of efficient algorithms for mining subgraph patterns with different properties. The major contributions of the research are:

- A detailed survey on the pattern mining of big data using graphs is provided. Special emphasis is given on graph clustering and frequent subgraph mining.
- The problem of graph clustering is presented in detail and the proposed solutions to this problem are explained.

- Scalable fuzzy clustering approaches are proposed for large graphs using different concepts of graph analysis. The three proposed approaches are:
  - **PGFC**: This algorithm finds fuzzy clusters by amending the structure of the classical Fuzzy C-means algorithm for large graph data. Instead of random initialization, cluster centers are selected using degree centrality measure. It is proved that PGFC performs better than state-of-art clustering algorithms in terms of partition coefficient and conductance.
  - **PFCA**: This proposed algorithm first selects the candidate cluster heads based on their influence in the network and then determines the number of clusters by analyzing the graph structure using PageRank algorithm. The results show that PFCA is consistent and highly scalable.
  - **DFuzzy**: It 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 by analyzing the graph structure in multiple layers.
- The problem of frequent subgraph mining is discussed in detail and the proposed solutions to this problem have also been explained.
- Scalable frequent subgraph mining approaches are proposed for a single large graph using pattern-growth based approach. The two proposed approaches are
  - **PaGro**: An exact subgraph mining algorithm is proposed by leveraging the operative communication primitives for better scalability. A two-step hybrid approach is developed for optimization of subgraph pruning task at both local and global level to avoid the excess communication overhead. PaGro performs better than state-of-art FSM algorithms in terms of processing time and memory overhead.
  - **Ap-FSM**: It is an approximate frequent subgraph mining algorithm which exploits sampling for faster processing. A novel sampling approach named G-Samp is proposed

for the selection of an approximate subgraph while capturing the original graph properties for convenient and relatively easy analysis. The results show that it outperforms the competent algorithms and is time efficient.

1.8 Thesis Organization

The Thesis is organized as per the following chapters:

Chapter 1: Introduction

This chapter introduces the concept of pattern mining in graphs, describing it as a computational problem. It touches the topic of graph mining approaches that can be used for big data. Then, it moves on to describe the significance of mining frequent patterns, laying a foundation for the introduction of the concept of frequent subgraph mining. The problem of graph clustering and frequent subgraph mining is stated, and various solutions to the problem are also enlisted. This chapter also discusses the challenges faced while dealing with large graph data, and the issue of handling graphs in distributed environment. In a nutshell, chapter explains the basic concepts of graph clustering, and shows how it is merged with the concept of frequent subgraphs.

Chapter 2: Background and Related Work

This chapter contains a detailed review of the literature on mining large graphs, with special emphasis on graph clustering and frequent subgraph mining. There is a list of case studies, which show that finding relation between data objects is the best way to deal with the problem of ever increasing data. A segment on related research discusses the existing surveys on this topic. A significant portion of this chapter consists of an extensive list of graph clustering approaches for big data mining. It also discusses the increasing use of graph affinity approaches such as PageRank in finding undiscoverable patterns in graphs. Further,

in-depth discussion on frequent subgraph mining is also presented. This chapter also discusses other approaches that are being used frequently by researchers and developers while mining big data using graphs. Finally, this chapter also contains a few comparative studies highlighting the advantages and drawbacks of the discussed approaches.

Chapter 3: Problem Formulation

This chapter lists the research gaps that are identified initially, while proposing the research objectives. These gaps identify the problems with the existing graph clustering algorithms. They point out the fact that there is a need for working on developing approaches that are based on finding patterns in large graphs that cannot fit in the memory of a single system, and applying them for developing applications that requires scalability. Based on these gaps, there are four research objectives that is addressed in different chapters of this thesis.

Chapter 4: Clustering in Large Graphs

In this chapter, the proposed approaches for graph clustering are introduced. The first algorithm is PGFC (Parallel Graph Fuzzy Clustering) which is implemented by amending the structure of the classical Fuzzy C-means algorithm for large graph data in distributed environment. BSP based Pregel model is followed to design the distributed version of the algorithm. The second algorithm named PFCA is proposed to overcome the issues present in PGFC. PFCA learns the structure of graph vertices by modeling sequence of random walks using PageRank algorithm. A Deep learning based model named ‘DFuzzy’ is designed by considering both structure and distance for clustering task which makes it efficient. The working of all the three algorithms is elaborated in this chapter.

Chapter 5: Experimental Evaluation

In this chapter, details of the experiments performed on real and synthetic graph data sets, to validate the proposed approaches are given. The first phase of experimentation is performed

on testing the proposed clustering algorithms, and comparing them with the existing graph clustering approaches. Standard performance evaluation metrics are used to evaluate the quality of the clusters obtained by applying the proposed approaches.

The next phase of experimentation involves testing the accuracy of the proposed approaches for real life graph datasets. The next step is to check the scalability of the proposed approaches by running them on very large datasets (with million or billions of vertices) using parallel and distributed computing platform. This chapter shows that the proposed approaches are suitable for tackling large graphs. The last part of the chapter presents a comparative analysis of the proposed approaches, stating their advantages and disadvantages, along with a short comparison to other state-of-art clustering.

Chapter 6: Frequent Subgraph Mining of Large Graph

In this chapter, the proposed frequent subgraph mining approaches are introduced. An exact subgraph mining algorithm named PaGro is proposed by leveraging the operative communication primitives for better scalability. A two-step hybrid approach is developed in PaGro for optimization of subgraph pruning task at both local and global levels to avoid the excess communication overhead. The NP-Complete graph isomorphism problem has also been optimized for fast graph processing. The working of PaGro is amended for performing approximate frequent subgraph mining in Ap-FSM by exploiting sampling for faster processing. A novel sampling approach named G-Samp is proposed in Ap-FSM for the selection of an approximate subgraph while capturing the original graph properties for convenient and relatively easy analysis. At last, the evaluation of both the approaches is performed using real life graph datasets. The results of PaGro and Ap-FSM show that both outperform the competent algorithms in various aspects. The scalability of both the algorithms is shown using parallel and distributed graph processing framework Giraph.

Chapter 7: Conclusion and Future Directions

This chapter concludes the thesis by providing a brief overview of the proposed framework, and providing an insight into the future scope of work. Graph clustering and frequent subgraph mining are evolving fields, and this chapter brings to light the huge amount of scope for developing approaches and applications in this domain. It highlights the contributions made by our work and enlists the points that need to be worked on in future.

2. BACKGROUND AND RELATED WORK

Graphs appear in a wide range of settings and constitute a large portion of real word data sets [21], [61]. This chapter contains a detailed review of the literature on mining large graphs, with special emphasis on graph clustering and frequent subgraph mining. There is a list of case studies, which show that finding relation between data objects is the best way to deal with the problem of ever increasing data. First graph mining approaches are discussed. Graph clustering and subgraph mining approaches are emphasized because they are the building block for other graph mining approaches such as graph classification, compression, and correlation analysis. The existing approaches for graph clustering and frequent subgraph are discussed in detail. At last, the challenges in various applications that are sources of big data are discussed.

2.1 Graph Mining

Mining graph data is of great interest and is the crucial research area in the field of data mining. Many types of data such as semi-structured or structured data as well as XML data can be represented as graphs [62]. The concept of fuzziness is also used along with graphs in certain research [63], [64].

Representing data as graphs to understand the relationships among data objects is gaining high interest. But, with the increase in the volume of data, the size of the graph is also increasing. The researchers are showing a lot of interest in applying graph analytics on data available from social media [65]–[67], biological networks [68], [69], business networks [70] etc..

2.1.1 Graph Mining Approaches

Graph mining has become an imperative topic of research in recent times because of several applications in widespread data mining issues including computational biology [30], chemical network [54], communication networking [53] and drug discovery. Many data mining algorithms such as classification [71], [72] clustering [73], indexing [74] and frequent subgraph mining [75] have been extended to graph setting. Frequent subgraph mining and graph clustering are popular research areas under graph pattern mining [35], [76], [77]. Both the approaches are the building blocks for graph classification, indexing, compression, pattern matching and correlation analysis.

2.1.2 Processing Large graphs

Large data graphs are big and ubiquitous with billions of nodes and edges. It is highly desirable for such graphs to be distributed instead of centrally stored. Most of the research has been focused on the small data using traditional graph mining approaches. Existing graph mining techniques are centralized and lack scalability. To have distributed versions of such approaches, a lot of research is required with practical and theoretical analysis to provide new methods.

Mining large graph poses so many challenges while handling massive amount of data. A promising solution to overcome the challenges is using graph data in distributed manner. The most popular framework for handling big data is Map-Reduce[59] 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. 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 [60] 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. An 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.

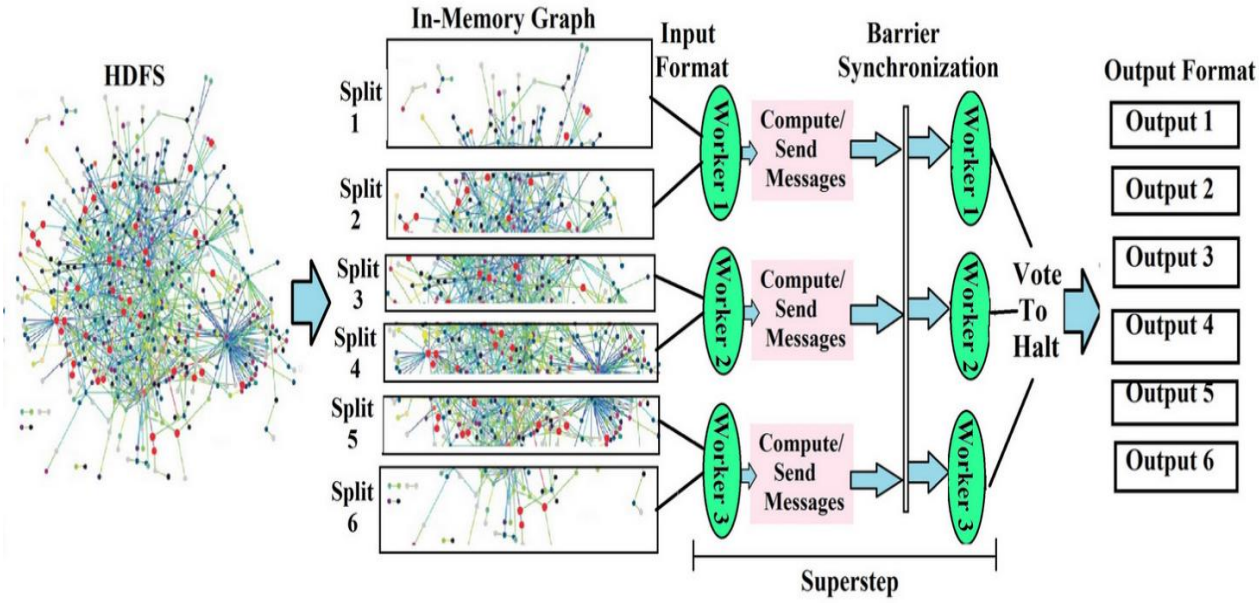
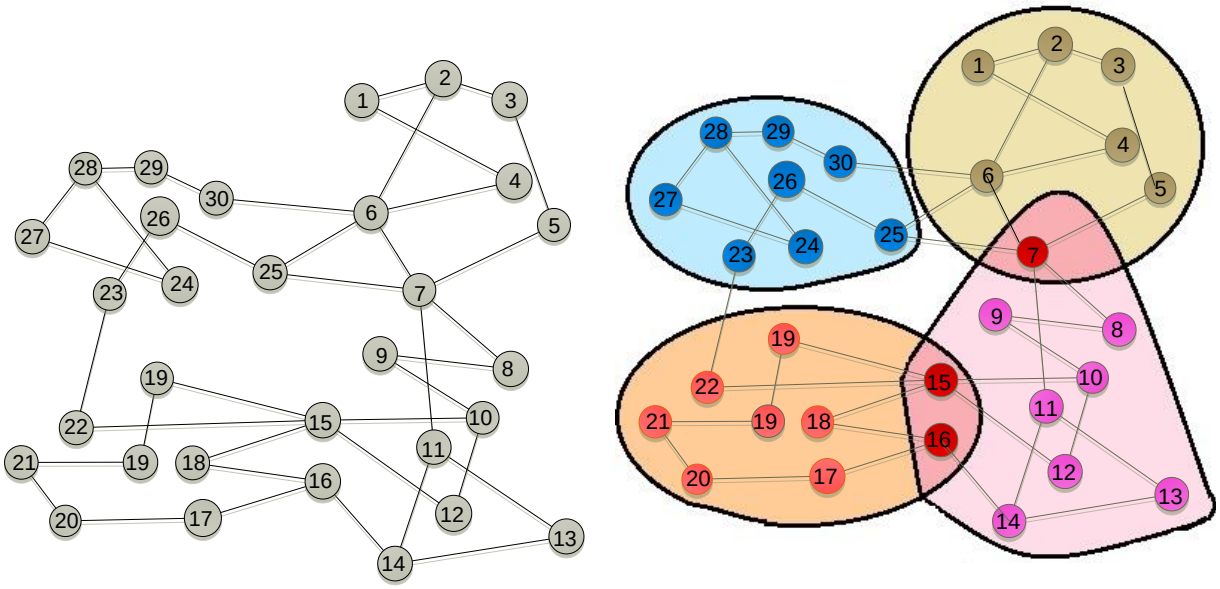


Fig. 2.1 Graph Processing in Giraph

The input graph is partitioned among the worker nodes as shown in Fig. 2.1. 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. There exist few work which exploited Pregel for large graph processing [78]–[80].



(a) (b)

Fig. 2.2 (a) A graph G with 30 nodes (b) Detection of Four Cluster in graph G

2.2 Graph Clustering

Clustering techniques can be categorized into partition based, hierarchical, density based and grid based [48]. The most popular partition based clustering algorithm is K-Means. However, K-means finds the non-overlapping clusters only. But in real life graphs, cluster may overlap such that a vertex or an edge belongs to more than one cluster. In such cases, it is necessary to allow overlapping among clusters.

For example in Fig. 2.2, four clusters are formed in the given graph of 30 vertices. Vertex 7, 15 and 16 belongs to two clusters while the rest of the 27 vertices belong to a single cluster. Such kind of graphs can be found frequently in real life applications. 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 [81]. In information retrieval and text mining [82], documents[83], news articles and web pages [84] can belong to many different categories.

Clustering algorithms can be categorized as local and global. The global information is not

appropriate for getting understandings from large scale complex networks as the worthwhile insights cannot be extracted from millions, but from many smaller groups of varying sizes. The local information of the network can be gathered by analyzing the structure of the network. Most of them were based on affinity based graph mining algorithms such as random walk, random walk with restart. Some of the existing research also focused on the similarity between vertices, edges and neighborhood of the vertices.

Label Propagation algorithm (LPA) is among the popular clustering algorithm which considers solely the network structure for clustering [85]. Many variants of LPA have been developed [86]. The fuzzy variants of LPA have also developed [87]. However, LPA suffers from the randomness issue thus different clusters might be detected in different runs for the same network. Such inconsistencies are not desirable and should be avoided while handling real-life scenarios.

For Complex networks, generating clusters by analyzing structure of the network provides high precision. To analyze the structure of the network, random walk based methods such as PageRank [88] have been used for performing clustering [32], [77], [89].

The comparative study of some features of the existing clustering algorithm is presented in Table 2.1. The features on the basis of which comparison is done are selection of cluster center, type of data, no. of cluster, network structure analysis, detection of fuzzy clusters and computation of membership towards cluster and consistency in results.

The existing research for major characteristics of the graph clustering algorithms is discussed in following sections:

2.2.1 Cluster Centrality

For identifying accurate clusters in the network, the selection of seeds or center heads play a vital role. The influence of the vertices of network can be a good centrality with respect to the degree of vertex. The higher is the degree of the vertex, more is the centrality.

Table 2.1: Comparison of properties of clustering algorithms

Algorithm	Selection of Cluster Center	Type Of Data	Number of Cluster	Analyze Network Structure	Find Fuzzy Clusters	Finds Fuzzy Membership	Consistency In Results
Standard Clustering algorithms							
K-Means	Random	-	Pre-Defined	No	No	No	No
Fuzzy K-means	Random	-	Pre-Defined	No	Yes	No	No
Fuzzy C-means	Random	-	Pre-Defined	No	Yes	Yes	No
CURE	Random	-	Pre-Defined	No	No	No	Yes
Graph Clustering algorithms							
LPA [90]	Random	Single Graph	Pre-Defined	Yes	No	No	No
GraphEncoder [91]	Node Similarity	Single Graph	Pre-Defined	Yes	No	No	No
DLC[92]	Random	Single Graph	Pre-Defined	No	No	No	Yes
OLP [93]	Random	Single Graph	Pre-Defined	Yes	Yes	No	No
StruClus [94]	Freq. Subgraphs	Set of Graphs	Pre-Defined	No	No	No	Yes
PFM [95]	Random	Single Graph	Using Probability	Yes	Yes	No	Yes
RM-CRAG [96]	Random	Single Graph	Pre-Defined	Yes	No	No	Yes
CSIP [97]	Influence-based	Single Graph	Pre-Defined	Yes	No	No	Yes
Graph Clustering algorithms for Big Data							
Semi-Clust.[60]	Score calculation	Single Graph	Pre-Defined	No	No	No	Yes
MR-FCM [98]	Random	Single Graph	Pre-Defined	No	Yes	Yes	No
BIGCLAM [99]	Random	Single Graph	Pre-Defined	Yes	Yes	No	Yes
SIMPLE [100]	Radom	Single Graph	Pre-Defined	No	No	No	No
ClusterWild [101]	Radom	Single Graph	Pre-Defined	Yes	No	No	No
PSCG [73]	Random	Set of Graphs	Pre-Defined	Yes	Yes	Yes	Yes

Also, the neighbors of the vertex with the higher centrality have more influence in network. The influence can be calculated by analyzing the connections. Many applications like social networks, collaboration networks, etc. use the concept of distance between the two vertices of network for estimating the connection between them.

Strategies for finding appropriate seed vertices includes degree centrality, edge betweeness, local minimal neighborhoods, etc. [5]. Appropriate cluster head selection is a challenge in wider range of applications [102], [103], [104]. However, if considered solely, they all avoid the distance between the seed and the rest of the vertices of the cluster. Following are certain centrality approaches that are used for finding appropriate cluster center:

A. Degree Centrality

Degree is defined as the number of links incident upon a vertex. The degree can be inferred in terms of the immediate risk of a vertex for catching whatever is flowing through the network such as some information or some virus. In directed networks, two distinct measures for degree centrality are defined, namely in-degree and out-degree.

Correspondingly, in-degree is a sum of the number of links directed to the vertex and out-degree is the number of links that the vertex directs towards other vertices. When links are concomitant to some positive aspects such as collaboration or friendship, in-degree is frequently interpreted as a form of popularity, and out-degree as gregariousness.

B. Betweenness centrality

It measures the number of times a vertex acts as a bridge in the shortest path between any two other vertices. It was presented by Linton Freeman [105] as a measure for computing the control of a human on the communication with other humans in a social network. It is based on the conception “vertices that have a high probability to occur on a randomly

chosen shortest path between two randomly chosen vertices have a high betweenness".

The betweenness can be computed by finding the shortest paths between two vertices *i* and *j*. Let σ_{ij} be the shortest path between vertices *i* and *j*. The betweenness centrality for a vertex *v* is the number of shortest paths σ_{ij} that pass through the vertex *v*. It is given as:

$$BC(v) = \sum_{i \neq v \neq j} \frac{\sigma_{ij}(v)}{\sigma_{ij}} \qquad (1)$$

The betweenness centrality can be calculated as:

- Vertex betweenness: The number of shortest paths σ_{ij} in the graph *G* that pass through a given node v.
- Edge betweenness: The number of shortest paths σ_{ij} in the graph *G* that pass through a given edge (*u, v*).

C. Closeness centrality

Closeness is a measure of the degree to which an individual is closely connected with all the other individuals in a network. It is calculated as the inverse of the sum of the shortest distances of each node with every other node in the network.

In a connected graph, the normalized closeness centrality of a vertex is the average length of the shortest path of the vertex with all other vertices in the graph. Hence the more central a node is, the closer it is to all the other nodes.

Bavelas [106] defined Closeness in 1950 as the reciprocal of the farness. It is given as:

$$C_l(v) = \frac{1}{\sum_u d(u,v)} \qquad (2)$$

where, *d(u,v)* is the distance between vertices *v* and *u*. It is most commonly referred in normalized form.

D. Eigenvector centrality

The Eigenvector approach uses a factor analytic procedure for measuring closeness. It discounts closeness to small local sub networks. It is a measure of the influence of a node in a network. It assigns relative scores to all the nodes in the network based on the concept that connections to high-scoring nodes are more contributing than equal connections among low-scoring nodes.

Let the adjacency matrix of graph G be $M=(m_{v,u})$. Similarly to notation $Ax=\lambda x$, the eigenvector centrality score of vertex v can be computed as:

$$\text{Eg(v)}=\frac{1}{\lambda}\sum_{u\in G} m_{v,u}.Eg(u) \qquad (3)$$

E. PageRank centrality

Like Eigenvector centrality, PageRank uncovers influential or important nodes whose reach extend beyond just their direct connections. There is a profound concept of connected walks on network 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. Given a graph G and a start node v_i, we select an adjacent node of v_i at random, and make a transition, after which we continue the random walk V' from the newly chosen v_j. The probability P_{ij} associated of jumping from node v_i to node v_j is $\frac{m_{ij}}{d_i}$, where m_{ij} is the entry in matrix M and $d_i = \sum_{j=1}^{N} m_{ij}$ is the degree of vertex v_i. Following a random walk from v_i to v_j, some vertices are visited more frequently than others. These are the vertices having more incident links than others and have more importance in the network.

For identifying such vertices, Brin and Page [88] introduced PageRank algorithm. The algorithm was introduced as a web search algorithm to find the most referred web pages. But, it can be very well fitted in various forms of graphs and is pretty much effectual for capturing

relationships among vertices of graphs.

Apparently, PageRank is a method to consolidate random walk of different sizes. Instead of counting the steps in a random walk, PageRank uses a real value 'α' where $\alpha \in [0,1)$. In a graph G with vertices $v_1,.....,v_N$, let the adjacency matrix be $M \in \mathbb{R}^{N*N}$ and the Diagonal Matrix having degrees on the diagonal be $D \in \mathbb{R}^{N*N}$. Let $P = D^{-1}M$ be the random walk transition matrix. The PageRank vector p is equal to:

$$p = (1-\alpha)(I_n - \alpha P)^{-1} s \qquad (4)$$

where, I_n is the identity matrix and $s \in \mathbb{R}^N$ is a vector referred as teleportation vector. In some recent work, PageRank had been used for clustering process [30] ,[31] and thus, is well suited for large scale complex networks [3].

2.2.2 Hard Clustering

Clustering algorithms that groups data into disjoint clustering are known as hard clustering algorithms. Most of the existing research focuses on finding disjoint clusters from the graphs. The most popular hard clustering algorithm is K-Means. Many modifications have been done on K-means to improve its performance [36], [108], [109]. The distributed improved version of K-Means was also introduced in literature [110]. Some other hard clustering algorithm were also proposed recently for large graphs [111], [112]. But most of them lack consistency in results. They produce different results each time the algorithm is executed.

Few of the existing work have used personalized PageRank algorithm for clustering large graph [83], [107], [113]. Personalized PageRank [88] is demarcated as a stationary distribution of the random walk starting from node c that takes an outgoing edge with the probability α and jump back to the center node with probability (1 - α). The Personalized PageRank transition matrix P_s is given by following equation:

$$P_s = (1-\alpha)c + \alpha M \qquad (5)$$

Personalized PageRank is an effective method to complete the assignment of vertices to clusters and has been used in many applications to find clusters [113]. The return probability in personalized PageRank can differ from cluster to cluster. The time for a random walk will be greater for big clusters in comparison to the small clusters. Let π_s be the stationary distribution of P_s. The vertices are allocated to clusters using Cluster (v_i) = $argmax_{s \in K} \pi_s(v)$. Personalized PageRank has very close relationship with graph cuts also.

Pujol et al. [87] proposed an approach by using spectral clustering for determining the community structure in large graphs. Label propagation based graph clustering algorithms were also introduced in literature [111], [114], [115]. It starts by allocation of labels to all the vertices of graph. The vertices propagate their label to their neighbor vertices in multiple iterations. In the end, the vertices with similar label are grouped together to form a cluster. However, it also suffers from the problem of inconsistency. Due to its random nature, the clustering results are unstable. Many improvements to label propagation based clustering were proposed in literature [14]–[18]. An edge based label propagation algorithm was also introduced which cluster edges instead of vertices [14]. But they all deal with small graphs only.

Some of the existing methods also used modularity for forming clusters. Modularity of any Graph *G* is a measure of efficiency of the clusters formed within graph [116]. A graph with large modularity has more intra-edges and less inter-edges in between various clusters. The objective function for calculating modularity for hard clusters is:

$$Q = \frac{1}{2h_c} \sum_{(i,j) \in V^2} \left(M_{ij} - \frac{\deg(i) * \deg(j)}{2h_c} \right) * C_{ij} \qquad (6)$$

Where, M_{ij} is the adjacency matrix, $h_c = \frac{1}{2}\sum_i \deg(i)$ is the number of edges and C_{ij} is the cluster containing vertices *i* and *j*.

Here, $C_{ij} = \begin{cases} 1, & \text{if } (i, j) \in C_i \\ 0, & \text{otherwise;} \end{cases}$

Higher the modularity better is the clustering. The formula in equation (6) conceals some characteristic tradeoff. More edges should be confined within the cluster to maximize the first part. However, the second part is minimized by splitting the graph *G* into clusters with less total degrees. The second part of the equation tells the probability of the presence of an edge amongst vertices v_i and v_j.

Deep learning methods have also been exploited in some research to find clusters in graphs [117], [118]. Jhang and Chen [119] presented a distributed deep belief network based on Map-Reduce framework exploiting data-level parallelism. In their work, several levels of distributed Restricted Boltzmann machine were stacked and then distributed back-propagation algorithm was used for the fine-tuning. Some recent work also presented approaches for finding clusters in large graphs.

2.2.3 Fuzzy Clustering

Fuzzy clustering finds overlapping cluster from the data such that a data object can be part or more than one cluster. Fuzzy C-means (FCM) algorithm is among the most widely used algorithm for finding overlapping among clusters. It was first introduced by Dunn (1974) and then improved by Bezdek(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 [95]. Some improvements implicate the alteration of objective function [120]. The approaches such as neural networks were used along with FCM in some applications

such as forecasting using time-series data [121] and classifying ECG signals [43]. Certain similarity measures too have been used in literature such as vertex similarity [47], Euclidean distance [122]. Several other overlapping cluster detection algorithms were also proposed in literature [123], [124].

The concept of fuzziness is applied on various other problems [125]. Most of the existing algorithms work on the centralized system only [64]. 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 [126]. 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 [95]. But they all have high communication cost and require heavy disk access.

Various metrics to evaluate fuzzy clustering model were also proposed. For calculating the modularity of fuzzy clusters, Nepusz et al. [47] introduced a measure for cluster quality given in equation (8). Membership value, μ_{ik} can be considered as the probability that vertex v_i is a member of cluster c_k. The probability of the case that vertex i and j belongs to the same cluster is the dot product of their membership values, resulting in a similarity measure δ which is used for the fuzzy variant of the modularity given by:

$$Q_{fuzzy} = \frac{1}{2h_c} \sum_{(i,j)\in V^2} \left(M_{ij} - \frac{degree(i) * degree(j)}{2h_c} \right) * \delta(i,j) \qquad (7)$$

where, $\delta(i,j) = \sum_{k=1}^{C} \mu_{ik}\mu_{jk}$. In case of hard clustering Q_{fuzzy} has the same value as of Q.

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 [46]. But for dealing with large graphs, distributed fuzzy clustering algorithms on frameworks like Pregel are required. A Pregel based Semi-clustering [60] was introduced for handling large graphs. But, it requires complex operations and do not find the extent of belongingness towards the candidate cluster centers [127].

Many other fuzzy clustering algorithms were also proposed in literature [32], [77], [128]. But, most of them do not analyze the structure in which data is presented. In summary, the existing algorithms do not provide scalability, consistency and precision simultaneously. In the recent years, with the unprecedented growth of digital data in various domains, there is a surge for designing scalable, consistent and precise algorithms.

2.3 Frequent Subgraph Mining

Frequent subgraph mining has become an imperative area of research because of wide variety of graph mining applications including chemical data analysis [54], drug discovery, computational biology [30], communication networks [53], and many others [129]. The subgraphs that are mined are either approximate or exact. In approximate subgraph mining, error tolerant or sampling approaches are used. Some work have also been done for finding maximal frequent patterns in graph [50], [130], [131]. Most of them have used the concept of edit distance for allowing structural differences [132], [133]. Another algorithm named APGM [134] is proposed to find inexact patterns by using a pre-defined substitution matrix consisting of similarities among labels. Similarity is measured by a product of probabilities. APGM is extended to allow label replacements of both vertices and edges in VEAM [135].

However, both APGM and VEAM require prior knowledge in the form of substitution matrix which is available in fewer applications only.

For a single graph, only few approaches for approximate subgraph mining algorithms were proposed in literature (Chen, Yan, Zhu, & Han, 2007, Flores-Garrido, Carrasco-Ochoa, Ariel, & Martnez-Trinidad, 2014). Most of the algorithms work for only small graph datasets and thus, face scalability issue [138].

Mainly, the frequent subgraph mining algorithms can be classified as apriori-based and pattern-growth based algorithms.

2.3.1 Apriori-based frequent subgraph mining

Apriori-based frequent itemset mining algorithm was first introduced by Agarawal and Srikant[139] in 1994. Later in 2000, Inokuchi et al.[52] proposed an algorithm with similar characteristics for mining frequent subgraphs. In apriori based approaches first subgraphs of size *k* are mined instead of mining the subgraphs of next level *k+1*. The size of the graphs is determined by the number of vertices present in graph. Here, breadth first search strategy must be preferred because graph is represented in level-wise manner.

For producing the larger subgraph of size (*k+1*), all subgraphs of size *k* are merged all together to generate the largest candidate. Two graphs of size *k* can join only if they have (*k-1*) equal elements and the first matrix has a code lesser than or equal to the second one. After they are joined, a graph with a normal form is generated. As shown in Fig. 2.3. First Join operation is performed and then Pruning is done. While performing pruning, the subgraph isomorphism is checked by counting the frequency of each candidate subgraph.

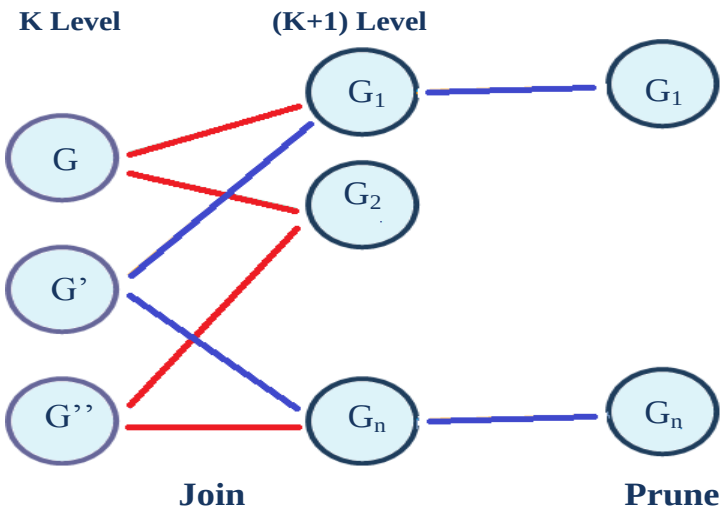


Fig. 2.3 Apriori-based Approach

Algorithm 2.1 : Apriori (D_G, *min_support*, F_k)

Input: A graph Dataset D_G and *min_support*.

Output: A frequent subgraph set F_k

1. $F_{k+1} \leftarrow \emptyset$
2. **For each** frequent $g_i \in F_k$ **do**
3. **For each** subgraph g of size-(k+*1*) formed by merging of g_i and g_j **do**
4. **If** g is frequent in D_G and $g \in F_{k+1}$ **then**
5. Insert g to F_{k+1}
6. **End**
7. **End**
8. **End**
9. **If** $S_{K+1} \neq \emptyset$
10. Apriori(D_G, *min_support*, F_{k+1})
11. **End**

In apriori-based approach, level-wise candidates are generated as shown in Fig. 2.3. Breadth First Search is necessary in this approach as size-*K* subgraphs has to be mined completely before mining size-(*K+1*) subgraphs. The general approach of apriori-based algorithms is given in Algorithm 2.1. Here, F_k is the frequent subgraph set of size-*k*. In every iteration, the size of recently revealed subgraph is increased by 1.

Table 2.2. Comparison of Apriori based approaches

Algorithm	Input type	Graph Representation	Candidate Generation	Frequency counting	Nature of output
FARMER [140]	Set of graphs	Trie Structure	Level-wise search ILP	Trie data Structure	Frequent subgraphs
FSG [141]	Set of graphs	Adjacency List	Single edge extension	Transaction identifier (TID) lists	Frequent connected subgraphs
HSIGRAM [142]	Single large graph	Adjacency Matrix	Iterative merging	Maximal Independent set	Frequent subgraphs
GREW[143]	Single large graph	Sparse graph Representation.	Iterative merging	Maximal Independent set	Maximal frequent subgraphs
ISG [144]	Set of graphs	Edge triplet	Edge triplet extension	TID lists	Maximal frequent subgraphs
SPIN [50]	Set of graphs	Adjacency Matrix	Join Operation	Canonical Spanning Tree	Maximal frequent subgraphs
AGM [52]	Graph database	Adjacency Matrix	Vertex extension	Canonical labeling	Frequent subgraphs

The newly discovered subgraphs are generated in the last call of apriori algorithm are generated by joining two slightly different but somehow similar frequent subgraphs. In step 4, the frequency of these discovered subgraphs is checked. And then is used to generate larger subgraphs in the next iteration.

Much work has been done on finding frequent subgraphs using apriori based approach by proposing various strategies for candidate generation. Some of the popular algorithms are compared in Table 2.2. Inokuchi et al. proposed AGM [52], which generates candidates using vertex-based methods that upsurges the substructure size by single vertex in each iteration. Two frequent graphs of size-*k* are merged only if both of them have the equivalent size, explicitly to be $(k - 1)$. Kuramochi and karypis [141] in 2004 proposed FSG which adopts an edge based method to increase the subgraph size by one edge in each iteration. In FSG, the size of graph is characterized by the number of edges. FSG was further modified to find

frequent patterns in geometric graphs in 2007 by Kuramochi and Karypis[145]. Another algorithm which finds the frequent subgraphs by adding an edge is FARMER [140]. It also follows the level-wise approach of apriori algorithms and finds the subgraph by adding one adjacent edge in all possible manners to find the estimates. FARMER uses queries in the trie data structure to mine frequent patterns.

Another apriori- based algorithm HSIGRAM [146] was proposed which uses iterative assimilation for subgraph generation. The main objective of HSIGRAM was to find the maximal autonomous set of a graph which is created from the embedding's of a frequent subgraph so as to weigh its frequency.

Thomas et al. have proposed ISG [144] algorithm which uses a very distinct method of graph representation. The input set of graphs are transformed to such itemsets which can be represented by using edge triplet such that one edge triplet can be added in each iteration. The frequent itemsets are discovered according to apriori based manner after transforming the itemsets that are also based on apriori. After that, the subsequent frequent itemsets are converted back to the subgraphs.

SPIN [50] does the mining of frequent subgraphs only that are not the part of any another subgraph. In the best case scenario, the size of the graph will be decreased exponentially. This algorithm provides an efficient improvement in performance by providing very good scalability with large graphs. The proficiency of Spin algorithm is tested using benchmarked chemical data. GREW [143] is an algorithm which overcomes all the limitations existing algorithms that does heuristic & complete frequent subgraph discovery. It is used for very large subgraphs to find patterns in connected subgraphs having the huge amount of vertex disjoint structures.

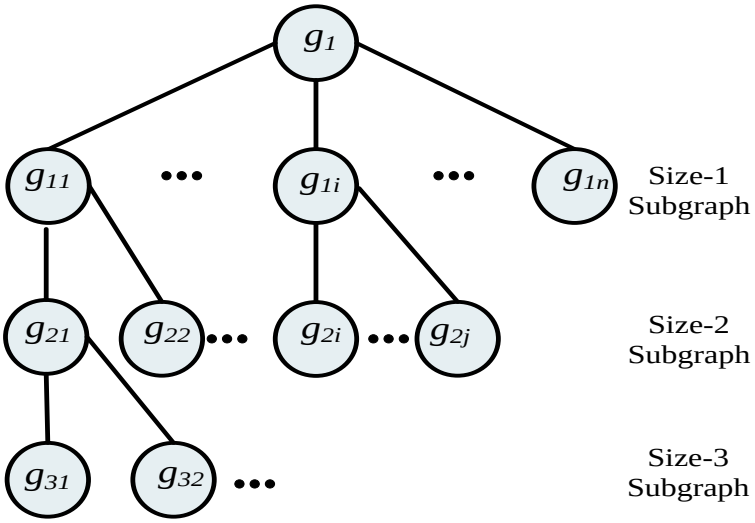


Fig. 2.4 Pattern Growth Approach for frequent Subgraph Mining

Although apriori-based algorithms find the frequent subgraphs efficiently in small graphs, it has considerable overheads in joining two subgraphs with size-*K* to generate candidates of size-(*K+1*). To overcome these overheads, non-apriori based algorithms were introduced which were basically based on pattern-growth approach.

2.3.2 Pattern growth based frequent subgraph mining

In Pattern growth approach, a new edge *e* is added to the existing last candidate to generate a new candidate subgraph *g*. By repeating the same procedure extension is done of frequent subgraphs in all the suitable manners. But, during the generation of frequent subgraph it may be possible that same subgraph is generated many times [147].

The overview of the working of pattern-growth approach is shown in Fig. 2.4. It initializes by extracting every distinct edge from the graph for discovery of frequent size-1 subgraphs. Starting from any size-1 subgraph *g*, the subgraph is extended recursively. An extension is performed by appending from certain vertex of subgraph *g*. In Fig. 2.4, the extension order of subgraph starting from g_1 is $g_{11} > g_{21} > g_{31} > g_{32} > g_{22}$.

The detailed Pattern-Growth based algorithm is illustrated in Algorithm 2.2. Here, the newly formed graph by adding e is denoted by $g\__v\ e$. It performs extensions recursively for each discovered graph g, until all the graphs with g entrenched are discovered.

Algorithm 2.2: PatternGrowth (g, D_G, *min_support*, S_G)

Input: A frequent graph g, a graph Dataset S_G and *min_support.*

Output: A frequent subgraph set S_G

1. **If** $g \in S_G$ **then**
2. | Return
3. **End**
4. **Else** insert g to S_G
5. | Scan D_G to find all edges e that can extend g to $g\__v\ e$
6. **End**
7. **For each** frequent $g\__v\ e$ **do**
8. | PatternGrowth ($g\__v\ e$, D_G, *min_support*, S_G)
9. **End**

The graphs generated by Algorithm 2.2 have some bottleneck in extending the graph as using this algorithm, same graph can be discovered multiple times.

To overcome this bottleneck, each discovered frequent graph should be extended conservatively as possible. This principle resulted in design of many new algorithms such as gSpan [54], FFSM [149], MOFA [30], Gaston [150] and SUBDUE [29]. A comparative study of these algorithms is done in Table 2.3.

gSpan [54] algorithm solves the issue of duplicate subgraph generation by using a right-most extension method, where the extensions are done only on the rightmost path. Beginning from the starting vertex V_1 up to the last vertex V_N, the right most path is constructed straightly according to the depth-first search method on the graph.

Table 2.3. Comparison of Pattern-Growth based approaches

Algorithm	**Input type**	**Graph Representation**	**Subgraph Generation**	**Frequency counting**	**Nature of output**
SUBDUE[29]	Single large graph	Adjacency matrix (Greedy Approach)	Level-wise Search	Minimum description code length	Complete set of Frequent subgraphs
gSpan[54]	Set of Graphs	Adjacency matrix	Rightmost extension	Depth first search (DFS) lexicographic order	Frequent graphs
Close Graphs[148]	Set of Graphs	Adjacency matrix	Rightmost extension	DFS lexicographic order	Closed Connected Frequent graphs
FFSM[149]	Set of graphs	Adjacency Matrix	Merging and extension	Suboptimal canonical adjacency matrix tree	Frequent subgraphs
Gaston[150]	Set of Graphs	Hash Table	Extension	Embedding lists	Maximal frequent subgraphs
TSP[53]	Set of Graphs	Adjacency matrix	Extension	TSP tree	Closed Temporal Frequent sub graphs
MOFA[30]	Set of Graphs	Adjacency matrix	Rightmost extension	DFS Lexicographic order	All frequent subgraphs
JPMiner[151]	Set of Graphs	Adjacency matrix	Rightmost extension	DFS lexicographic order	Frequent jump patterns

In this method each graph is represented using a DFS canonical code. The isomorphism test is done by comparing these codes. This code is written based on the DFS tree. Meanwhile a graph can have numerous different DFS codes; the code with the conditions noted in is called the minimum DFS code in order to produce a unique code.

Huan et al. [149] proposed FFSM algorithm, which employs vertical search arrangement, contained by an algebraic framework for graph. For counting frequency, it uses a sub-optimal adjacency matrix in canonical form. To generate candidate, it make the use of join operation and avoids extensive testing of subgraph isomorphism.

CloseGraph [148] introduced by Yan and Han in 2003 is an algorithm that uses pattern growth approach by finding out looping methods of interest. It does not exactly eliminate unnecessary graphs. Rather than that it increases the mining efficiency up to great extent especially in the case the graph is very large.

SUBDUE [29] finds a subgraph that compress graph G maximally by using the MDL technique. It is a kind of theory according to which for the given set of the data, the best hypothesis will be that which does the best compression. In this method a threshold of similarity is defined which is the cost of changes in one graph to isomorphs it with another one. SUBDUE's uses the MDL principle for searching as specified in equation (1) where *DL*(*S*) denotes the substructure's description length, *DL* (*G*|*S*) represents the description length of the graph as compressed by the substructure, and *DL*(*G*) represent the description length in the original graph. Among all, the one which have highest compression ratio will be the best substructure:

$$Compression = \frac{(DL(S) + DL(G \mid S))}{DL(G)} \qquad (8)$$

Where, *DL*(*G*) is a measure of the minimum number of bytes of information required to represent graph *G*.

Another algorithm named MARGIN [130] is proposed for mining maximal frequent subgraphs. In this, the set of candidate subgraphs that are likely to be maximally frequent are the set of edge frequent subgraphs with a *z*-edge infrequent subgraph. The Margin algorithm computes such candidate set proficiently and locates the maximal subgraphs using the step of

post-processing. It has been proved that in comparison of other pattern growth approaches like gSpan, this algorithm is 20 times quicker for some particular data sets.

Cheng-Te Li et. al [53] have introduced an efficient algorithm named "TSP-algorithm" (Temporal Subgraph Patterns algorithm) for mining the patterns that contain some temporal information to form somehow a connective subgraph. In TSP algorithm, patterns are grown in depth first search method. Heu et al. [151] proposed JPMiner for mining Jump Patterns by integrating them in DFS code tree enumeration framework. The authors claimed that mining frequent Jump patterns can capture interesting patterns despite of reduction in the number of output subgraphs.

2.3.3 Classification and Comparison

The existing subgraph mining algorithms can be classified on the basis of the way to represent graphs, nature of input, search strategy and completeness of output. The classification is shown in Fig. 2.5. The detail of each of the parameter for classification is explained below

Graph Representation: The way of representing graph has a significant and direct influence on the execution time and memory usage in frequent subgraph mining algorithms. The simple ways of representing graph are adjacency list and adjacency matrix. Adjacency matrix uses vertices as entities in rows and columns where an entry in row *i* and column *j* represents a presence of edge connecting two vertices v_i and v_j. Adjacency matrix is relatively easier in implementation but it is not memory efficient in the case if graph is sparse. Most of the existing algorithms uses adjacency matrix for graph representation [3][9]. To find isomorphism in between graphs, it is anticipated to use a reliable labeling strategy so that two similar graphs could be labeled in the similar way irrespective of the order of the representation of vertices and edges.

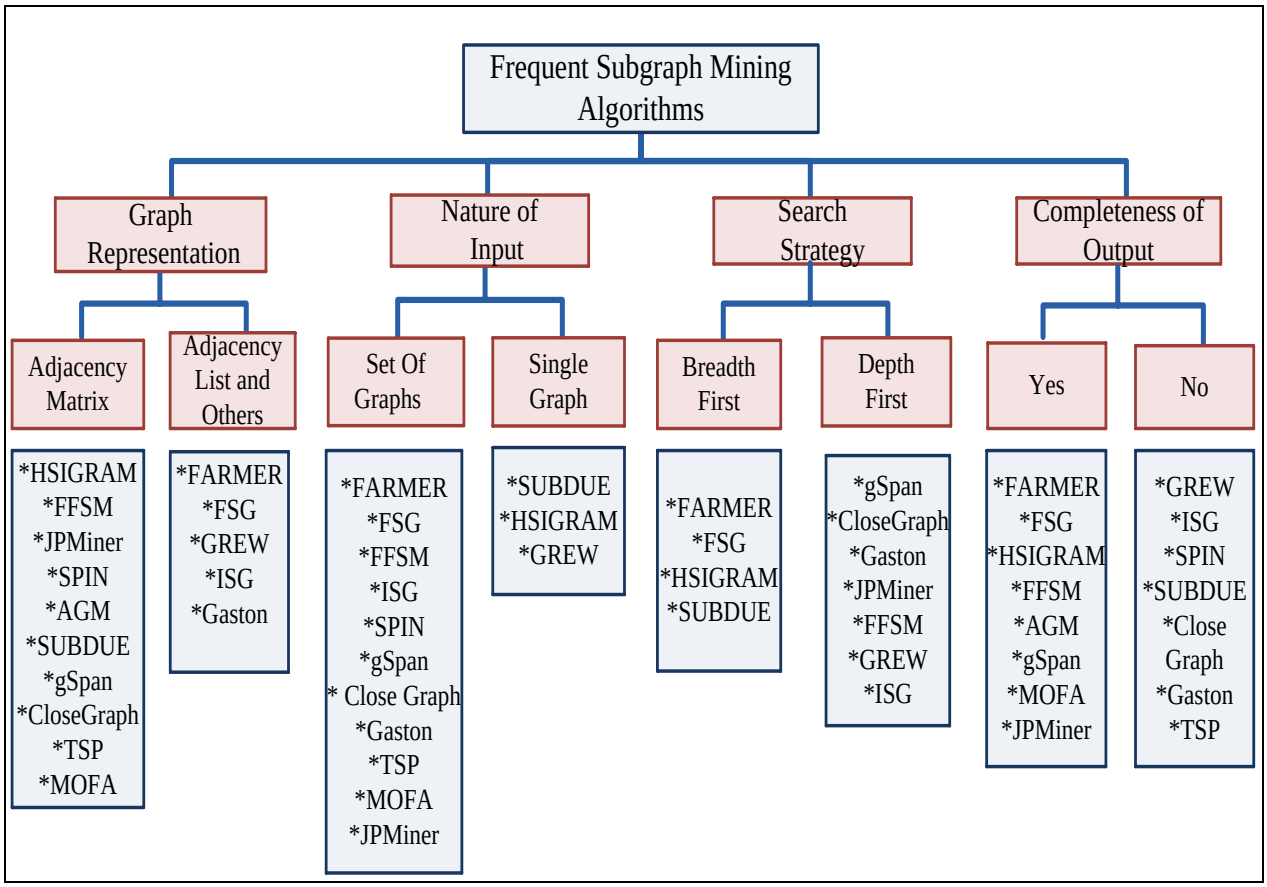


Fig. 2.5 Classification of Frequent Subgraph Mining Algorithm

The representation of graph as adjacency list consumes less memory in comparison. Hence, the algorithms that use adjacency list are somewhat memory efficient [141] than others. FARMER [140] uses trie data structure for graph representation which is a tree data structure where candidates are discovered using queries from parent nodes to other vertices. Other than that, Gaston [150] uses Hash table for graph representation.

Nature of Input

Another major feature that a frequent pattern mining algorithm for graphs possess, is the nature of the input. An algorithm can either take a set of graphs as input known as transactional graph or can take a single large graph as input. While finding frequent subgraphs from graph transactions, the similarity is discovered between two or more graph from a set of graph transactions. However, for discovering similarity in a single large graph,

node and edge similarity are considered mainly. For example, in the case of social networks, telephone network, the graph is a single large graph and mining frequent patterns from it is more tedious than mining from graph transactions. The example of graph transactions is in the case of molecular databases such as in MOFA [30], chemical compound database [29],[13], [20], medical datasets etc. While, SUBDUE [29], HSIGRAM [142], GREW [143] falls in the category where nature of input is a single graph.

Search Strategy

Two types of search strategy are considered for comparison, Breadth first (BFS) and Depth first (DFS). BFS searches level by level so discovers all the frequent subgraphs those are above threshold or *min_support*. While DFS suffers from redundancy as same subgraph can be discovered multiple times. But, DFS consumes less memory then BFS because the number of list stored in DFS is proportional to the depth of the graph while in BFS it is proportional to the width of the graph.

The algorithms that use BFS are SUBDUE, HSIGRAM, GREW and FSG. The bottleneck of these algorithms is that they have more computational time than the algorithm that uses DFS such as gSpan, CloseGraph, Gaston, JPMiner, FFSM, GREW and ISG. The Classification is shown in detail in Fig. 2.5.

Completeness of Output

The classification under this category can be done into two subgroups, complete and incomplete. The first one generates a complete set of frequent subgraphs, while the later do not find the complete set. In certain cases, only a subset of frequent subgraphs are required rather than the whole set. Also, it is not practical to mine a whole set of frequent patterns in large subgraphs like those of social networks. While in some examples, it is likely to find the

complete set of frequent subgraphs. Hence, there is a trade-off between completeness and performance depending specifically on the application.

SUBDUE produces incomplete set of frequent subgraphs as it uses greedy search without involving backtracking, thus cannot discover all frequent subgraphs. Later, a heuristic algorithm GREW was proposed which also cannot mine complete set of frequent subgraphs. ISG, SPIN, CloseGraph, Gaston, TSP all falls in this category. Many algorithms mine all the frequent subgraphs including FARMER, FSG, FFSM. JPMiner, MOFA, gSpan and HSIGRAM. FARMER follows an inductive logic programming approach and mines all frequent set of subgraphs, but is inefficient in terms of computation. FSG, gSpan are more efficient then FARMER but are still inefficient in terms of computation when compared to the other category.

2.3.4 Frequent subgraph mining in large graphs

The existing research considers only sparse graphs for mining [152], [153]. But, the real world graphs are dense and have high average degree. The first algorithm which finds the frequent subgraphs from large dense subgraph is GRAMI [154]. It uses constraint satisfaction problem for finding the minimum set of instances for satisfying support threshold. AGRAMI is proposed by extending GRAMI to find approximate subgraphs. However, both of them are centralized algorithms that cannot handle very large graph. For handling the large graph in distributed environment, many subgraph mining algorithms using Map-Reduce were proposed [57], [155]. A density based partitioning technique was proposed for large scale subgraph mining using Map-Reduce framework for balancing computational load [156]. Other subgraph mining algorithms for single large graphs are ScaleMine [157], FSM-H [75], Arabesque [158], MRFSE [159], gApprox [136] and pegi [160]. However, all the aforementioned research performs expensive computation.

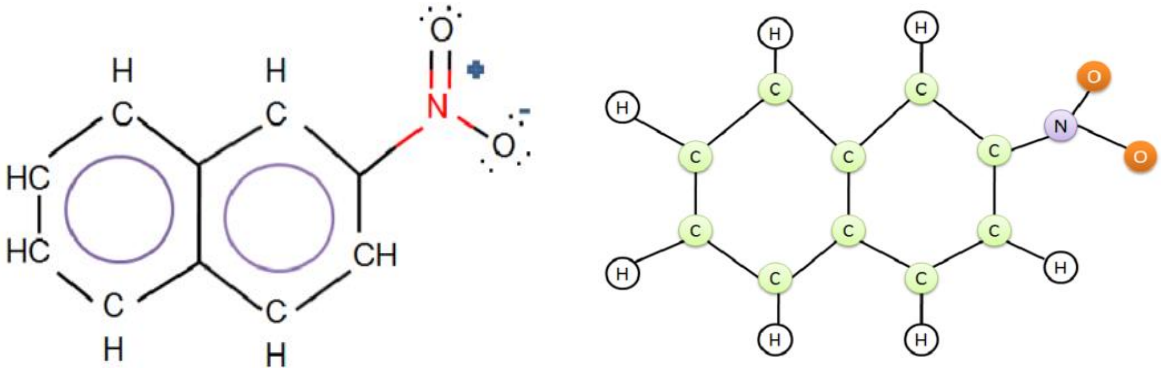


Fig. 2.6 Graph representation of a chemical compound (a) A chemical compound, (b) A graph representation of a chemical compound

To reduce the computational cost, FSM is applied on the representative graph sample. Many graph sampling algorithms were proposed in literature [161]–[164]. However none of them were applied for the task of subgraph mining and none of the existing

2.4 Challenges of mining large graphs

Mining graph data has many applications. Each application has different challenges. Few of the are discussed in this section:

2.4.1 Social Networks

Social network can be a friendship network [165], [166], [167]or a collaboration network [168] . In friendship network, a group of friends can be called communities or adding new person to closely related communities. Collaboration network consists of co-authorship network or business network like “who buy X, will also by Y”. Finding such relationships have many challenges as people may start new employment or make new collaboration or friends. Such relations are highly susceptible to mutate.

2.4.2 Chemical Networks

Chemical data is frequently denoted as graphs in which the vertices correspond to atoms, and the links correspond to the bonds between atoms as shown in Fig. 2.6. In certain cases, individual vertices can be used as chemical patterns. In such cases, the individual graphs are

small in size, however there are substantial repetitions among the different nodes [169]. This leads to isomorphism challenges in applications such as graph matching. The challenge is for discovering isomorphism as the vertices in a given pair of graphs may match in a variety of ways [145], [170], [171]. Thus, the number of possible matches can be exponential in terms of the number of the vertices.

2.4.3 Biological Networks

The vertices are typically prudently designed sections of the biological models. A typical example of a vertex in a DNA application could be of an amino-acid. A single biological network could easily contain thousands of vertices [172], [173]. The sizes of the overall database are also large enough that it cannot be fitted in a memory of a single system. The disk-resident nature of the data set often leads to unique issues, which are subsequently highlighted in the scenario of biological networks.

From a view-point of the computer science, the protein structure can be considered as a set of elements in which each element can be an amino acid residue, an atom or a secondary structure fragment. Several graph representations are developed for preprocessing of protein structure, ranging from coarse representations in which each vertex is a secondary structure fragment to fine representations in which each vertex is an atom. Certainly, a protein interaction network can be represented by a graph where an edge links a couple of proteins when they participate in a particular biological function.

2.4.4 Computer Network and Web Data

In the case of computer networks and the web, the number of vertices in the underlying graph may be massive. This can lead to a very large number of distinct edges [174], [175]. Handling such large dense graphs is hard to hold in the available storage space. Thus, various approaches need to be designed for summarizing and working with compressed

representations of the graph data sets. In certain applications, the edges in the graph may be in the form of a data stream. In such cases, a second challenge arises from the fact that it may not be possible to store the incoming edges for future analysis in disk-resident processing frameworks. Therefore, the summarization techniques are especially required for the aforementioned case.

3. PROBLEM FORMULATION

This chapter lists the research gaps that are identified initially, while proposing the research objectives. These gaps identify the problems with the existing graph clustering algorithms. They point out the fact that there is a need to work on developing approaches that are based on searching patterns in large graphs that cannot fit in the memory of a single system, and applying them for developing applications that require scalability. Based on these gaps, there are four research objectives that are addressed in different chapters of this thesis.

3.1 Research Gaps

After comprehensive survey, following research gaps are identified:

- Most of the existing algorithms do not provide scalability up to numerous millions and billions of vertices and edges. In addition, the existing work relies on a shared memory model, which confines their capability to handle very large and disk-resident distributed graphs.
- Clustering methods that are able to explore and structure large graphs are highly desirable. But, there is no parallel structural clustering algorithm that can handle large and distributed databases in the form of graphs.
- Most of the existing clustering algorithms for graphs are sensitive to the selection of initial centers and require the number of clusters to be mentioned in the beginning. But for real-world large graphs, structure of the network plays an important role in determining the number and size of clusters. Consequently, if the number of clusters in such networks is predefined, the actual clusters might break up or get merged. It may also

result in creation of clusters even when the data cannot be clustered actually. Such kind of instability is highly undesirable in real-life applications and thus, should be avoided.

- Most of the existing work focuses on graph transactions for performing FSM. But, finding frequent subgraphs from a single massive graph is far more challenging because it involves higher computations in evaluating the support and in performing NP-Complete subgraph isomorphism.

3.2 Research Objectives

The objectives of the proposed work are given below:

1. To better understand and explore various concepts, techniques and tools available for mining of graphs.
2. To propose improved algorithm(s) of pattern mining for big data using graphs.
3. To implement the proposed algorithm(s).
4. To verify and validate the proposed algorithm(s).

3.3 Methodology for achieving the objectives

In an effort to accomplish the stated objectives, the following steps are followed. The workflow of steps followed is given in Fig. 3.1. The work done to achieve each one of the objectives is given below:

Objective 1:

- Identified the approaches used for graph pattern mining such as graph clustering and frequent subgraph mining
- A comprehensive review of various algorithms and graph mining methods which can be used to mine big data sources is done. Their pros and cons are also identified.

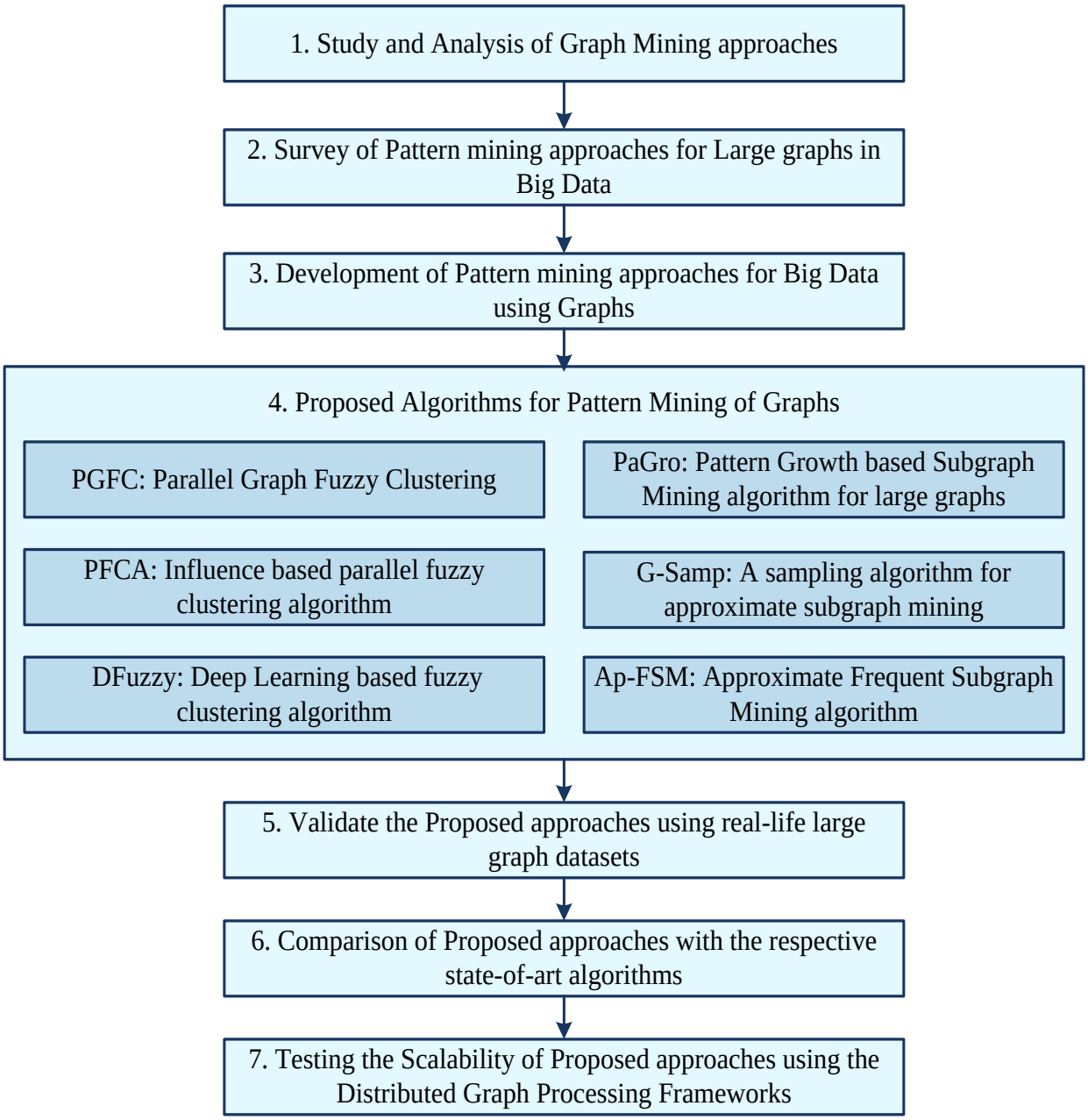


Fig. 3.1 Flow of the Work Done

- Identified the tools and frameworks that can handle large graphs efficiently.

Objective 2:

- Proposed PGFC (Parallel Graph Fuzzy Clustering) algorithm by amending the structure of the classical Fuzzy C-means algorithm for large graph data. For the initialization of cluster centers, degree centrality is considered. BSP based Pregel model is followed to design the distributed version of the algorithm.

- Proposed PFCA (Parallel Fuzzy Clustering algorithm) which finds the number of clusters by analyzing the structure of the network on the top of Hadoop framework using Pregel. PFCA avoids multiple iterations of finding suitable cluster centers by considering the influence of vertices for initialization of cluster centers. Graph structure is analyzed to provide robustness using random walk based PageRank and is later modified using modularity to provide high precision.
- A Deep learning based model named 'DFuzzy' is proposed. First, DFuzzy 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.
- Two pattern-growth based frequent subgraph mining algorithms are proposed named Ap-FSM and PaGro. Ap-FSM is approximate FSM algorithm which uses sampling for fast graph processing. Whereas, PaGro is an exact FSM algorithm that finds all frequent subgraphs, optimizes graph isomorphism and support computation in distributed environment.

Objective 3:

- The algorithms are implemented on 8 dell machines each with 8GB of RAM, 1024 GB hard disk, having Ubuntu Linux OS installed, forming a Hadoop cluster. All the machines are connected via a gigabyte network. One of the nodes is considered as the master node, and the rest are considered as slave nodes.
- PGFC, PFCA and DFuzzy are implemented using Pregel based Giraph framework on top of the Hadoop cluster. Java is used for the implementation.
- Ap-FSM and PaGro are implemented in Java and R. Both are implemented using Giraph framework on top of the Hadoop cluster.

Objective 4:

- The performance of the proposed PGFC, PFCA and DFuzzy algorithms are evaluated using real and synthetic graph datasets. The efficiency is proved by comparing their performance with the state-of-art graph clustering algorithms such as K-means, Fuzzy C-mean, Pregel based semi-clustering algorithm.
- Tested Ap-FSM and PaGro using real life graph datasets and evaluated the performance in terms of processing time, memory overhead and number of supersteps.
- The data scalability of the proposed algorithms is tested by considering datasets of different sizes.
- The node scalability of the proposed algorithms is tested in terms of Speed-up by varying the number of Worker nodes.

4. CLUSTERING IN LARGE GRAPHS

In this chapter, the approaches proposed for graph clustering approaches of large graphs are explained. The existing clustering algorithms lack scalability and consistency. First, the background and preliminaries are given in this chapter. Then, a brief overview is provided of tradition Fuzzy Clustering in section 4.2. Further, the working of the proposed fuzzy graph clustering algorithms is given. The working of proposed Parallel fuzzy clustering algorithm (PGFC) algorithm is elaborated in section 4.3.1. The working of PFCA is explained in section 4.3.2. At last the working of deep-learning based proposed graph clustering model named DFuzzy is given in section 4.3.

4.1 Background and Preliminaries

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 $E \subseteq V$ x *V*. Set *V* contains the vertices and set *E* defines the structure of graph. For gaining deeper insights from a graph, clustering techniques are used.

4.1.1 Clustering in graphs

A graph cluster is generally anticipated as a connected component having similar vertices within the cluster and few edges between clusters. Similarity function is demarcated using the confined 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 $G = \{v_1, v_2, ..., v_n\}$ is an assignment of vertices v_i to k cluster sets C_k such that $\bigcup_{j=1}^{k} |C_k| = |C|$ and $C_i \cap C_j = \emptyset$ for each cluster pair i and j. Therefore:

$$\sum_{j=1}^{k} |C_j| = |C| \qquad (9)$$

A partition is denoted as C = { C_j }.

Definition 2: A soft clustering of a graph $G = \{g_1, g_2, \ldots\ldots, g_n\}$ 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:

$$\sum_{i=1}^{n} \sum_{j=1}^{k} \mu_{i,j} = |C| \qquad (10)$$

The actual overlapping in fuzzy clustering is determined by the number of clusters and selected membership function. And the appropriate selection of initial cluster centers reduces the time complexity. In a graph, each vertex has a varying degree. The vertices with high degree have relatively more connections than other vertices. So, vertices with high degree are appropriate candidates for cluster heads.

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

In the proposed algorithms, 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.

The problem of Fuzzy clustering for large graphs is considered in this chapter, 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, vertex set V is partitioned into overlapping clusters set C = (C_1,C_2, ... , C_k) such that *cost*(C_i) is minimized.

4.2 Fuzzy Clustering: Overview

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 the same cluster. It was initially developed by Dunn in 1973 and further upgraded by Bezdek (1981).

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, ……., v_N} into C ={c_1,c_2,c_3, ...,c_k} fuzzy clusters by assigning some degree specified by a membership value within the interval [0, 1]. FCM is based on minimizing the squared error objective function in accordance to the distance and the membership value.

$$J = \sum_{i=1}^{N}\sum_{j=1}^{C}(U_{ij})^m \| v_i - c_j \|^2 \qquad 1 \leq m \leq \infty \qquad (11)$$

Where m is the fuzziness index of value greater than 1 and U_{ij} is the membership of v_i in the j^{th} 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 iteratively as described in Algorithm 4.1. Convergence is reached when the edge cut is less than threshold ε.

Algorithm 4.1: 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:

$$\mu_{ij}^{k} = \frac{1}{\sum_{k=1}^{c} \left\{ \frac{x_i - c_j}{x_i - c_k} \right\}^{\frac{2}{m-1}}} \qquad (12)$$

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

$$c_j = \frac{\sum_{i=1}^{n} (\mu_{ij}^{k})^m . v_i}{\sum_{i=1}^{n} (\mu_{ij}^{k})^m} \qquad (13)$$

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

$$\forall i; j : \max_{ij} \left\{ \mu_{ij}^{k+1} - \mu_{ij}^{k} \right\} < \in \qquad (14)$$

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

4.3 Proposed graph clustering algorithms

In this section the proposed clustering algorithms are elaborated. In real-life graphs, clusters may overlap. Thus, three algorithms are proposed for finding fuzzy clusters in large graphs namely PGFC, PFCA and DFuzzy. PFCA is proposed to overcome the limitations of proposed PGFC. DFuzzy is proposed to overcome the limitations of both PGFC and PFCA.

4.3.1 PGFC: Parallel Graph Fuzzy Clustering

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 is 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. 4.1. 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 zero as their initial value. 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, thus is more important in comparison to other vertices of the network. All the vertices are sorted in the descending order of their degree. FCM [37] 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 the 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 communicate through messages to update cluster centers and membership values of each vertex corresponding to clusters iteratively, until the convergence 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 may result in the inaccurate output as a vertex can be part of more than one cluster.

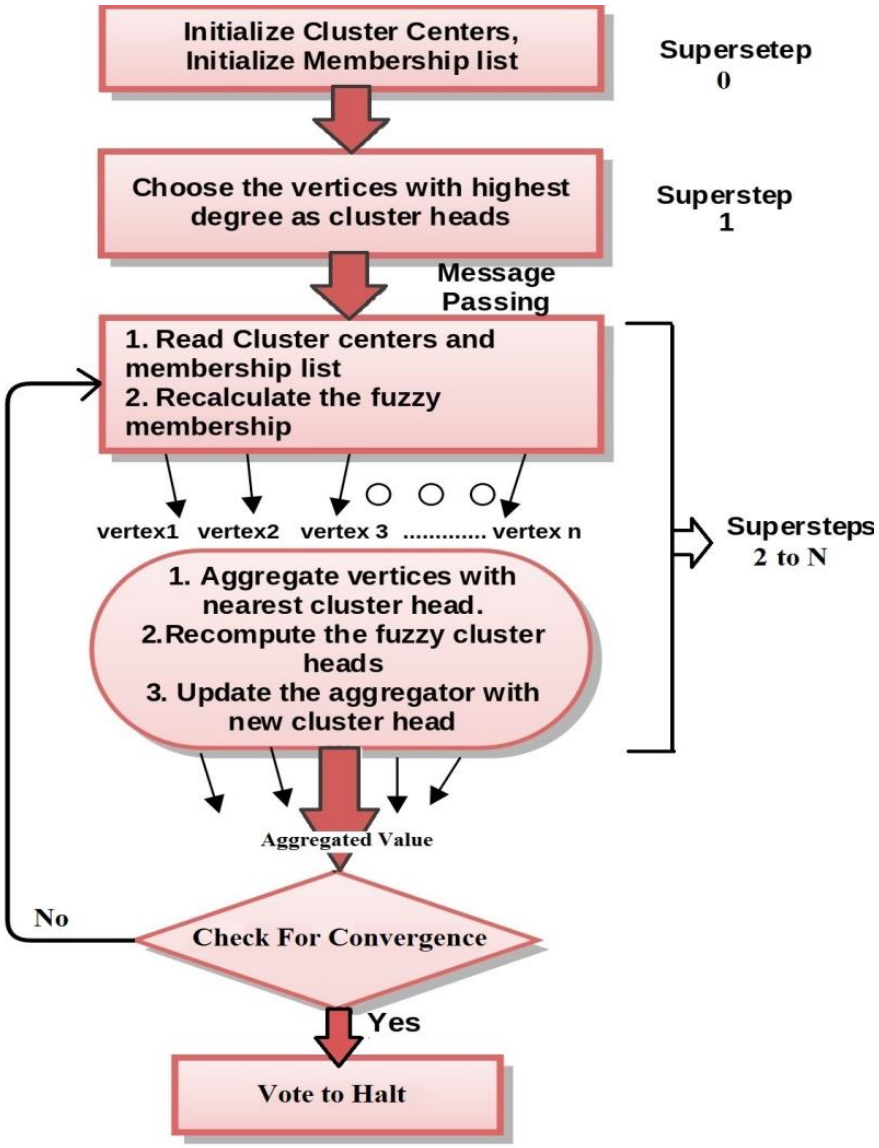


Fig. 4.1. Flow Chart of PGFC Algorithm

PGFC maintains the native graph structure and minimize the communication overhead incurred 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 4.2.

Algorithm 4.2: PGFC Algorithm

Input: Graph G

Output: List of Vertices with the final membership values corresponding to final cluster centers

```
1.  Function Compute(msg,superstep)
2.      If (this.getSuperstep( ) == 0)
3.          initialize the cluster centers
4.          initialize the Membership_Matrix
5.      End
6.      If (this.getSuperstep( ) ==1)
7.          initial clusters = vertexValue[k]      // From Algorithm 4.3
8.          getValue(initial_centers)
9.          getValue(membership_matrix)
10.     End
11.     Else
12.         VertexCompute( )                       // From Algorithm 4.4
13.         For each Message: msg do
14.             // compute the new clusters centers using equation (12)
15.             newClusterCenters[K]
16.         End
17.     End
18.     If (getSuperstep( ) > maxIteration | | J = min || max_ij {μ_ij^(k+1) − μ_ij^k} <∈
19.         printFinalClusters_Centers( )
20.         haltComputation( )
21.     End
22.     Else
23.         // update the aggregator with the new cluster centers
24.         setAggregatedValue(newCluster_Centers)
25.         Send msg(newCluster_Centers , updatedMembershipMatrix) to all neighbours
26.     End
27. End function
```

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.

Algorithm 4.3: 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.          End
7.      End
8.      If (this.getSuperstep( )  > 1)
9.          Long sum = 0;
10.         For ( msgs->Next( ) )
11.             Sum++
12.         End
13.         vertexValue = getValue( )
14.         vertexValue.set(sum)
15.         setValue(vertexValue)
16.         Mergesort(vertexValue, msgs)
17.     End
18.     voteToHalt( )
19. End Function
```

In further supersteps, the cluster centers are recomputed using equation (12) until the value of *J* mentioned in equation (14) is minimized or the convergence criteria mentioned is fulfilled.

A. 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, first the degrees of all the vertices using a standard in-degree and out-degree algorithm in giraph are calculated. 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 4.3.

Algorithm 4.4: VertexCompute

```
1. Function VertexCompute(Vertex<Long,Double,Float>, Message <Double>)
2.     ReadClusterCenter( )
3.     For each ClusterCenter do
4.         // Calculate the Fuzzy Membership using equation (13)
5.         Membershipfunction( Distance, fuzzinessIndex)
6.     End
7.     Membership = max(Membership[K])
8.     Aggregate(Vertices , Membership)
9. End Function
```

K vertices with highest range of degree are considered as the initial cluster heads. In the proposed approach, an assumption is taken that two cluster heads must not be the direct neighbour of each other.

Thus, if two adjacent vertices are selected as cluster head on the basis of degree, then the vertex with larger degree is considered as cluster head and the second one is simply ignored.

A. Finding Membership values

Algorithm 4.4 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 equation (13). The calculation is performed based on the updated potential minimum distance from the cluster center using the messages they receive from the neighbours.

All the vertices pass their membership values 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 possible shorter distances between the vertices and the cluster center.

4.3.2 PFCA : Influence based Fuzzy Clustering algorithm

The proposed approach finds the overlapping among clusters in large complex networks using parallel BSP based Pregel framework. PFCA considers the number of edges incident on the vertex as the criteria to calculate its importance in the network. The detailed functioning of the proposed algorithm is shown in Fig. 4.2.

The proposed algorithm is divided into four phases: Cluster Head Selection, Cluster Expansion, Optimization and Filtering. In the first phase, the candidate cluster heads are selected based on their prominence in the network. In Cluster Expansion phase, the clusters are formed by following a random walk starting from cluster head. The clusters are expanded using personalized PageRank algorithm. Further during optimization phase, the cluster heads and the formed clusters are optimized based on distance measure and modularity respectively.

In filtering phase, the fuzzy vertices are identified from formed fuzzy clusters so that the degree of membership can be calculated corresponding to their respective clusters. The whole computation in PFCA is performed on Pregel by repeating two tasks iteratively: vertex value update and merge update. The vertex value update is performed by worker nodes for updation of the value associated with each vertex. The vertex value consists of its *vertex_id* with the *cluster_id*. After the filtering phase, membership values are also added to the vertex's value in vertex value update phase. The master node manages the list of all the vertices allocated to the worker nodes. It gathers the data from all the workers in each superstep in the form of messages and after aggregation and updation, passes the data to corresponding worker nodes. The detailed phase-wise working of the proposed PFCA is described below:

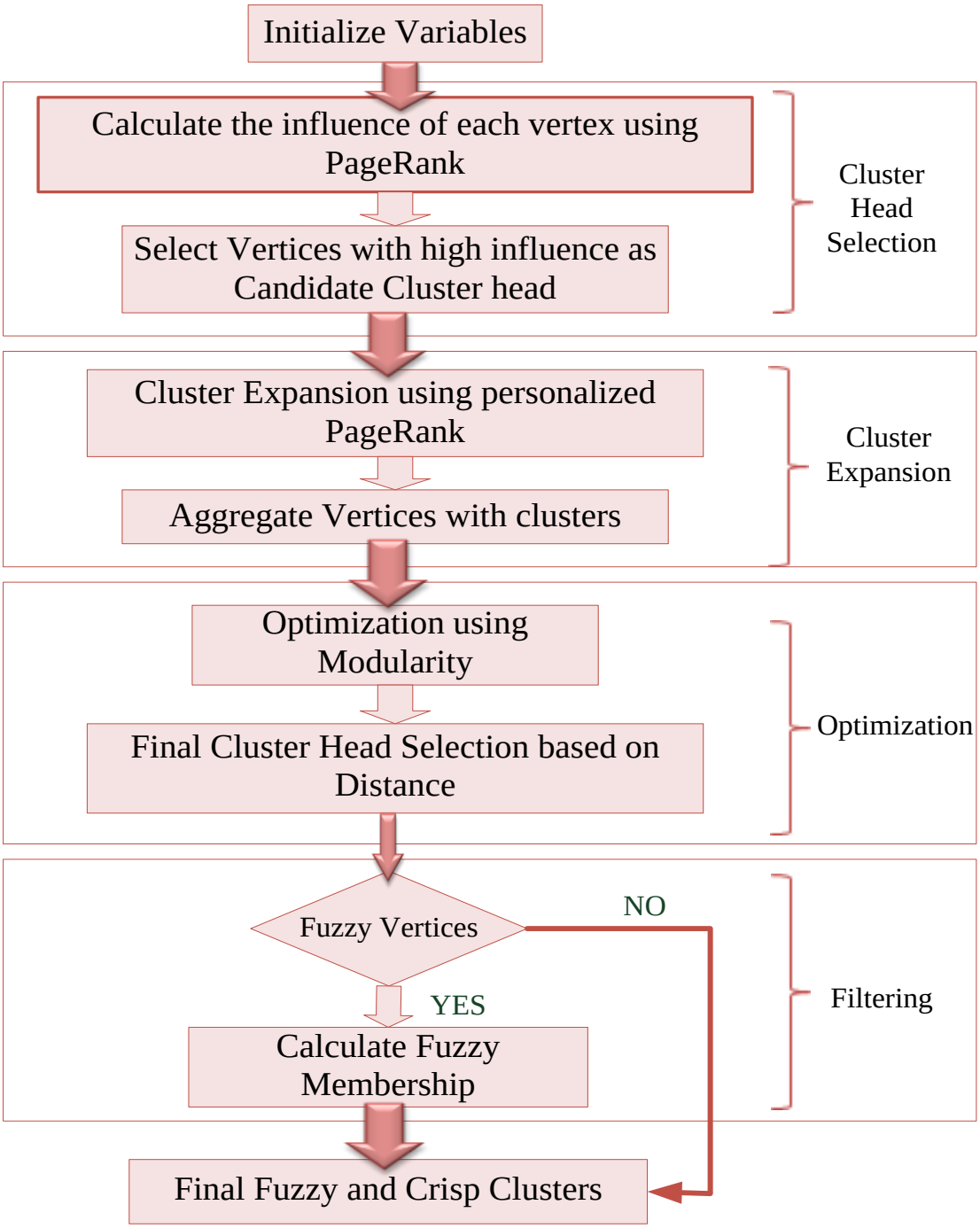


Fig. 4.2 Working of the Proposed Algorithm

A. Phase 1: Cluster Head Selection

Cluster heads act as representatives of the whole cluster. FCM initializes the cluster heads by picking random points from the data. Initializing them randomly from the input gives sub-optimal results. To overcome this shortcoming of FCM, the proposed algorithm starts with

the selection of the vertices which have more prominence than other vertices in the network using the popular PageRank algorithm. The algorithm starts by initializing the vertices of the network with a value *1/N* at superstep 0. Vertices then try to learn their importance in the network by considering the structure of the network. Each vertex passes its tentative PageRank value divided by the number of incident edges to each outgoing edge. The process starts from superstep 1 where each vertex adds the values received via messages into Sum and forward its own PageRank value by calculating (1-α) */N +α*Sum*. After attaining the maximum limit of supersteps or when the PageRank values stops changing, all vertices stop passing messages and vote to halt. The vertex with higher PageRank value has more incident edges in comparison to others, so is considered as an important vertex of the network. The pseudo code of cluster head selection phase is presented in Algorithm 4.5.

Algorithm 4.5: *Cluster Head Selection*

```
Input: Complex Network G(v,e)
If superstep=0 then
    Vi = 1/N;        // Initialization of vertex value
End
Else
    While (superstep ≤ 25) do
        Superstep+1;
        While vi←Msgi do
            Sum = Sum(messages);
            p= (1-α) /N +α*Sum;
            vi.setvalue(p)
        End
        sendMsg(value)
    End
End
While (superstep < 30) do
    While v← Msgi do
        MergeSort(v.list, Msgi)
    End
    Send (Msgi → v.list);
End
```

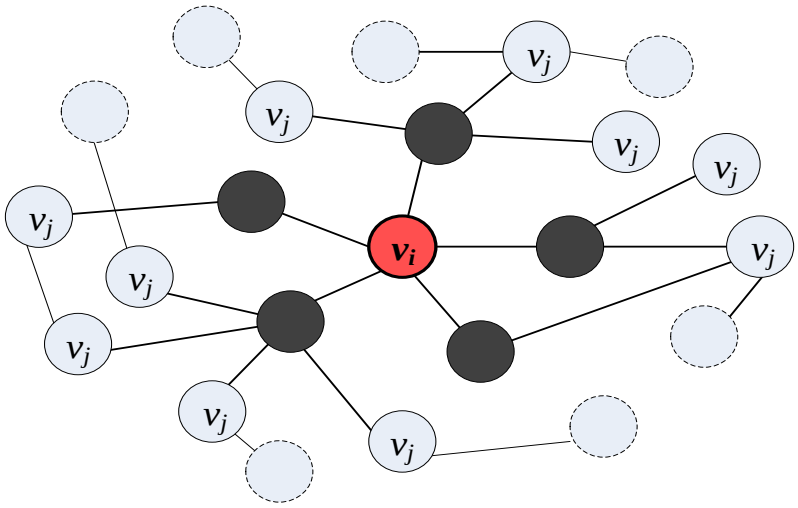


Fig. 4.3 Cluster head Selection: Direct neighbours of v_i are not considered

The vertices are sorted in the decreasing order of their PageRank value. Top 10% vertices from the sorted list of vertices according to their PageRank 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 discarded from the candidate cluster head list.

Fig. 4.3 illustrates one of such scenario. Vertex v_i and one of its adjacent neighbours are in the candidate cluster head list. Randomly, vertex v_i is selected and the adjacent vertex with same influence is discarded.

A. Phase 2: Cluster Formation

Once a list of cluster heads is ready, the process of node assignment can be started. The vertices with high PageRank values are selected as the candidate cluster heads in the previous phase. The node assignment for the expansion of cluster is done by using personalized PageRank algorithm starting from the selected centers.

The algorithms starts lazy random walk from cluster head c_i and visit the neighbour 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 4.6. The upper limit of number of iterations is set to 40 in order to avoid computational overhead. A cluster C_a comprises the center c_{a0} having high PageRank value with several other vertices ($v_{a1}, v_{a2}, \ldots\ldots\ldots, v_{ap}$). The prominent cluster head spread the influence to its neighbour vertices in multiple iterations. Closer the vertex to cluster head more is its influence. The cluster center c_i transmits its impact with label information to its adjacent neighbours. The neighbours pass their impacts to their adjacent neighbours in further iterations. In each iteration, vertices classify the received messages according to the label. In further iteration, each vertex generates new messages and sends them to their neighbours. The process continues till the mentioned number of supersteps is reached. Finally, the sum of messages having same label is calculated by adding the number of edges in a random walk $(c_{a0}, v_{a1}, v_{a2}, \ldots, v_{ap})$ using $Sum_a = \sum_{i=1}^{msg_n} Edge_a$, where msg_n is the number of messages with label a.

Algorithm 4.6: *Cluster Formation*

```
Input: Selected Cluster heads C[k]
1   While Superstep ≤ 40 do
2       Superstep + 1;
3       If v←Msg_i do
4           IF v_i == c_i do
5               Sum_i =0;
6               value = (1- α) x v_i + α * Sum_i;
7           End
8           Else
9               Value = α * v_i * Sum_i ;
10          End
11          V.label = label_i
12          SendMsg(value)
13      End
14      Else
15          VoteToHalt( );
16      End
17      SendMsg( Clusters);
18      Aggregate(vertices, Cluster_Head)
19  End
```

Algorithm 4.7: *Phase 3: Optimization*

Input: Formed clusters with cluster head set $C[k]$

```
1  While Superstep ≤ 60 do
2      Superstep + 1;
3      For each c_i ∈ C[k]
4          While Modularity(c_i)≤threshold do
5              Vertices(c_i) → neighbours(c_i)
6          End
7      End
8      While C[k] do
9          For each v_i ∈ c_i do
10             minDistance = min(Msg)
11             SendMsg(c_i , minDistance)
12         End
13         minDistance(c_i) = c_j
14     End
15     Aggregate( c_j , Vertices)
16 End
```

B. Phase 3: Optimization

The proposed algorithm is further optimized to produce better clusters with the best selected cluster heads. In this case also, the convergence criterion is considered as the maximum number of iterations. Optimization phase consists of the two steps.

(a) Fuzzy Modularity Optimization

The formed clusters are assessed by the crispness and fuzzyfication of the modularity function Q *and* Q_{fuzzy} from equation (6) and equation (7) respectively. The cluster with low modularity tends to have more edges outside the cluster in comparison to the edges within the cluster. The optimization process is accomplished in an iterative manner by arranging the generated clusters in descending order of 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 any cluster with low modularity is encountered by the master, it informs the worker to allocate the vertices (v_1,v_2....,v_n) of such clusters to other neighbouring clusters to which vertices are connected.

Workers re-assign all such vertices to neighbouring clusters and report the master with the new list of altered clusters. Thus, the clusters with lower modularity are removed from the cluster list.

(b) Updation of Cluster Head based on Distance

Up to now, the distance impacts from the cluster head have been neglected. The selected candidate cluster heads have high influence and their impact does not change if one walk from neighbour vertex to any other vertices in the cluster. Its influence will remain the same. Thus, the distance of vertex v_i from center c_k does not matter. This does not lead to better clustering as some vertices in the cluster could be far away from the cluster heads. The accurate membership of those vertices towards the corresponding cluster cannot be calculated.

To deal with this problem, the distance should be taken into consideration. Farther the vertex v_i from the center c_k, lesser is the influence of c_k on v_i. To better estimate the impact of cluster head on all the vertices of cluster on the basis of distance, equation (12) is used in algorithm. 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 conveying others about their distance from the cluster head. Worker nodes use combiners to minimize the number of messages.

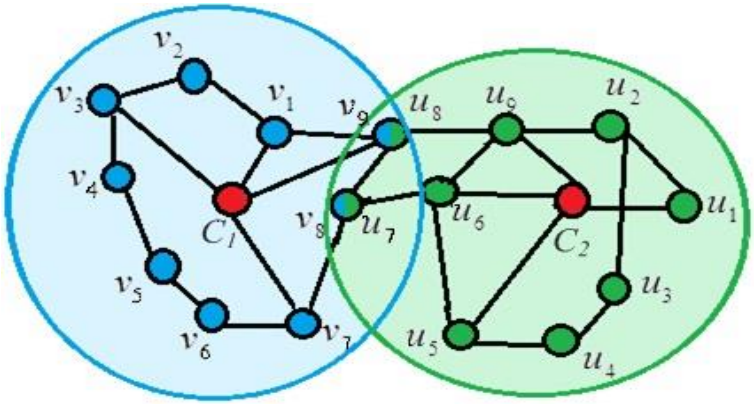


Fig. 4.5 Fuzzy Clusters formed using personalized PageRank

So, each vertex needs to evaluate only some part of the received message. In the next iteration, worker nodes selects the new cluster heads for all the clusters using equation (12). Here for now, the membership value μ_{ij}^{k} is taken as 1 by default. The value of μ_{ij}^{k} is calculated in the succeeding phase.

The iterative computations performed by FCM using equation (12) and equation (13) for finding cluster heads and membership values are unrealistic in case of very large complex networks and thus, are avoided in PFCA. Instead, PFCA detects fuzzy clusters with membership values of vertices towards clusters in a fast single-pass manner. No multiple iterations over the data are performed in PFCA.

C. Phase 4: Filtering: Identifying Fuzzy vertices

In the formed cluster, a vertex v_i can be a part of more than one random walks initiated from different cluster centers as shown in Fig. 4.5. Vertices with label v_8 and v_9 from random walk $(C_1, v_1, v_2, \ldots\ldots, v_9)$ and vertices with label u_7 and u_8 from Random walk $(C_2, u_1, u_2, \ldots\ldots, u_9)$ are same. The two clusters with cluster head C_1 and C_2 overlap and form fuzzy clusters with two fuzzy vertices v_9/u_8 and v_8/u_7 respectively.

Algorithm 4.8: *Identifying Fuzzy and Crisp Vertices*

Input: Formed clusters $C[k]$

```
1  While Superstep ≤ 70 do
2      Superstep + 1;
3      While v←Msg_i do
4          For c_i ∈ C[k] do
5              For each vertex v_i ∈ c_i do
6                  If v_i ∈ (c_i....c_f) then
7                      \\ For fuzzy clusters
8                      Membership(v_i)→(c_i....c_f) //using equation 13.
9                  End
10                 Else
11                     \\ For crisp clusters
12                     Membership(v_i) → 1
13                 End
14             End
15         End
16         Aggregate (Membership(μ_ik), v_i)
17     End
18 End
```

Master node identifies such vertices while performing merge update and instructs workers to calculate the membership of fuzzy vertices towards the connected clusters. The detailed process is given in Algorithm 4.8. The membership values can be achieved in few iterations. Thus, the maximum number of iterations is considered as 70. In each iteration, workers calculate the membership value μ_{fk} of fuzzy vertices v_f towards the connected clusters c_k using equation (13). Manhattan distance $d(v_i,v_j)$ is used to compute the distance which is equal to $\sum_{k=1}^{c}|v_{ik} - v_{jk}|$.

The Manhattan distance is used in PFCA because it is not a squared function and is less sensitive to noise.

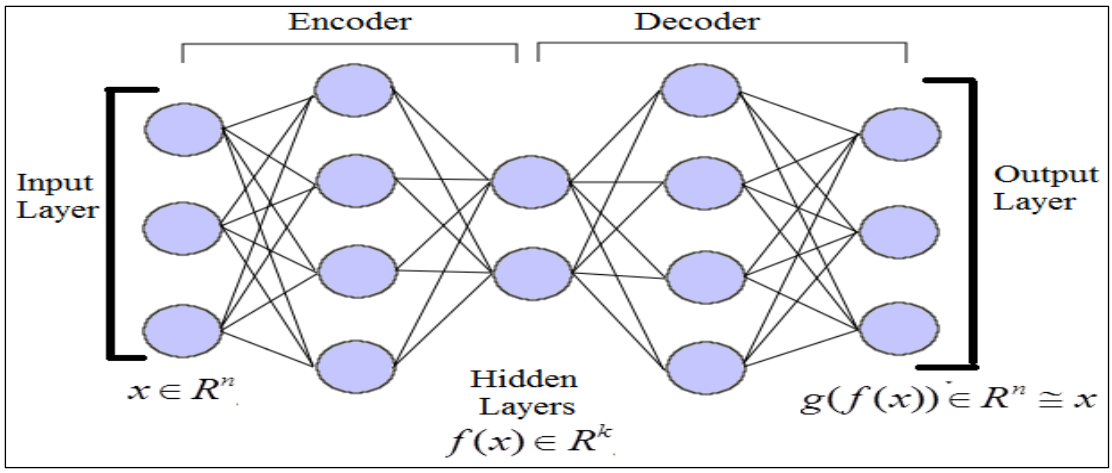


Fig. 4.6 Autoencoder in Deep Neural Network

4.3.3 DFuzzy: Deep Learning Model for Graph Clustering

Deep learning is not something new. It originated way back in 1960s. 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 towards the growing business glitches.

A. Autoencoders

For performing unsupervised learning in deep architectures, autoencoder play a vital role in transfer learning and other tasks. Autoencoder in deep neural networks are responsible for extraction and embedding of features in an unsupervised manner. Fig. 4.6 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 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.

B. 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 are the building blocks of the DFuzzy. All operations are performed using stacked autoencoder by providing greedy layer wise training. The use of autoencoder for graph clustering is shown in Fig. 4.7 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-tune 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.

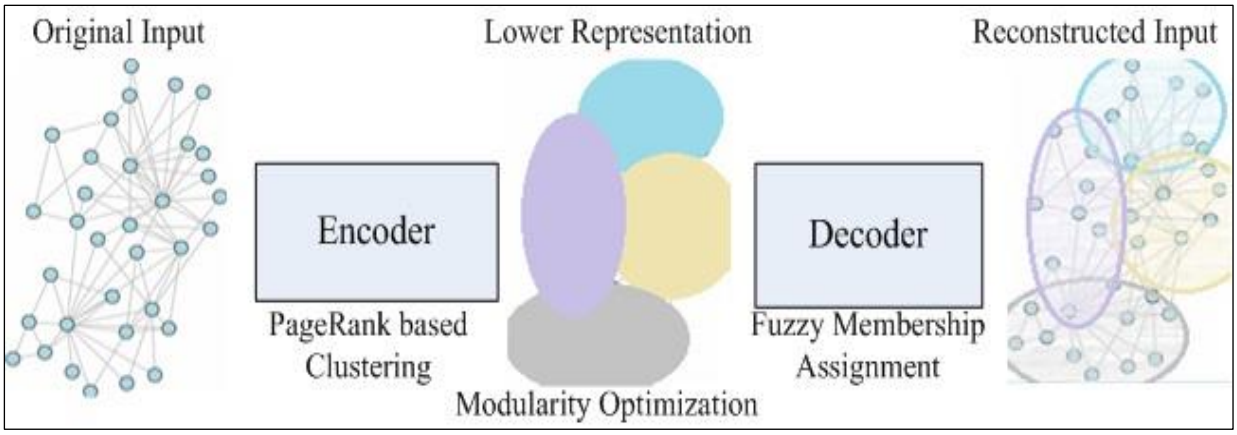


Fig. 4.7 Graph Clustering using Autoencoder

Autoencoder attempt to reconstruct those lost vertices in the final output to minimize the information loss. It is important to note that autoencoder only reconstruct original graph by linear reconstruction. Thus, DFuzzy exploits both the approaches so that the shortcomings of either approach can be compensated by the advantages of the other.

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:

$$X_l = D_{A,l}(X_{l-1} - m_l) \qquad (15)$$

where, $m_l = \frac{No.\ \text{Of Connections}}{2}$. 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 hi and output vector f_i are computed using:

$$h_i = H_o(Wx_i + a) \quad and \quad f_i = F_o(Xh_i + b) \qquad (16)$$

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

$$L(i) = \min_{\theta} = \sum_{i=1}^{N} \left\| f_i - x_i \right\|_2 + \beta(\rho) \qquad (17)$$

where, β controls the weight of the sparsity penalty term and ρ is average activation of hidden unit H_o. 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 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 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 number 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 is needed to be minimized. Hence, individual messages from each worker node are combined into a single message, for overcoming from the most costly network communication. The working of DFuzzy using autoencoders is summarized in Algorithm 4.9.

Algorithm 4.9: 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 (4.5)
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 (4.6)
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 (7)
14 **End**
15 Precision Enhancement

The functioning of the phases of DFuzzy in distributed environment Pregel is given below:

A. Unsupervised Pre-Training

The pre-training is performed before the fine-tuning phase to optimize the number of iterations used in fuzzy clustering. Here, basic autoencoders are used by setting the target values V_0 to be equivalent to the input values *V*. The autoencoder tries to learn PageRank values of the vertices using equation (4).

B. Fine-Tuning

The Pre-Training step initializes the value of vertices and then updates them using the PageRank algorithm to reach closer towards 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. 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 clusters are formed using personalized PageRank 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 $C[k]$.

Let the modularity matrix $D=[q_{ij}] \in \Re^{N*N}$ where, $q_{ij} = a_{ij} - \frac{\deg(i)*\deg(j)}{m}$. The modularity matrix D is the input of the corresponding autoencoder layer. The encoder map the data D to a lower dimensional matrix $B= [b_{ij}] \in \Re^{p*N}$ where, $p<N$. Such that i^{th} column will be equal to:

$$b_i = f(q_i) = H_o(W_B x_i + p_H) \qquad (17)$$

Where, $W_B \in \Re^{p*N}$ and $p_H \in \Re^{p*1}$ are the parameters to be learn. The parameters will be learned from the change in modularity value ΔQ_{fuzzy} of the clusters to which vertices are added. It is given as:

$$\Delta Q_{fuzzy} = \frac{1}{m}\left[\left(\sum_{j\in p} a_{vj} - degree(v)\frac{\sum_{j\in p} degree(j)}{2m}\right) - \left(\sum_{j\in q} a_{vj} - degree(v)\frac{\sum_{j\in q} degree(j)}{2m}\right)\right]$$

The activation function H_o is an element-wise non-linear mapping. Decoder converts the latent representation $B= [b_{ij}] \in \Re^{p*N}$ to the original data form $D= [q_{ij}] \in \Re^{N*N}$ with the modifications in associated cluster of certain vertices.

C. Precision Enhancement

It is the final phase of the proposed autoencoder based model. In this phase, the input graph matrix M= $a_{ij} \in \Re^{N*N}$ is reconstructed. The parameter of the encoder $\{W, a\}$ are learned by computing the distance of vertices in a cluster. To better estimate the impact of seed node on all the vertices of community, Manhattan distance is used to compute the distance because it is not a squared function and is less sensitive to noise.

The decoder reproduces the original graph with formed clusters. The value of the vertex is reconstructed by associating it with the constituting clusters.

5. EXPERIMENTAL EVALUATION

This chapter has the details of the experiments performed on real and synthetic graph data sets, to validate the proposed approaches. Standard performance metrics are used to evaluate the quality of the clusters obtained by applying these approaches. The detail of the metrics is given in section 5.2. In further sections the performance of the proposed algorithms is examined.

The first phase of experimentation is performed on testing the proposed clustering algorithms, and comparing them with the state-of-art graph clustering approaches. The next phase of experimentation involves testing the accuracy of the proposed approaches for real life graph datasets. The next step is to check the scalability of the proposed approaches by running them on very large datasets (with million or billions of vertices) using parallel and distributed computing platform. This chapter shows that the proposed approaches are suitable for tackling large graphs. The last part of the chapter presents a comparative analysis of the proposed approaches.

5.1 Dataset

To validate the proposed approaches, experiments are performed on six real life datasets from social networks, collaboration networks and product networks and four synthetic datasets. These datasets have been taken from Stanford large network dataset collection [176] and Koblenz network collection [177].

5.1.1 Graph Datasets

Ten datasets of various sizes are used for the evaluation. Six dataset are real-life graphs and four datasets are generated synthetically using Graphgen tool.

Table 5.1 Synthetic graph Datasets used

Data Code	Data Set Type	| V |	| E |
DS_1	Synthetic	68,209	253,330
DS_2	Synthetic	165,220	502,872
DS_3	Synthetic	281,341	1,128,850
DS_4	Synthetic	10,412,300	124,876,078

Table 5.2 Real life graph Datasets used

S. No.	Name	|Vertices|	|Edges|	Description
1	arXiv hep-th	18,771	198,050	Collaboration Network of authors
2	Amazon	334,863	925,872	Amazon Product Network
3	DBLP	317,080	1,049,866	DBLP Collaboration Network
4	YouTube	1,134,890	2,987,624	YouTube Online Social Network
5	LiveJournal	3,997,962	34,681,189	LiveJournal Social Network
6	Orkut	3,072,441	117,185,083	Orkut Online Social Network

The detail of synthetic datasets is given in Table 5.1. The features of six real-life dataset used are given in Table 5.2 Large graphs constitute some topological features that are non-trivial in nature such as heavy tail present in the degree distribution as shown in Fig. 5.1 [177] and high clustering coefficient. They also possess assortative [178] or disassortative nature among vertices.

Degree Distribution $P(d)$; d=0,1,,,,, is the proportion of the vertices in the network with degree k. Formally, $P(d) = n_d / n$, where n is the size of network and n_d is the number of nodes in network with degree d. The degree distribution of the real life complex networks follows a wide-tail power law of the form $P(d) \sim d^{-\gamma}$, $\gamma > 1$.

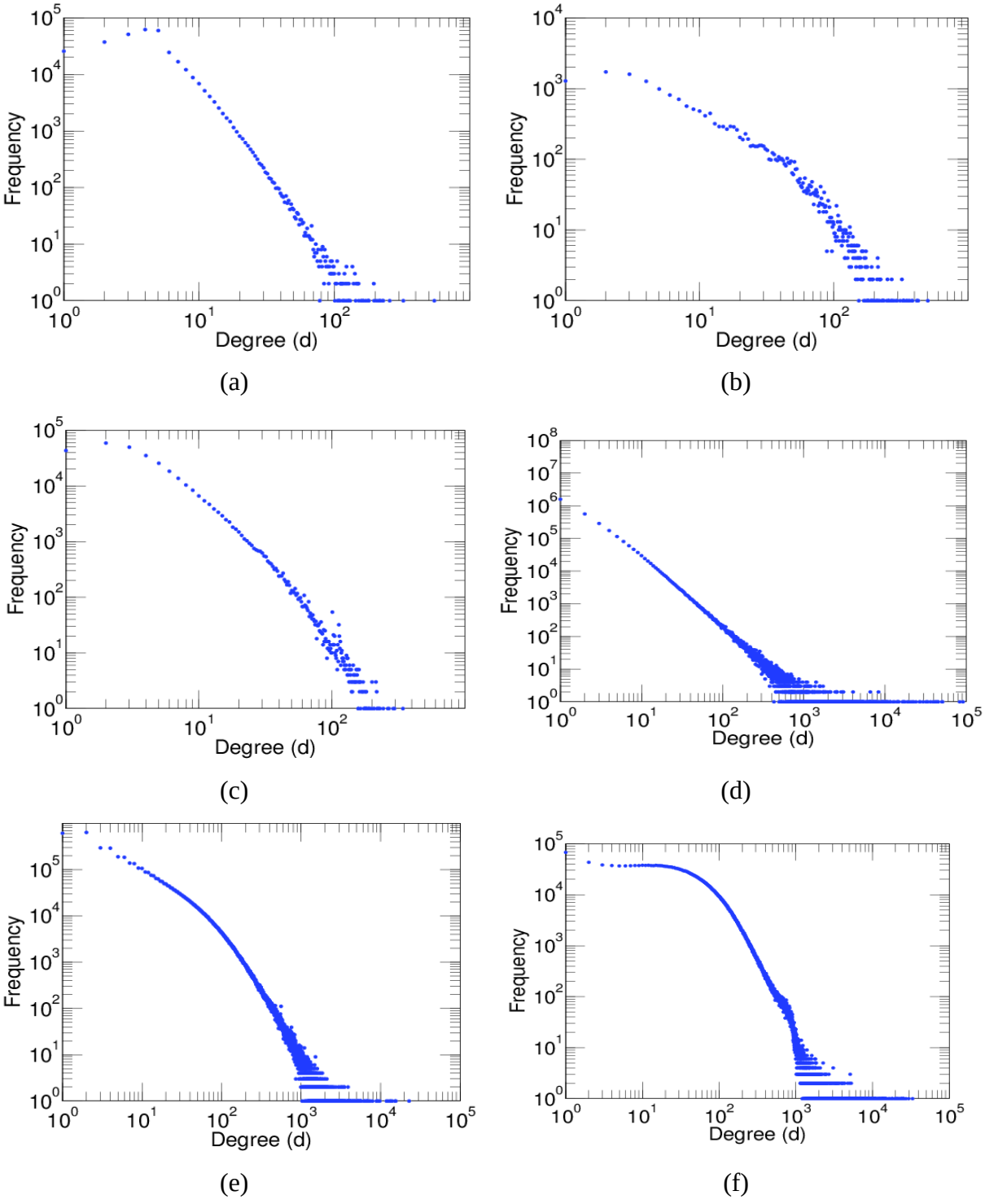


Fig. 5.1 Degree Distribution of the dataset used: (a) arXiv hep-th, (b) Amazon, (c) DBLP, (d) YouTube, (e) LiveJournal, (f) Orkut

Table 5.3 Dataset with ground truth communities

Name	Type	|Vertices|	|Edges|	Communities
Amazon	Undirected, Communities	334,863	925,872	75,149
DBLP	Undirected, Communities	317,080	1,049,866	13,477
YouTube	Undirected, Communities	1,134,890	2,987,624	8,385
LiveJournal	Undirected, Communities	3,997,962	34,681,189	287,512

5.1.2 Dataset with ground communities

Out of six real life dataset from Table 5.2, Four dataset have ground truth data available [176]. The detail of the data with the ground truth communities is given in Table 5.3. Following are the details of the four graph datasets.

- **Amazon:** This graph was collected by crawling Amazon website. This dataset is based on "*Customers Who Bought This Item Also Bought"* feature of the Amazon website. If a product *i* is frequently co-purchased with product *j*, the graph contains an undirected edge from *i* to *j*. Each product category provided by Amazon defines each ground-truth community. Each connected component in a product category is considered as a separate ground-truth community.
- **DBLP:** This dataset is co-authorship network where two authors are connected if they publish at least one paper together. Any publication venue such as journal or conference, defines an individual ground-truth community. Authors who published to a certain journal or conference form a community. Each connected component in a group is considered as a separate ground-truth community.
- **YouTube:** This is a social network dataset in which users form friendship with each other and users can create groups which other users can join. User-defined groups are considered as ground-truth communities.

- **Orkut:** It also allows users form a group which other members can then join. User-defined groups are considered as ground-truth communities.

5.2 Cluster Quality Measures

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

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

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

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,……,N-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 $\bar{C}_k$:

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

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:

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

F-Measure: Given a complex network $G(V,E)$, ground truth information C_g about the clusters and the detected clusters C_k by PFCA.

For assessing the amount of correspondence between C_g and C_k, F_1 and F_2 measure are evaluated. F_β measure is defined as follows:

$$F_\beta(C_g) = (1+\beta^2).\left(\frac{precision(C_g).recall(C_g)}{(\beta^2.precision(C_g)) + recall(C_g)}\right) \quad (9)$$

where, β is a positive real value, precision is the fraction of pairs correctly put in the same cluster and recall is the fraction of actual pairs that were identified. The precision and recall of C_g are defined as:

$$precision(C_g) = \frac{|C_k \bigcap C_g|}{|C_k|} \quad \text{and} \quad recall(C_g) = \frac{|C_k \bigcap C_g|}{|C_g|}$$

F_1 also known as balanced F-score is the harmonic mean of precision and recall. Whereas, F_2 measure calculates harmonic mean by giving more weigh to recall than precision.

The average F_β measure can be calculated as:

$$Avg(F_\beta) = \frac{1}{|C_g|}\left(\sum_{C_{gi} \in C_g} F(C_g)\right) \quad (10)$$

Modularity: It is a measure of efficiency of the clusters formed within graph [116]. A graph with large modularity has more intra-edges and less inter-edges among various clusters.

The objective function for calculating modularity for hard clusters is:

$$Q = \frac{1}{2h_c} \sum_{(i,j) \in V^2} \left(M_{ij} - \frac{\deg(i) * \deg(j)}{2h_c}\right) * C_{ij}$$

Where, M_{ij} is the adjacency matrix, $h_c = \frac{1}{2}\sum_i \deg(i)$ is the number of edges and C_{ij} is the cluster containing vertices i and j. Here,

$$C_{ij} = \begin{cases} 1, & \text{if } (i,j) \in C_i \\ 0, & \text{otherwise;} \end{cases}$$

Higher the modularity better is the clustering.

5.3 Performance Evaluation

The proposed algorithms can be executed on any commodity system cluster. The experiments to evaluate the efficiency of the algorithms are performed above a Hadoop cluster consisting of on 8 dell machines each with 8GB of RAM, 1024 GB hard disk, having Ubuntu Linux OS installed. All the machines were connected via a gigabyte network. One node is considered as master node, and rest are considered as slave nodes. The master node is also used as slave node. Hadoop 2.6.0 and Giraph 1.1.0 are used for performing various experiments.

5.3.1 PGFC

Six datasets of different sizes are taken from Table 5.1 and Table 5.2. Two real world datasets taken from Stanford Large Network Dataset Collection [176] 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.

The proposed PGFC algorithm is compared with the state-of-art algorithms namely K-Means, FCM 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*[180].

5.3.1.1 *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 clustering algorithms. The comparative values of the considered algorithms in terms of partition coefficient and conductance for various datasets are given in Table 5.4. For both the quality measures, lower is the value better are the clustering results.

Table 5.4: Cluster Quality Comparison

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
LiveJournal	0.90	0.411	1.0	0.572	0.78	0.292	0.71	0.269
Orkut	0.89	0.352	1.0	0.492	0.76	0.247	0.68	0.212
DS_4	0.88	0.374	1.0	0.433	0.79	0.283	0.71	0.248

For K-Means, the value of partition coefficient has been always 1.0 because, it forms hard clusters. For Orkut graph 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_4, 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, FCM and K-Means for all the datasets.

5.3.1.2 Number of iterations

The number of iterations before convergence of PGFC along with state-of-art algorithms is shown in Fig. 5.2. 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 proposed PGFC is designed to work with large graphs thus, finishing its execution in higher number of iterations in comparison to other algorithms for small datasets.

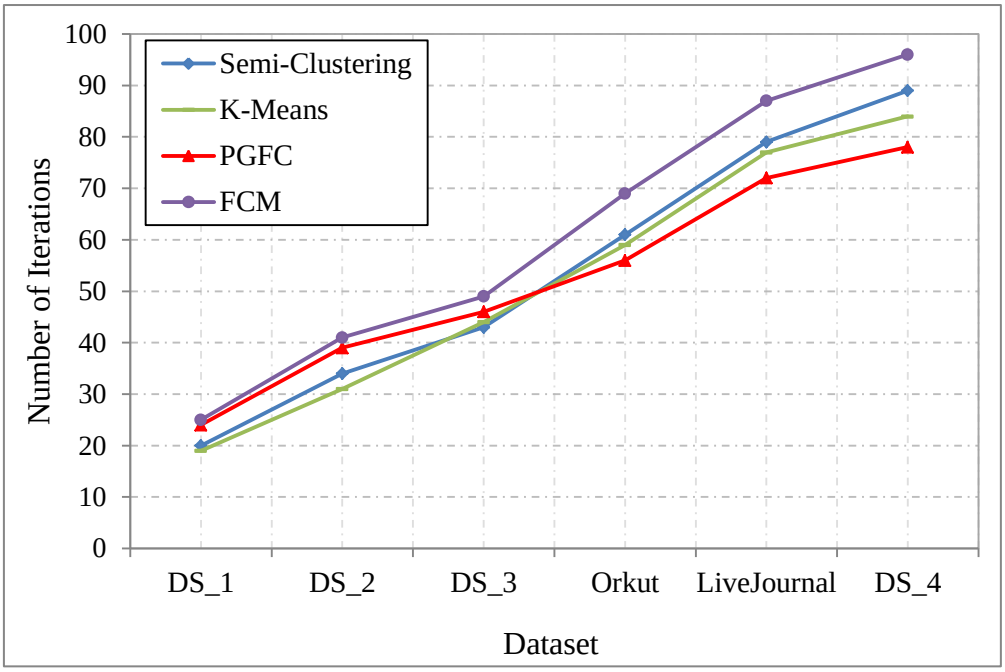


Fig. 5.2 PGFC: Comparison in terms of number of Iterations

However, for larger datasets, the gradual improvement in the performance of PGFC can be noticed. The results show that despite of having large 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 orkut dataset, 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.

5.3.1.3 Scalability of PGFC

Run Time of an algorithm is the time taken by a system to execute that algorithm. The total run time to execute the algorithm includes the time of loading the graph to Giraph and the time to run that algorithm.

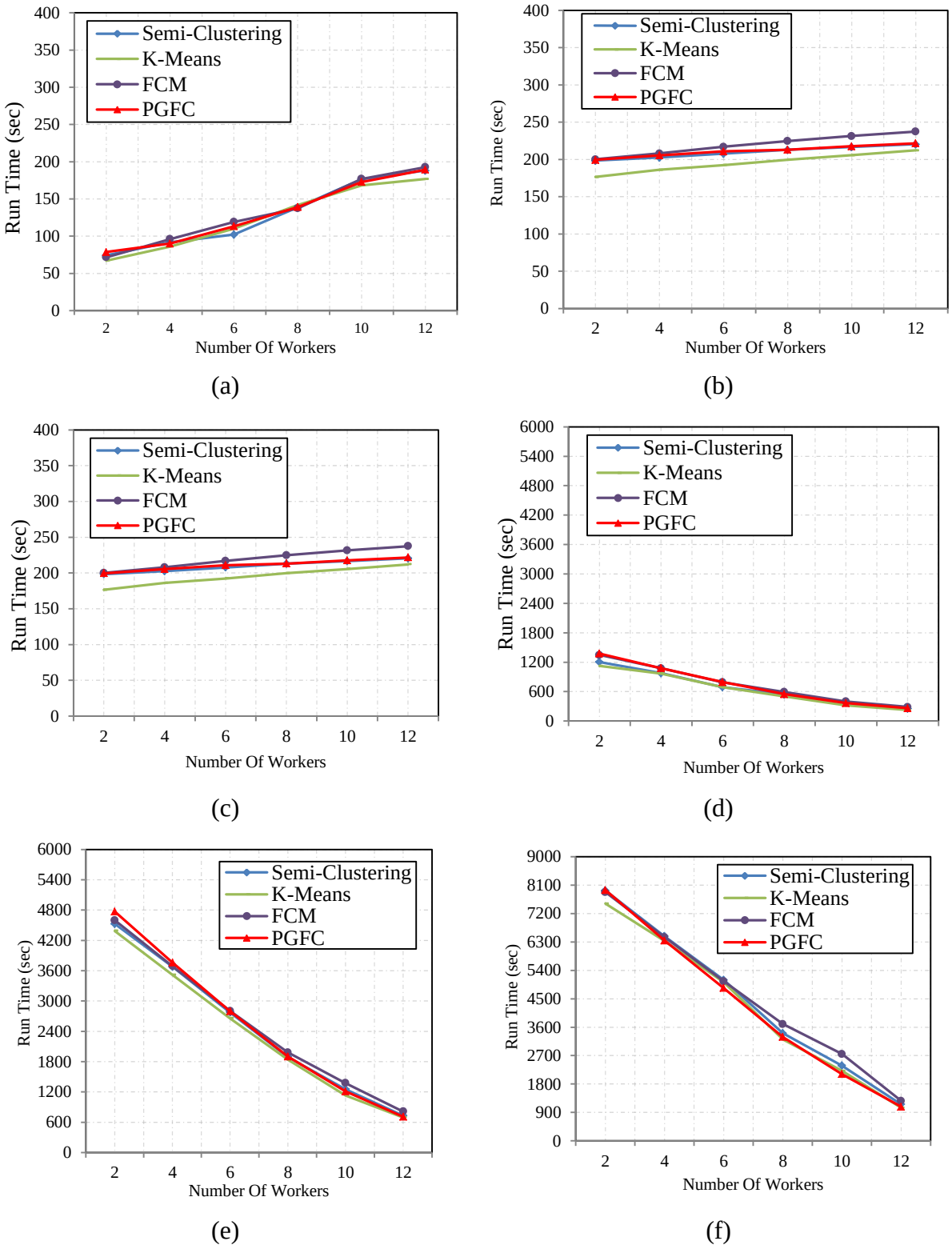


Fig. 5.3. Scalability: Run Time of clustering algorithms with different Number of Workers (a) DS_1, (b DS_2, (c) DS_3, (d) LiveJournal, (e) Orkut, (f) DS_4.

In Fig. 5.3, run time of semi-clustering, K-Means, FCM and proposed PGFC algorithm is shown. PGFC, FCM and Semi-clustering are finding the overlapping clusters, whereas K-Means is finding the hard clusters.

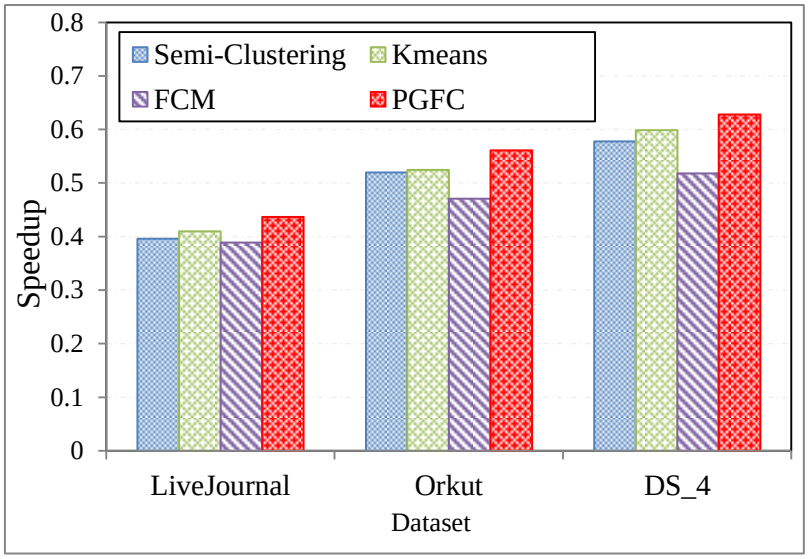


Fig. 5.4. Speedup behavior of clustering algorithms (higher bar indicates better scalability)

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 first find the degrees of all the vertices and then performing full inverse-distance weighting.

From the Fig. 5.3, linear run time is observed for all the algorithms when the number of worker nodes is increased. For large graph datasets Orkut and D_4, 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 Orkut and DS_4 graph datasets respectively. Also, PGFC is 12.8% and 14% faster than its inherent algorithm FCM for Orkut and DS_4 respectively.

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:

$$Speedup = \frac{Run\_Time_1}{P * Run\_Time_P} \qquad (12)$$

Where, P is the number of processors used. For ideal parallization, *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. 5.4 for three large graph datasets LiveJournal, orkut and DS_4. 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_4.

5.3.2 PFCA

In this section, the performance analysis of the proposed PFCA algorithm is done on the basis of run-time, scalability, F-score and modularity. The performance of PFCA is compared with the state-of-art algorithms such as K-Means and FCM, Label propagation and with Pregel based semi-clustering on the datasets from Table 5.2. In all of the aforementioned state-of-art algorithms, the number of clusters is pre-defined. The same number of clusters is considered that were computed by PFCA for comparison analysis. The run time, scalability and processor usage statistics on six real life datasets from various big data domains has been depicted. The accuracy of PFCA in predicting the number of clusters is verified using the ground truth data from real life datasets from domains like social network, collaboration network and product network.

A series of experiments were performed to quantify the efficiency of PFCA in comparison to the K-means, FCM, Pregel Semi-clustering and Label Propagation. For evaluation, the fuzzyfication index m is considered as 2 and the epsilon ε as $le - 3$ [180]. The value of dumping factor α is considered as 0.85.

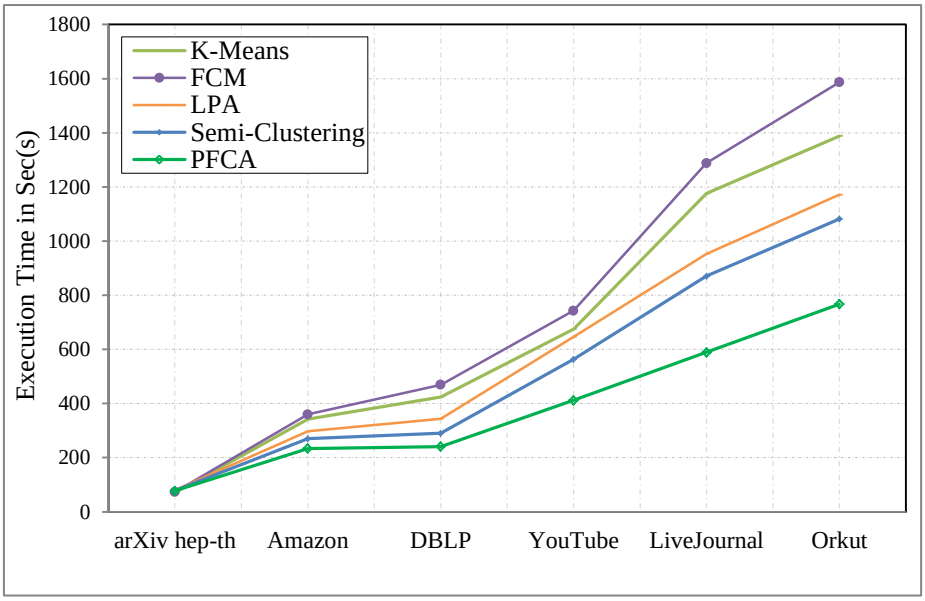


Fig. 5.5 PFCA: Run Time Comparison

5.3.2.1 Run Time

Run Time of an algorithm is the time taken by a system to execute that algorithm. The comparison of computational time of the proposed parallel fuzzy clustering algorithm 'PFCA' with the state-of-the-art algorithms over the mentioned datasets is shown in Fig. 5.5. The communication overhead is reduced in PFCA by using combiners of Giraph. Combiners aggregate the messages prior to transfer in each superstep. Combiners in PFCA aggregate all the PageRank values received from worker in the superstep and forward it as a single grouped message.

The lesser communication overhead here results in faster computation as shown in Fig. 5.5. Despite of discovering the number of clusters according to the structure of network, PFCA is significant in terms of execution time in comparison to other competent algorithms for all the datasets. The proposed algorithm becomes more efficient in terms of time with the increase in the size of network. FCM algorithm gets struck in the local minima thus, takes more time in comparison to other algorithms for large complex networks.

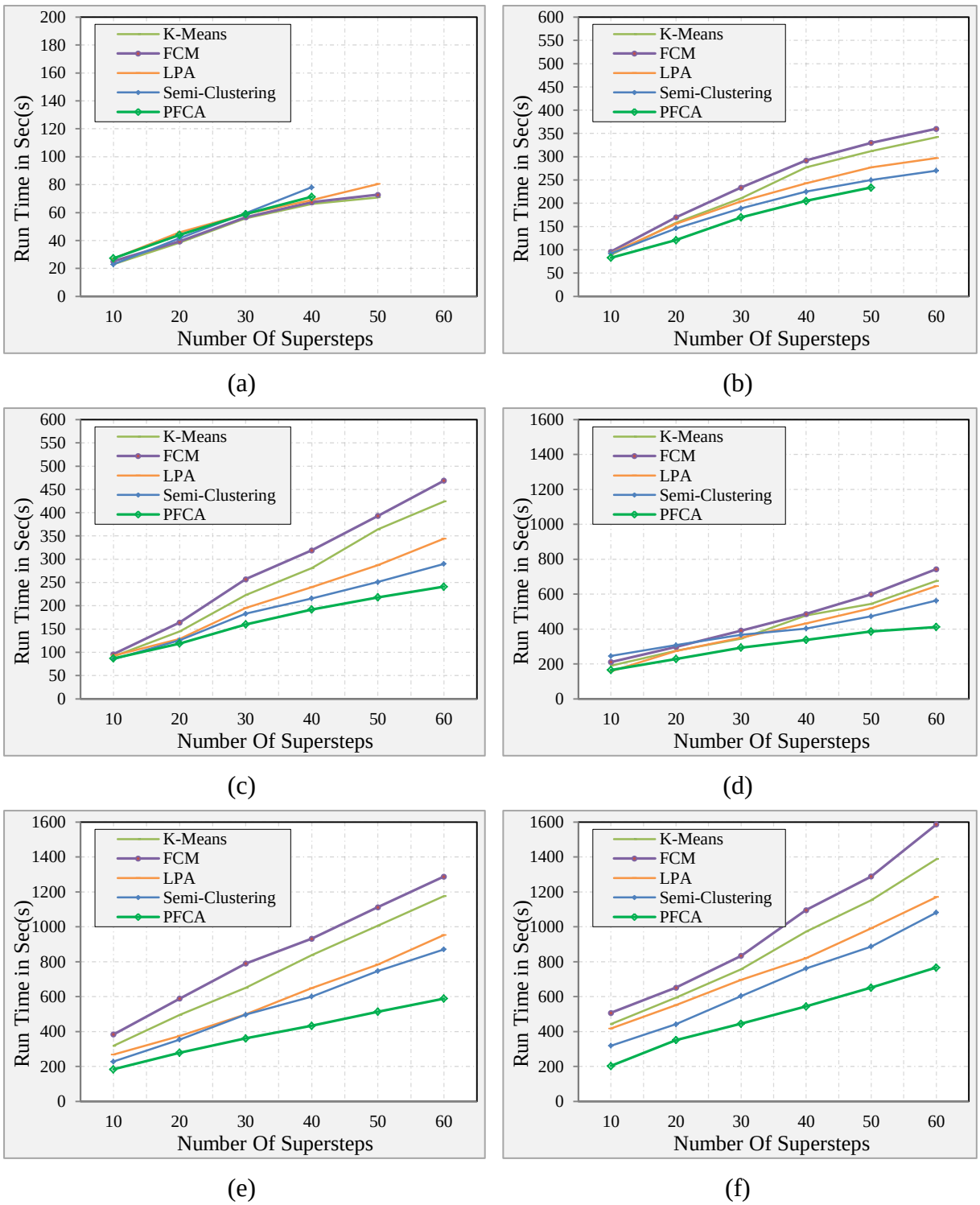


Fig. 5.6 PFCA: Run Time vs Number of supersteps (a) arXiv hep-th, (b) Amazon, (c) DBLP, (d) YouTube, (e) LiveJournal, (f) Orkut

For all the datasets, proposed PFCA is twice as magnitude faster than FCM. For the largest dataset Orkut, the proposed algorithm is 0.44%, 0.51%, 0.34% and 28.9% computationally efficient in terms of Run Time from K-Means, FCM, LPA and Semi-Clustering respectively.

Fig. 5.6 shows the run-time of PFCA and the other state-of-art clustering algorithm in various supersteps. It can be observed that the difference in run time is more noticeable for large networks in later supersteps. For arXiv and Amazon networks, PFCA finishes its execution in lesser supersteps than all the other algorithms.

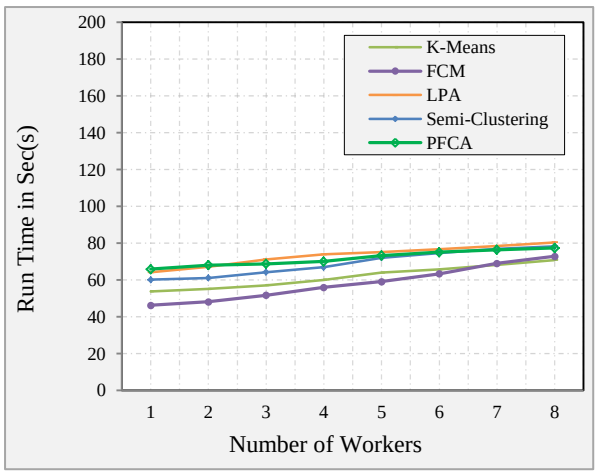
Fig. 5.6(f) illustrates the result over the Orkut network. PFCA achieves up to two orders of magnitude better performance than the baseline FCM algorithm.

5.3.2.2 Speedup and scalability

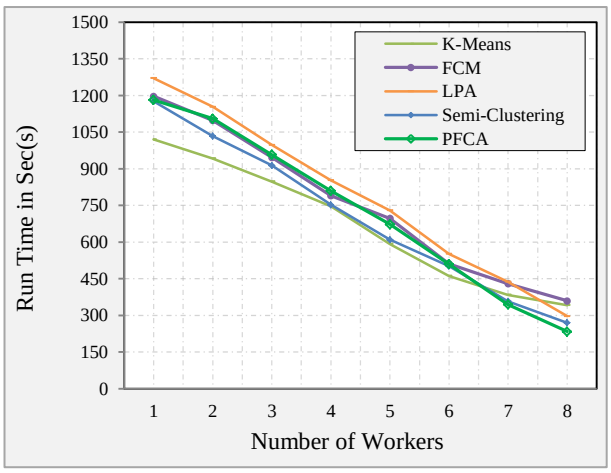
The second experiment is to analyze the scalability of the proposed PFCA by increasing the number of worker nodes and the size of the network. Fig. 5.7 shows the execution time of PFCA for increasing number of worker nodes over the considered graph datasets. As anticipated, increase in the number of processors results in decreased run time. As PFCA performs CPU bounded computations, a linear run time is observed when the numbers of worker nodes are increased.

Despite of that, PFCA manages to have significant run-time in comparison to the other state-of-art clustering algorithms. The significance of PFCA is more noticeable for large networks such as YouTube, LiveJournal and Orkut. For All the networks PFCA has relatively more processing time then K-Means and Semi-Clustering for lesser number of Worker nodes. However, after using around 4 to 5 Worker nodes, the proposed PGFC performs better in terms of processing time then the competent algorithms.

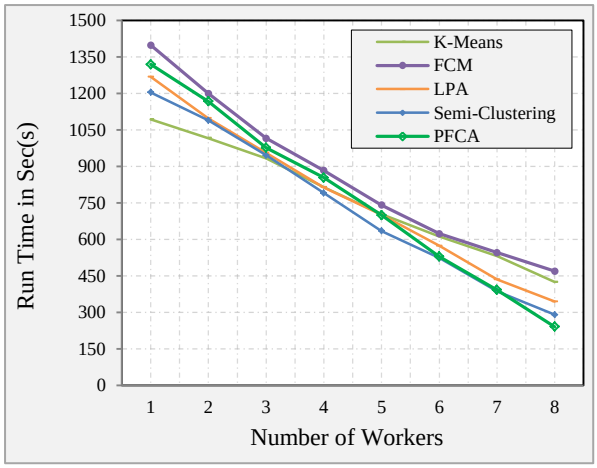
For largest dataset used Orkut, PFCA is 2 times more efficient in terms of running time than FCM as shown in Fig. 5.7(f). PFCA also outperforms Label propagation algorithm and semi-clustering by 12.09% and 14.8% for Orkut network.



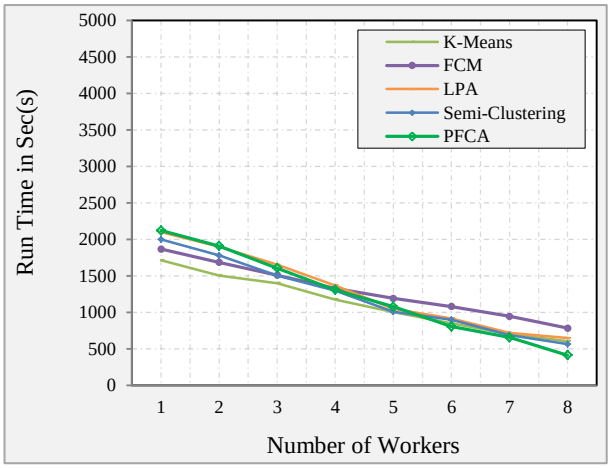
(a)



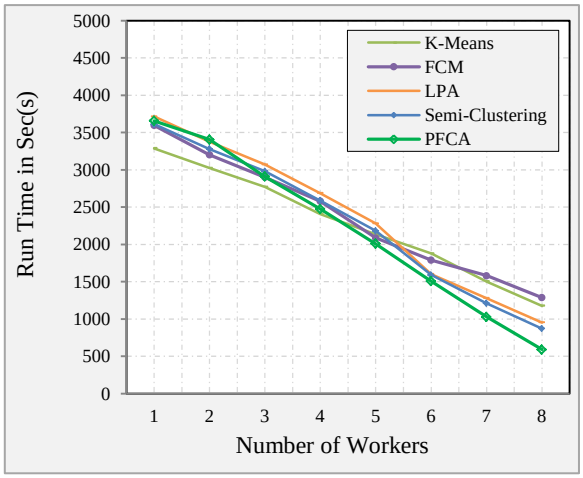
(b)



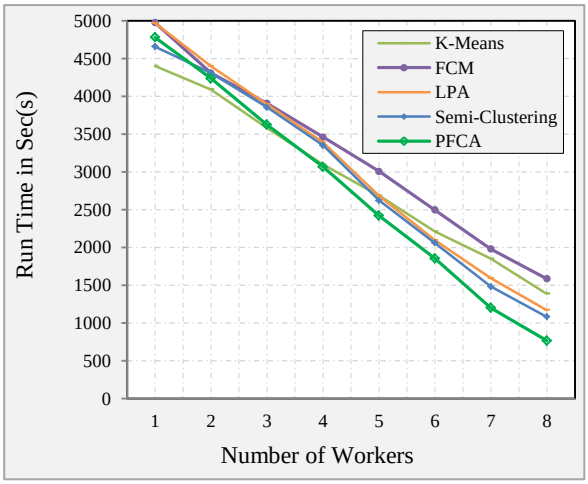
(c)



(d)



(e)



(f)

Fig. 5.7 Scalability: Number of workers vs. Run Time (a) arXiv hep-th, (b) Amazon, (c) DBLP, (d) YouTube, (e) LiveJournal, (f) Orkut

Table 5.5: Speedup Factor of PFCA for various dataset

Datasets	Algorithms	Total Run Time using		Speedup Factor
		One Processor	Eight Processors	
arXiv hep-th	K-Means	53.88	70.81	0.095113685
	FCM	46.23	72.8	0.079378434
	Semi-Clustering	60.12	78.1	0.096222791
	LPA	64.15	84	0.09546131
	PFCA	67.92	77.3	0.109831824
Amazon	K-Means	1010	342	0.369152047
	FCM	1060	360	0.368055556
	Semi-Clustering	1176	297	0.494949495
	LPA	1271	297	0.53493266
	PFCA	1182	234	0.631410256
DBLP	K-Means	1223	334	0.457709581
	FCM	1298	329	0.493161094
	Semi-Clustering	1204	283	0.53180212
	LPA	1268	344	0.460755814
	PFCA	1329	241	0.689315353
YouTube	K-Means	1913	625	0.3826
	FCM	1865	783	0.297733078
	Semi-Clustering	1998	513	0.486842105
	LPA	2098	646	0.405959752
	PFCA	2120	412	0.643203883
LiveJournal	K-Means	3387	1176	0.360012755
	FCM	3597	1288	0.349087733
	Semi-Clustering	3612	891	0.506734007
	LPA	3715	953	0.48727702
	PFCA	3656	589	0.775891341
Orkut	K-Means	4403	1378	0.399401306
	FCM	4978	1587	0.392091997
	Semi-Clustering	4659	982	0.593049898
	LPA	4971	1082	0.574283734
	PFCA	4783	767	0.779498044

The speedup values of PFCA along with the state-of-art clustering algorithms are given in Table 5.5. As shown, PFCA has high Speedup in comparison to all the competent clustering algorithms. Although, the other algorithms are also scalable, but they are not so efficient in terms of Run-time. PFCA achieves 23%, 19%, 15% and 22% higher speedup from K-Means, FCM, Semi-Clustering and LPA respectively for DBLP dataset. The difference becomes higher for large datasets. For Orkut network, PFCA has 38%, 38%, 18% and 20% higher speedup than K-Means, FCM, Semi-Clustering and LPA respectively. Proposed PFCA achieves the speedup of 0.77 for LiveJournal and Orkut online social network.

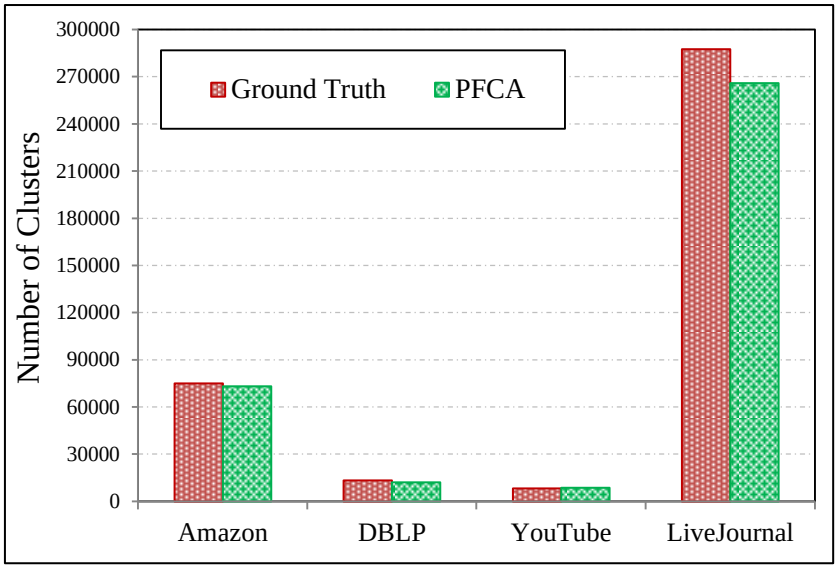


Fig. 5.8 Accuracy of PFCA in Prediction of Number of clusters corresponding to ground truth data

5.3.3.3 Number of Clusters

PFCA analyzes the structure of the graph to find the number of clusters. The ground truth data [38] is used to validate the number of cluster prediction by PFCA. Fig. 5.8 shows the composite performance of PFCA on the graphs for which ground truth data is available. Given a graph $G(V,E)$, ground truth information C_g about the clusters and the detected clusters C_k by PFCA. For assessing the amount of correspondence of C_g to $C_{k,}$ accuracy is calculated using:

$$Accuracy\ in\ \% = \left(1 - \frac{|(|C_g| - |C_k|)|}{|C_g|}\right) \times 100$$

For the datasets Amazon, DBLP, YouTube and LiveJournal, the ground truth data is available. PFCA achieves accuracy of up to 97% in terms of predicting the number of clusters. For LiveJournal network, PFCA achieves accuracy of 92%.

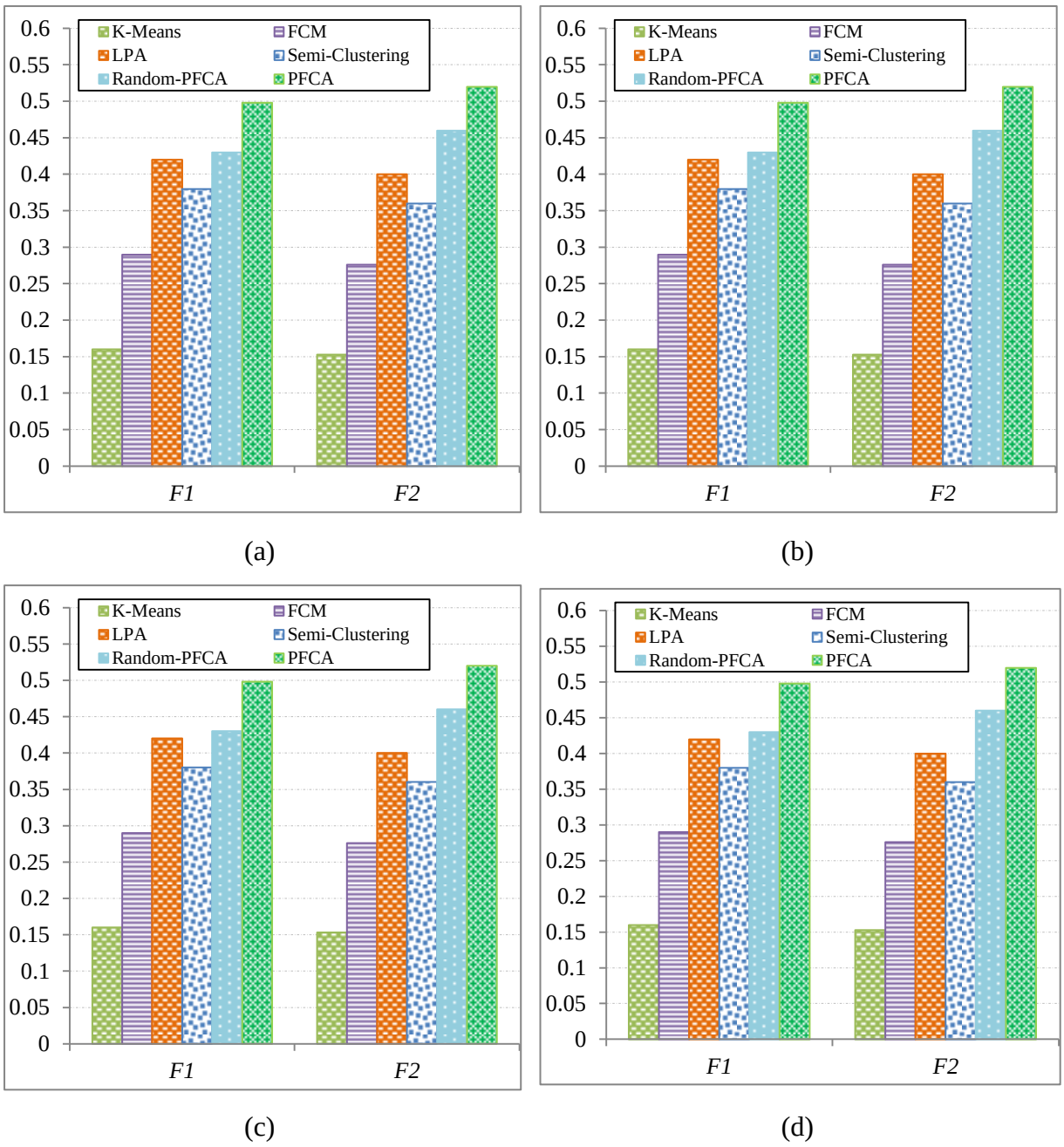


Fig. 5.9 Accuracy in terms of F_1 and F_2 Measures

5.3.3.4 F-Measure

Fig. 5.9 shows the composite performance of PFCA in terms of average F_1 and F_2 scores with the state-of-art clustering algorithms. It can be observed that PFCA outperforms other competent algorithms in terms of both F_1 and F_2 measures on all the considered networks with ground truth data available.

Table 5.6. Modularity and Conductance values of different clustering algorithms

Network	K-Means		FCM		LPA		Semi-clustering		PFCA	
	Q	C_k	Q	C_k	Q	C_k	Q	C_k	Q	C_k
arXiv hep-th	1.0	0.397	0.49	0.304	0.59	0.378	0.57	0.357	0.63	0.265
Amazon	1.0	0.413	0.54	0.293	0.76	0.345	0.71	0.355	0.77	0.291
DBLP	1.0	0.456	0.51	0.327	0.61	0.398	0.57	0.402	0.64	0.271
YouTube	1.0	0.439	0.45	0.281	0.53	0.411	0.52	0.391	0.58	0.265
LiveJournal	1.0	0.492	0.53	0.247	0.726	0.378	0.684	0.352	0.728	0.203
Orkut	1.0	0.487	0.51	0.301	0.663	0.302	0.632	0.382	0.681	0.225

It is also noticed that even in the case of Random-PFCA, when the cluster centers are selected randomly, the proposed algorithm performs best in terms of both F_1 and F_2 measures for all the considered datasets.

5.3.3.5 Modularity and Conductance

The quality of the clusters formed using the proposed PFCA is measured on the basis of modularity Q_{fuzzy}.

The modularity comparison of proposed PFCA with the existing algorithms is shown in Table 5.6. The high modularity value of PFCA for all the considered networks indicate that it discovers the overlapping communities efficiently. K-Means is a hard clustering algorithm and thus, has modularity value 1 for all the datasets. The fuzzy clustering algorithms such as FCM, overlapping LPA, semi-clustering have low modularity in comparison to the proposed PFCA. The difference between modularity values of LPA and proposed PFCA is very less in the case of Amazon and LiveJournal dataset. But for all the datasets, PFCA gains noticeable advantage over the other competent algorithms.

Conductance of a cluster is the ratio of the number of inter-cluster edges and the number of edges having an endpoint in the cluster or that do not have endpoint with in the cluster, whichever is lesser. The lower value of conductance indicates better clusters quality. As

shown in Table 5.6, PFCA has lower value of conductance for all the datasets. K-means has high conductance values among all while, FCM, LPA and semi-clustering have high conductance in comparison to proposed PFCA.

The better values of both Modularity and Conductance assures that the quality of clusters formed by PFCA is much higher than the other state-of-art clustering algorithms.

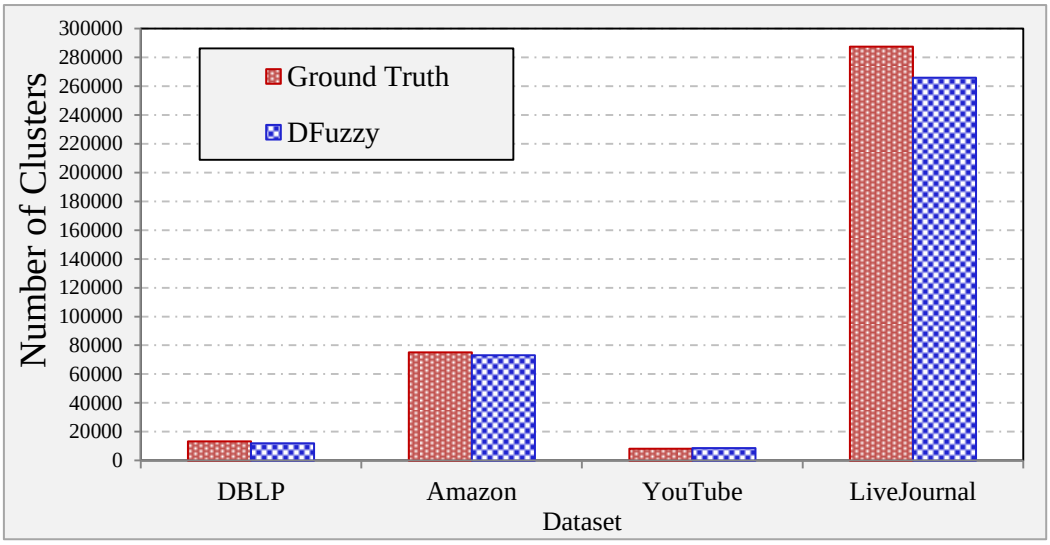
5.3.3 DFuzzy

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. The number of layers for fine tuning is considered as 5 and the maximum number of iterations to be performed on each layer as 30. The clusters with low modularity value tends to have more inter-cluster edges than intra-cluster edges thus, modularity threshold θ is taken as 0.2 [179]. 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 [180].

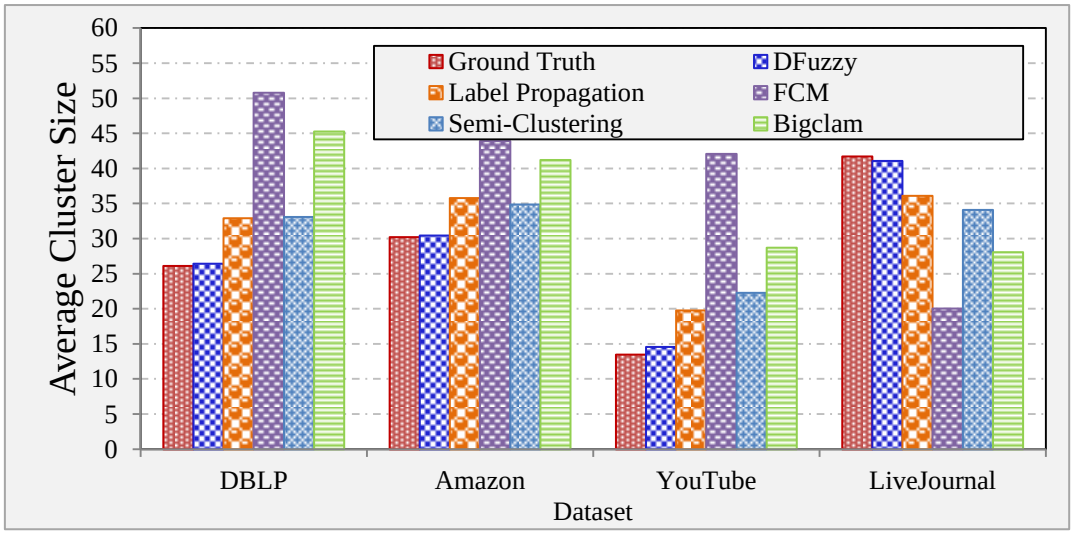
The performance of the proposed DFuzzy is compared with four state-of-art clustering algorithms namely Fuzzy C-means (FCM), Label Propagation, Semi-Clustering and BigClam. The native implementation of both Label propagation and FCM has been modified for better performance in distributed environment. Also, label propagation based overlapping clustering approach is used 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 [180].

5.3.3.1 Accuracy corresponding to ground truth data

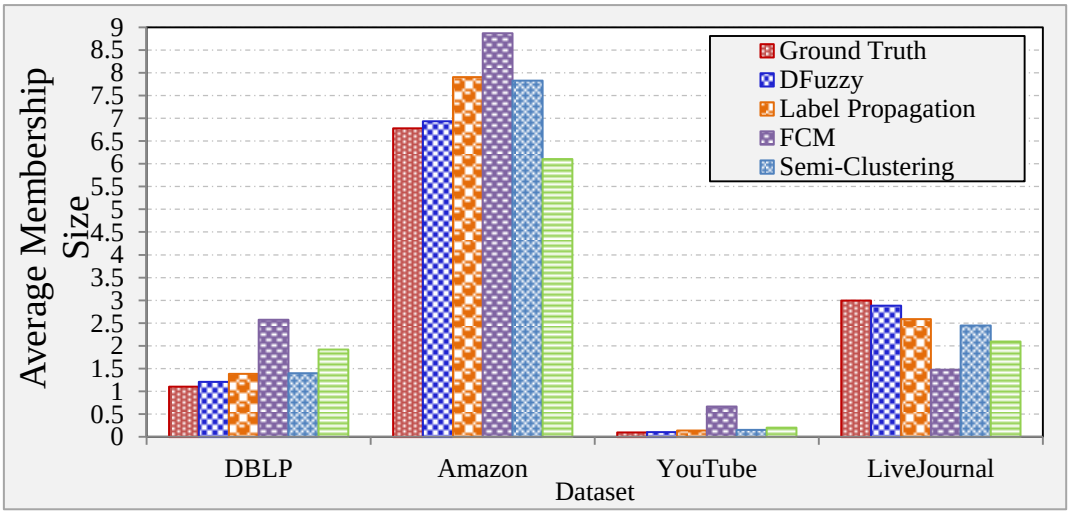
The accuracy of DFuzzy for predicting the number of clusters, average cluster size and average membership of nodes towards various clusters is measured.



(a) Number of clusters



(b) Average cluster size



(c) Average Membership Size

Fig. 5. 10 Accuracy of DFuzzy corresponding to ground truth data

The datasets mentioned in Table 5.3 with ground truth communities taken from are considered for accuracy measurement. The relevance between the clusters detected by DFuzzy against the ground truth data is analyzed.

As per the results shown in Fig. 5.10 (a), 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.

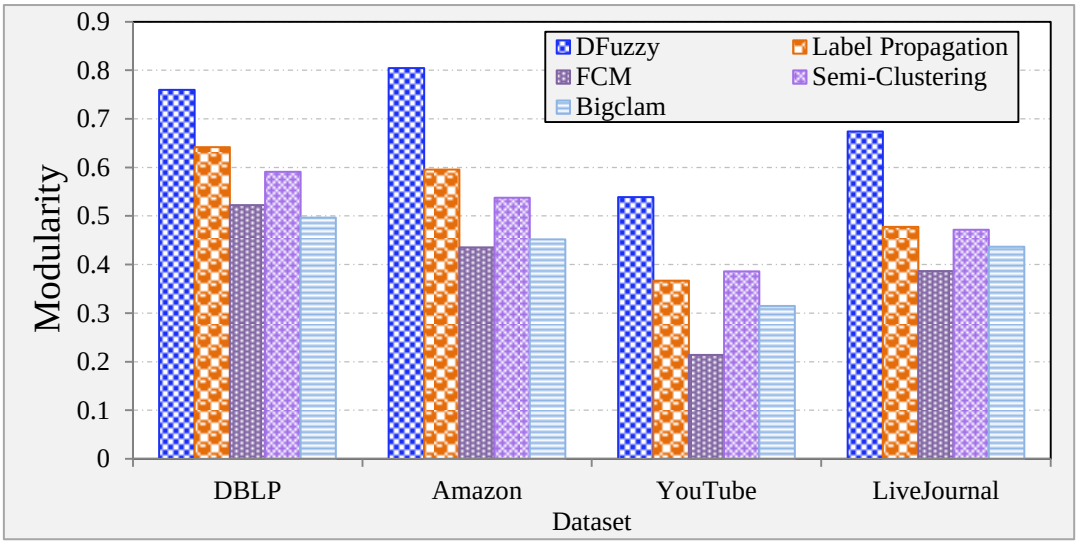
5.3.3.2 Cluster Quality

The efficiency of the DFuzzy is measured on the basis of modularity, conductance and partition coefficient. 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.

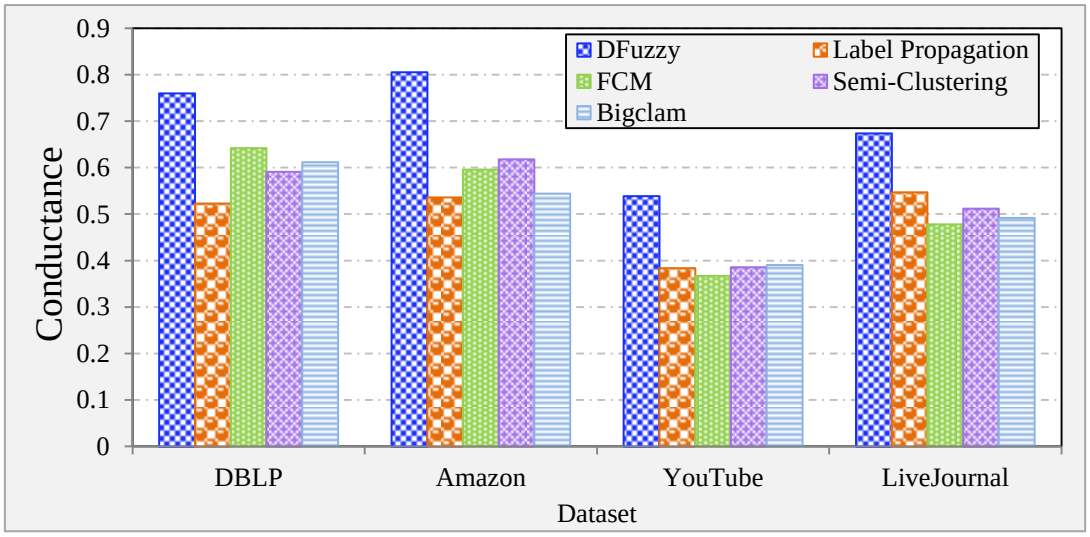
DFuzzy optimize the clusters using modularity in the precision enhancement phase. Higher is the value of modularity and conductance, better is the quality of cluster. For partition coefficient, lower is the value better are the clustering results.

Fig. 5.11(a) 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.

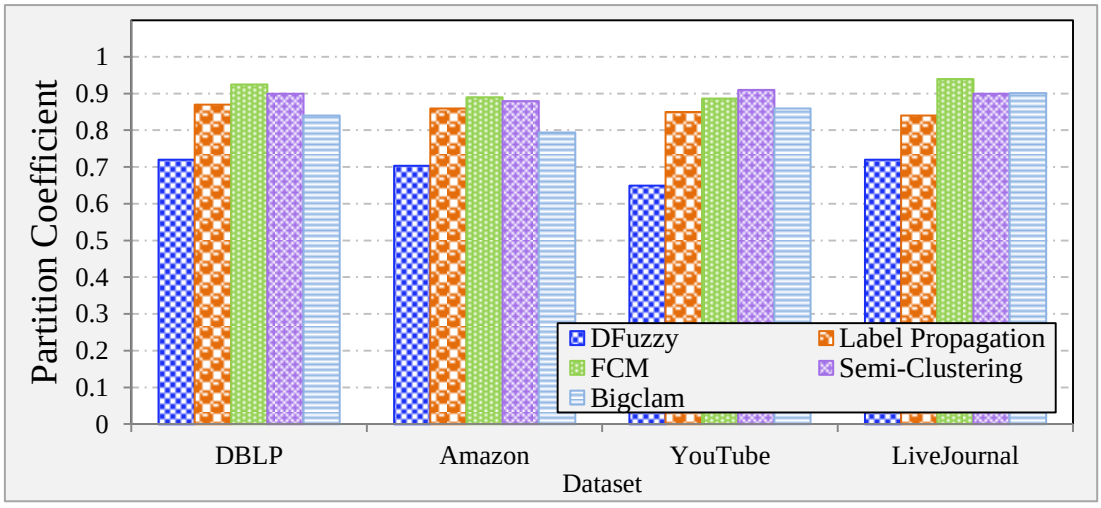
The high modularity value of DFuzzy for all the considered graphs indicate that it discovers the overlapping communities efficiently.



(a) Modularity



(b) Conductance



(c) Partition Coefficient

Fig. 5.11. Quality Assessment of DFuzzy

Fig. 5.11 (b) 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 to other algorithms.

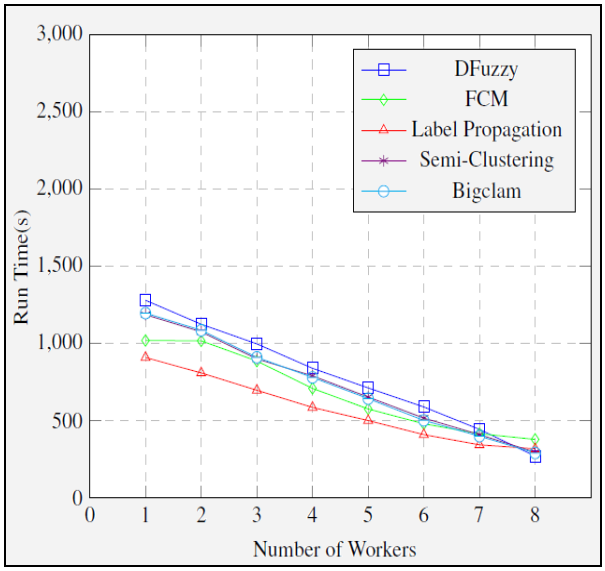
The value of partition coefficient over the considered four graph datasets is shown in Fig. 5.11 (d). The absolute average value for partition coefficient of DFuzzy is 0.696, while for overlapping label propagation is 0.855, FCM is 0.91, Semi-clustering 0.89 and Bigclam is 0.848.

Thus, DFuzzy is 22%, 30%, 28% and 21% more efficient than overlapping label propagation, FCM, Semi-clustering and Bigclam respectively.

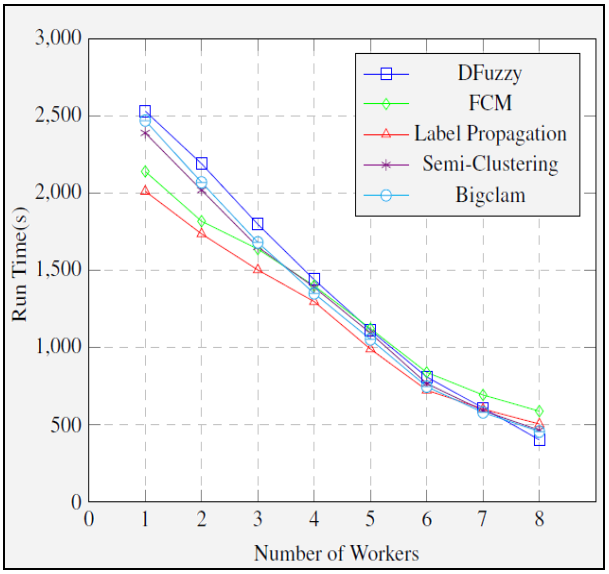
DFuzzy based on autoencoder is proved to gain a significant advantage over the other competent algorithms in terms of quality. Hence, deep structures provide better graph representations and improved clustering results.

5.3.3.3 Scalability

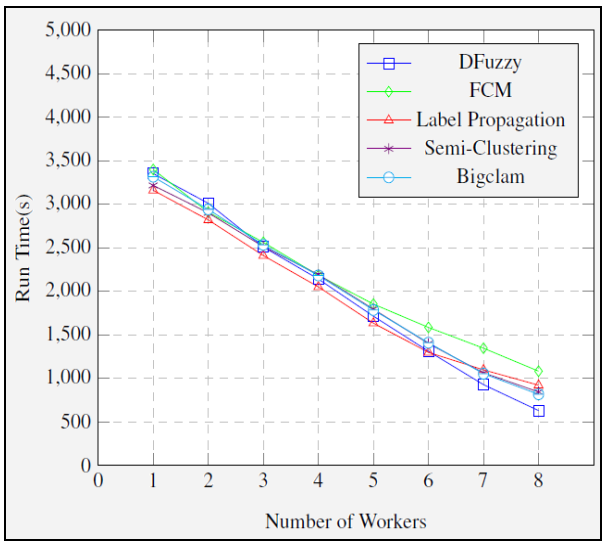
To analyze the Scalability of DFuzzy, the number of available workers *P* in Giraph job is varied. In Fig. 5.12, the scalability of DFuzzy, Label Propagation, FCM, semi-clustering and Bigclam algorithm is shown using up to eight nodes. As anticipated, increase in the number of processors results in decreased run time. 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 thus, can be scaled. But, FCM incurs high communication cost in distributed environment and hence, is not computationally efficient.



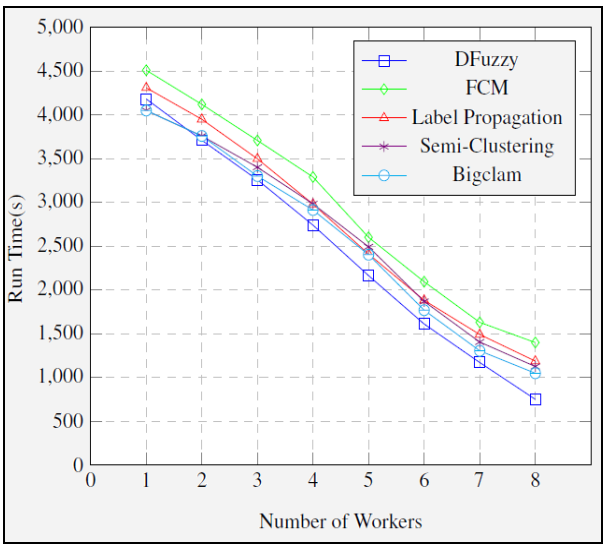
(a) Facebook



(b) Twitter



(c) GooglePlus



(d) LiveJournal

Fig. 5.12. Scalability: Number of workers vs. Run-Time

Bigclam is also scalable in its native implementation. With added worker nodes, DFuzzy performs well in terms of run-time and achieve a linear speedup as shown in Fig. 5.12.

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.

5.3.3.4 Effect of Deep Layers on Quality

For showing the effect of stacked autoencoders on processing time and accuracy, the number of layers of the autoencoder pipeline in DFuzzy are varied.

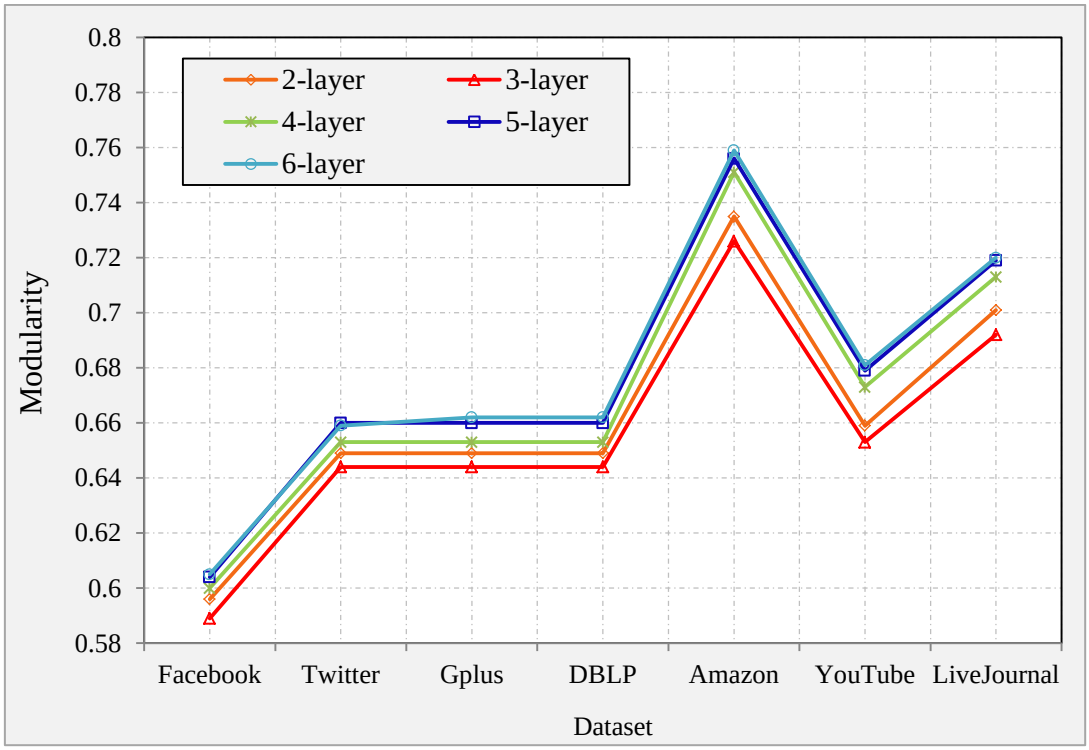


Fig. 5.13. Effect of multiple layers: Improvement in quality in terms of Modularity

The modularity of DFuzzy is shown for the shallowest 2-layer model to the deepest 6-layer model in Fig 5.13. The effect of 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, 5-layer DFuzzy model is used 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.

Table 5.7 Number of clusters and graph coverage

Graph Dataset		FCM	LPA	BigClam	S-Cl	PGFC	PFCA	DFuzzy
arXiv hep-th	Coverage (%)	100	71.2	63.1	97.1	100	88.1	100
	No. of Clusters	100	100	100	100	100	109	109
Amazon	Coverage (%)	100	90.1	99.2	91.3	100	99.9	100
	No. of Clusters	75,149	75,149	75,149	75,149	75,149	73254	73254
DBLP	Coverage (%)	100	90.3	94.6	84.7	100	99.9	100
	No. of Clusters	13,477	13,477	13,477	13,477	13,477	12186	12186
YouTube	Coverage (%)	100	91.1	79.5	90.2	100	91.4	99.8
	No. of Clusters	8,385	8,385	8,385	8,385	8,385	8683	8683
LiveJournal	Coverage (%)	100	82.2	43.9	91.2	100	88.9	99.7
	No. of Clusters	287,512	287,512	287,512	287,512	287,512	287512	287512
Orkut	Coverage (%)	100	80.5	-	90.1	100	90.2	99.5
	No. of Clusters	5,000,000	5,000,000	-	5,000,000	5,000,000	6,128,183	6,128,183

The clustering results became more accurate by using autoencoders on reaching to deep layers from shallow layers.

5.4 Performance Comparison of Proposed Approaches

The performance of proposed PGFC, PFCA and DFuzzy are compared with the state-of-art clustering algorithms in terms of number of clusters and graph coverage. Table 5.7 shows the returned coverage in percentage and number of clusters of the algorithms. Graph coverage indicates the amount of vertices allocated to clusters. For example, for 90% graph coverage, 10% vertices are not assigned to any of the cluster. In certain applications, the loss of vertices from the output may lead to loss of crucial information. FCM and PGFC form clusters by computing mean of the distance among the vertices. Hence, the graph coverage is 100% in the case of both the algorithms. PFCA covers all those vertices which fall in the

personalized PageRank walk. So, the coverage is not 100% in PFCA. On the other hand, DFuzzy recovers the lost vertices by minimizing the reconstruction error and try to include all those vertices in final output also which are not covered by personalized PageRank.

Proposed PFCA and DFuzzy determine the number of clusters by analyzing the structure of the clusters. While for all the other algorithms including proposed PGFC, the number of cluster is predefined. The ground truth cluster value is considered as number of clusters for algorithms such as proposed PGFC, FCM, BigClam, Semi-clustering and LPA.

6. FREQUENT SUBGRAPH MINING OF LARGE GRAPH

In this chapter, the proposed subgraph mining algorithms are discussed. In first section, preliminaries are given. In section 6.2, the working of proposed pattern growth based PaGro is given by leveraging the operative communication primitives for better scalability. A two-step hybrid approach is developed in PaGro for optimization of subgraph pruning task at both local and global level to avoid the excess communication overhead. The NP-Complete graph isomorphism problem has also been optimized for fast graph processing. Also for performing approximate subgraph mining, the working of PaGro is amended in Ap-FSM by exploiting sampling for faster processing. In section 6.3, the working of Ap-FSM is given. At last, the performance evaluation of both the approaches is given in section 6.4.

6.1 Preliminaries

A Graph G can be represented as a set $\{V, E\}$, where V is a set of N vertices and $E \in [V * V]$ is a set of edges. A graph's vertices and edges can be labeled as well as unlabeled.

Definition 5: (**Labeled Graph**) A labeled Graph G_L with node label L_V and an edge label L_E is a tuple of four elements $G = (V, E, \mu_V , \mu_E)$, where μ_V and μ_E are the vertex and edge labeling functions respectively such that $\mu: V \rightarrow L_V$ and $\varepsilon: E \rightarrow L_E$.

In certain applications, a large graph may contain a set of smaller graphs. For an instance, a graph $G= (g_1, g_2, \ldots. g_n)$, where the graph G is an aggregation of smaller subgraphs $(g_1, g_2, \ldots. g_n)$. Formally a subgraph can be defined as:

Definition 6: (**Subgraph**) A labeled graph $g_l = (v, e, \mu_g, \varepsilon_g)$ is a subgraph of a labeled graph G_L, if the vertices $v \subseteq V$, edges $e \subseteq (E \bigcap (v \times v))$, $\mu_g(i) = \mu(i)$ for all $i \in v$, and $\varepsilon_g(i,j) = \varepsilon(i,j)$ for

all $(i,j) \in e$.

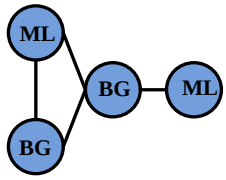
On the contrary, graph G is called a supergraph of g, if g is subgraph of G. A subgraph g is frequent if the *support* (g) is no less than the frequency threshold. For matching two similar subgraphs, subgraph isomorphism is computed which map every node of subgraph g_1 to g_2 in such a way that node and edge labels are consistent while preserving edge structure.

Definition 7: (**Subgraph Isomorphism**) Two Subgraphs $g_1=(V_1, E_1, \mu_1, \varepsilon_1)$ and $g_2=(V_2, E_2, \mu_2, \varepsilon_2)$ are isomorphic if they have identical topologically and there exist a mapping $f_v : V_1 \rightarrow V_2$ such that $\mu_1(i)= \mu_2(f(i))$ for all $i \in V_1$ and for all edges $e_1(i,j) \in E_1$, there exist an edge $e_2 (f(i), f(j)) \in E_2$ and $\varepsilon_1(i,j)=\varepsilon(f(i),f(j))$.

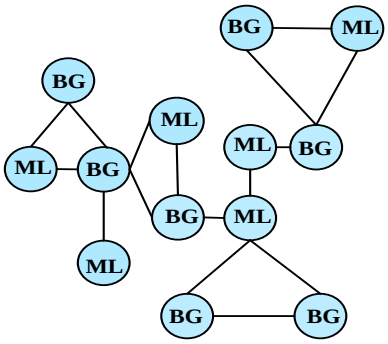
Definition 8: (**Graph Embedding**) Given a subgraph g of graph G, suppose $g \subseteq G$ such that there exist a mapping $f_v : V_g \rightarrow V_G$. An embedding φ is a pattern of vertices in V_G that are mapped to the vertices of V_g such that $e = [f(v_1), \ldots, f(v_i), \ldots, f(v_{|V_g|})]$, where $v_i \in V_g$ and $f(v_i) \in V_G$.

Example: Consider the graphs in Fig. 6.1, there are three possible embeddings of the subgraph shown in Fig. 6.1(a) in the input graph shown in Fig. 6.1(b). Embeddings shown in Fig. 6.1(c) and Fig. 6.1(e) do not overlap, so are edge-disjoint, whereas the embedding shown in Fig. 6.1(d) share vertices and edges with the other two embeddings. Thus, if overlapping is allowed, the frequency of subgraph of Fig. 6.1 (a) is 3, and if overlapping is not considered then frequency is 2.

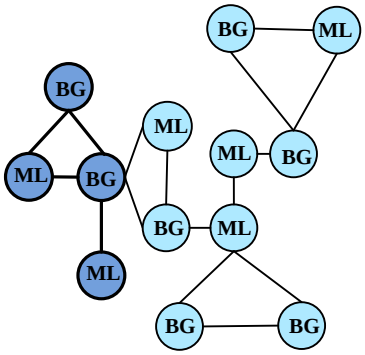
Frequent subgraphs can be overlapped based on both vertex as well as edge. Various anti-monotone metrics for computing support based on overlap are introduced in literature. Kuramochi and Karypis [153] introduced maximum independent set (*MIS*) for computing support of frequent embedding or occurrence of subgraph g in a single graph setting by considering all possible embeddings φ_i of subgraph g in graph G.



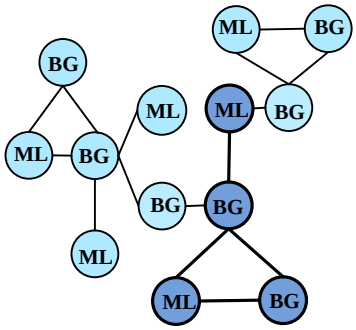
(a) Subgraph



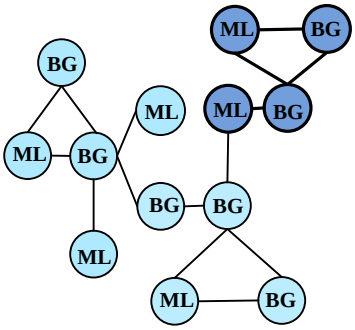
(b) Single Graph as Input



(c) Subgraph_Embedding_1



(d) Subgraph _Embedding_2



(e) Subgraph _Embedding_3

Fig. 6.1 Embeddings of a subgraph in a single graph

Further binary relationship namely harmfull overlap (*HO*) were formalized. A harmfull overlap [181] of embedding φ and φ' of subgraph g exists if $\exists v \in V_g : \varphi(v), \varphi'(v) \in (V_g) \bigcap \varphi'(V_g)$. However, the computations performed in both *MIS* and *HO* are *NP*-complete. For instance in Fig. 6.1 *MIS* gives support value of 2 while there are three embeddings of subgraph *g*.

In PaGro, minimum image (*MNI*) based support counting method is used because it avoids expensive computations and provides the superset of the results of *MIS* and *HO*. *MNI* [182] is based on the number of unique vertices in the graph *G* to which vertices of the subgraph *g* are mapped. It can formally be defined as:

Definition 9: **(MNI Support Count)** Let $g_1, ..., g_m$ be the isomorphic subgraphs and the set

of their embeddings be $E(g) = \{g_1(v), ..., g_m(v)\}$. The minimum image based support (*MNI*) σ_i of subgraph *g* in graph *G* is defined as:

$$\sigma_i(g, G) = \min_{v \in V_g} \left| \{\varphi_i(g) : \varphi_i = g_i(v) \forall g_i \in E(g)\} \right|$$

6.2 Proposed PaGro

The proposed Pattern Growth based PaGro is an efficient approach for finding exact frequent subgraphs from a single large graph without enumerating entire subgraph isomorphism. PaGro employs a simple generation-and-authentication method during each iteration on the top of distributed graph processing paradigm Pregel. The working of PaGro is divided in three phases: Graph Extension, Local subgraph Pruning and Global Subgraph Pruning. The Generation process is accomplished in Phase I and the authentication is performed during phases II and phase III, first by Workers locally and later by Master node globally.

In phase I, while performing first iteration, all frequent single vertices are enumerated to form size-1 subgraphs. Then in all the subsequent iterations, computations are performed to find frequent subgraph extensions. During each iteration, PaGro generates all candidate subgraphs which are larger in size by one vertex and edge than the subgraphs generated in previous iteration.

Further, in second and third phase, PaGro counts the frequency of each of the candidate subgraph and prunes those subgraphs which are not frequent. PaGro terminates when no extension is performed in a particular iteration. The block diagram of the proposed PaGro is given in Fig. 6.2. In the following subsections, the three phases of proposed PaGro are described in detail.

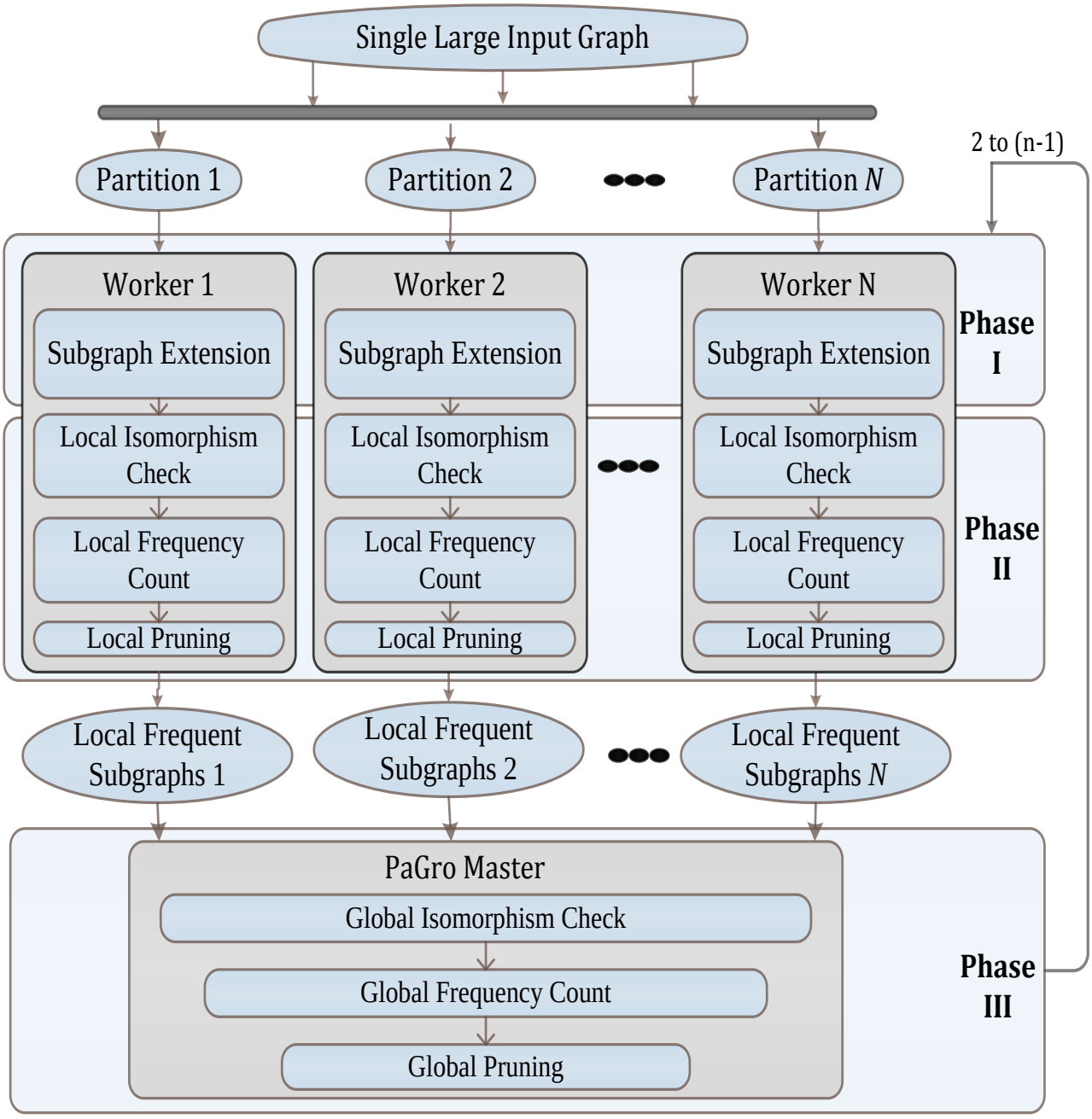


Fig. 6.2 Block Diagram of PaGro

6.2.1 Phase I: Subgraph Extension

In this phase, the frequent subgraphs are identified in multiple iterations by performing one node and edge extension in each iteration. All the computations of this phase are performed on the worker nodes as shown in Fig. 6.2. In this phase, the main focus is on generating a superset F^g_{i+1} from subset F^g_i for all candidate subgraphs.

The working of this phase is summarized in Algorithm 6.1.

Algorithm 6.1: **Phase I: *Subgraph Extension* (G_L)**

Input: A Graph G_L (V,E)
Output: All candidate subgraph F^{k+1} of G_L (V,E)

```
1.  Function VertexCompute(msg,superstep)
2.   If (Superstep=0)
3.   |  F ← ∅
4.   |  F^1 ←Frequent subgraphs of size-1 in G'
5.   |  S(F^1) ←all embeddings of F^1 in G'
6.   End
7.   Else
8.   |  S(F^k) ←all embeddings of F^k in G'
9.   |  Ext(F^k) ←possible extensions of all the embeddings of F^k
10.  |  F ←F^∪ PaGro-EXTEND(F,G',τ,d_i)
11.  End

 PaGro-EXTEND(F, G_L')
1.   F^{k+1} ←∅
2.   For each embedding φ of subgraph g in S(F^k) do
3.   | For each candidate c^{k+1} ∈ C^{k+1} do
4.   |   If F^k is not the parent of C^{k+1} then
5.   |   | Continue
6.   |   End
7.   |   For each ext(e,v) or ext(e) in Ext(g) do
8.   |   |  If ((g,e)→v) then
9.   |   |   |  If (v is frequent in Ext(g)) then
10.  |   |   |   |  C^{k+1} ← C^{k+1}∪ ext(e,v)
11.  |   |   |   End
12.  |   |   |   Else if ((g,e)→v(g)) then
13.  |   |   |   |  C^{k+1} ← C^{k+1}∪ ext(e)
14.  |   |   |   End
15.  |   |   End
16.  |   |  End
17.  |   |  F^{k+1} ← C^{k+1} ∪ F^{k+1}
18.  |   End
19.  End
     Return F^{k+1}
```

PaGro starts its execution by finding all size-1 subgraphs. Further each size-1 subgraph is extended by performing recursive depth-first-search (DFS) based extension. The PaGro-

EXTEND function given in Algorithm 6.1 is used for subgraph generation. PaGro-EXTEND takes as input set F^k of size-k subgraphs and all of the embedding's $S(F^k)$ of F^k in graph partition G' and continue as follows. For every embedding $\varphi \in S(F^k)$ of size k, the proposed PaGro recognizes and store each possible size $(k+1)$ subgraph embeddings $(\varphi+1)$. PaGro does that by exploring every possible extension $Ext(g)$ of each embedding φ of size-k subgraph g, where $g \in F^k$. The extension $Ext(g)$ contains all possible extensions $ext(e,v)$ consisting of vertex v and edge e or extensions $ext(e)$ consisting of only edge e in the case of backtracking. The extension $e' \in Ext(g)$ which is frequent in all the embeddings of subgraph g is added to size-k subgraphs F^k and are stored in candidate subgraph set C^{k+1}.

Pruning Extensions: The multiple discovery of the same vertex in depth-first tree rooted at that vertex should be avoided as each vertex at level k is linked with up to k different vertices at (k-1) level. If allowed, the algorithm will end up in performing redundant computations and will adversely affect the overall performance. The proposed PaGro eliminates such redundant computations by assigning every vertex at level k, a unique parent vertex at level (k-1).

Lemma 1: Given a subgraph set F^k, a subgraph c^{k+1} is a candidate frequent subgraph if and only if it has a generating parent in F^k.

By exploiting parent and subgraph relationship given in lemma 1, a significant amount of computations can be avoided. Hence in proposed PaGro, subgraphs only from F^k are permissible to create subgraphs in F^{k+1}. The corresponding subgraph from F^k is called the parent of respective subgraph in F^{k+1}. Thus, for every candidate subgraph $c^{k+1} \in C^{k+1}$, PaGro store only those subgraphs in C^{k+1} which have F^k as the generating parent to overcome the storage overhead.

The remaining subgraphs in C^{k+1} are size (k+1) frequent subgraphs which are obtained by performing a single edge extension $ext(e)$ or a single vertex and edge extension $ext(e,v)$ of

F^k are further used as input for proceeding the succeeding recursive call. The algorithm continues until no frequent extension is found.

6.2.2 Phase II: Local Subgraph Pruning

This phase involves three steps. In the first step, an isomorphism test is performed on candidate subgraphs to determine whether the candidate subgraphs are similar. In the second step, the support of the frequent subgraph is computed. And in third step, the candidate subgraphs having support less than the user defined minimum support are discarded. Further, the remaining subgraphs are added to the list of frequent subgraphs.

The computation of the aforementioned three steps can be carried out efficiently in centralized systems. But, in distributed environment where graph is partitioned and stored in distributed way, finding the frequent subgraphs from whole graph is a challenging task. Therefore, the computations of subgraph pruning in proposed PaGro are performed in two steps in each iteration at local level on each Worker node and then on globally Master node during Phase III.

The frequency threshold is also categorized as local frequency threshold τ' and global frequency threshold τ. As, some subgraphs might not be frequent at one graph partition but frequent in other graph partition residing at other Worker node. Thus, the value of local frequency threshold τ' should always be less than global frequency threshold τ.

The output of local pruning from each Worker is forwarded to Master node for global pruning. The pseudo codes for Local and Global pruning are given in Algorithm 6.3 and Algorithm 6.4 respectively. For Local pruning of candidate subgraphs F^{k+1} received from extension phase, an isomorphism test is performed locally at each Worker node first and later at Master node.

Optimized Isomorphism Discovery: Detecting isomorphism is computationally expensive and is NP Complete. Let the number of candidate subgraphs $|F^{k+1}|$ at stage (k+1) be p. The

isomorphism in between p subgraphs is checked ($p*p$) times. More than half of computations involved in checking isomorphism between subgraphs are of no use. They can be avoided for better performance. In PaGro, for avoiding such expensive computations, those subgraphs are identified and aggregated which could be isomorphic on the basis of lemma 2.

Lemma 2: Given a subgraph g with an embedding φ and another subgraph g' with embedding φ' in graph G. The degree of subgraph g and g' are $d(g)=\sum_{v\in g} deg(v)$ and $d(g')=\sum_{v'\in g'} deg(v')$ respectively. The subgraphs g and g' are not subgraph isomorphic if $d(g)\neq d(g')$.

For all subgraphs $g_i \in F^{k+1}$, $d(g_i)$ is computed. Subgraph g_i with embedding φ_i is stored along with $d(g_i)$ locally on Worker. Further, all stored subgraphs in F^{k+1} are reorganized by sorting in descending order of their corresponding $d(g_i)$ values.

All subgraphs with similar $d(g_i)$ values are aggregated using the Aggregator function of Pregel. According to lemma 2 the two non-isomorphic subgraphs can be identified but the subgraph isomorphism cannot be confirmed. For confirming the isomorphism in between subgraphs g and g', canonical labeling based on depth first subtree (DFS) code is used.

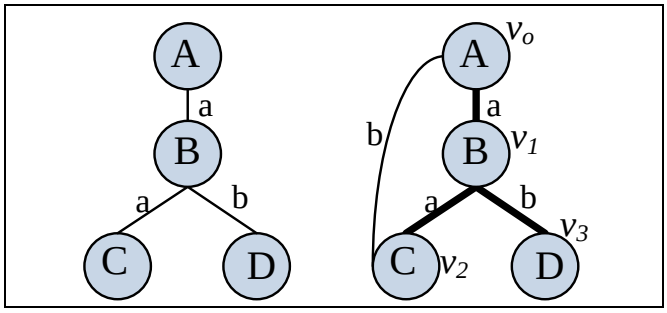


Fig. 6.3 Generation of DFS Code sequence

DFS code sequence: The minimum DFS code is a sequence of unique vertex and edge labels [54]. Fig. 6.3 shows a candidate frequent subgraph whose edges and vertices are labeled uniquely. The highlighted edges form a depth first search tree. The vertices are labeled according to their search pattern in the graph. Here, forward edge consists of all the

edges covered using DFS whereas the backward edge comprises of the remaining edges. The edges (v_0, v_1), (v_1, v_2) and (v_1, v_3) are forward edges while (v_2, v_0) is the backward edge. The DFS sequence of subgraph is ((v_0, v_1), (v_1, v_2), (v_2, v_0) and (v_1, v_3)). For a vertex v_i, backward edge will be included after the forward edge pointing to the vertex v_i. Such as when edges (v_i, v_j) and (v_j, v_k) are given and if $j<k$ then edge (v_i, v_j) comes formerly to (v_j, v_k) in DFS sequence.

Thus, for vertex v_2, edge (v_2, v_0) comes before (v_1, v_3) in DFS sequence.

Lemma 3: Given subgraphs g and g', if the minimum DFS code of g and g' are same, then g and g' are isomorphic subgraphs.

Based on Lemma 3, the isomorphic graphs are then stored in set $C_{g_i}^{k+1}$. The subgraph set F^{k+1} is then yet again initialized to store the local frequent subgraphs.

If the support of $g_i \in C_{g_i}^{k+1}$ is greater than local frequency threshold τ', then all the subgraphs $g_i \in C_{g_i}^{k+1}$ are added to F^{k+1}. The algorithm terminates after exploring all the candidate subgraphs.

Computing Support for Local Pruning: The local support is computed using the MNI support of a subgraph g_i.

Lemma 4: Given a candidate frequent subgraph g discovered in graph partition G' at worker node w_i. The local support (τ') < global support (τ).

PaGro uses the above lemma to make the graph pruning process more efficient. Given the set of embeddings $C_{g_i}^{k+1}$ consisting of candidate frequent subgraphs discovered at worker node w_i, let the corresponding set of mappings for any vertex $v_i \in$ subgraph $g_i(v)$.

The local support τ' of a subgraph g_i in partition w_i can be computed based on MNI based support count using $\sigma_{g_i} = \arg\min_{v \in V_P} \{\varphi_i(g)\}$.

Algorithm 6.2: ***LocalPruning*** (F^{k+1},$d(g_i)$, τ')

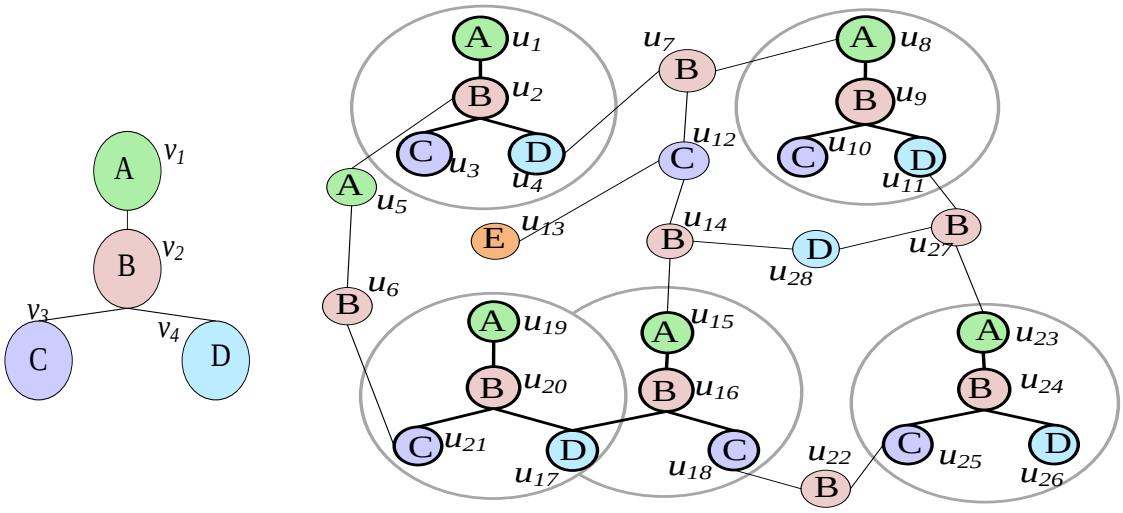
Input: All candidate subgraphs F^{k+1} at level $(k+1)$, Degree d_i, Graph G', Local frequency threshold τ'

Output: Local Frequent subgraph set F_{Wi}

1. Function **VertexCompute**(*msg*,*superstep*)
2. **For each** subgraph $g_i \in F^{k+1}$ **do**
3. | Compute $d(g_i)$
4. **End**
5. Sort all $g_i \in F^{k+1}$ in descending order of their $d(g_i)$
6. Agg(g_i) based on $d(g_i)$ similarity
7. **For each** $g \in F^{k+1}$ **do**
8. | **If** ($d(g_i) = d(g_j)$) **then**
9. | | **If** ($DFS(g_i) = DFS(g_j)$) **then**
10. | | | Declare g_i and g_j isomorphic
11. | | | Add g_i and g_j to $C_{g_i}^{k+1}$
12. | | **End**
13. | **End**
14. **End**
15. $F^{k+1} \leftarrow \emptyset$ // For storing local frequent subgraphs
16. **For each** $g_i \in C_{g_i}^{k+1}$ **do**
17. | **If** ($Support(g_i) \geq \tau'$) **then**
18. | | Add all the embeddings of g_i to F^{k+1}
19. | **End**
20. **End**
21. $I(F^{k+1}) \leftarrow$ A copy of all the frequent subgraphs $g_i \in F^{k+1}$, $d(g_i)$, $Support(g_i)$

If the support σ_i of a subgraph $g_i \in C_{g_i}^{k+1}$ is greater than or equal to τ' in a graph partition at worker node w_i, then the embeddings of subgraph g_i are added to the frequent subgraph set F^{k+1}. Formally:

$$\sigma(g_i) \geq \tau'$$



(a) (b)

MNI Table

v_1	Count	v_2	Count	v_3	Count	v_4	ount
u_1	1	u_2	1	u_3		u_4	1
u_8	1	u_9	1	u_{10}	1	u_{11}	1
u_{23}	1	u_{24}	1	u_{25}	1	u_{26}	1
u_{15}	1	u_{16}	1	u_{18}	1	u_{17}	2
u_{19}	1	u_{20}	1	u_{21}	1		
5		5		4		5	

(c)

Fig. 6.4 Support Count: (a) Candidate Subgraph g (b) Embeddings of Subgraph g (c) MNI Table for support count for $\tau = 4$

Example: In Fig. 6.4, given $\tau' = 4$ the support of the subgraph g is computed by analyzing the embeddings of *g*. The embeddings of subgraph *g* are highlighted using circles in Fig. 6.4(b). The corresponding MNI table is shown in Fig. 6.4(c). Consider the local support threshold as four. To be categorized as frequent, the support subgraph *g* should have support equal to or greater than local support threshold. Thus, each column in MNI table should have at least four distinct vertices. As all the columns have length greater than equal to four, subgraph *g* is frequent.

The algorithm terminates after exploring all the candidate frequent subgraphs and

computing their support σ_i. Thus, every worker node finds the embeddings of frequent subgraphs locally to prune those subgraphs which are not frequent in graph partition of corresponding worker node.

Computing support locally on each worker node helps in avoiding unnecessary computations which will incur while determining the support globally.

Optimizing Communication Cost: A single subgraph can have numerous numbers of embeddings in a single graph partition. It will be quite expensive if all the embeddings are passed collectively to master node for the computation of global support. The communication cost will be even higher for large graphs with millions or billions of edges.

Thus, to reduce the communication cost in PaGro, a single subgraph image is passed to master for each of the isomorphic frequent subgraphs. For example, consider a subgraph g_1 with embedding φ_1 in graph G whose local support is σ_{g1}. Instead of passing σ_{g1} number of g_1 subgraphs to master, only an image of g_1 subgraph is passed. Image $I(g_1)$ consists of single copy of subgraph g_1, local support σ_{g1} of g_1 and degree of subgraph $d(g_1)$. Formally:

$$I(g_i)=\{ g_i, \sigma_{gi}, d(g_i)\}$$

The images of all the frequent subgraphs are aggregated in $I(F^{k+1})$ such that:

$$I(F^{k+1}) =\{ I(g_1) \cup I(g_2) \cup \ldots.. \cup I(g_h)\}$$

here, h is the number of images of isomorphic frequent subgraph sets in $I(F^{k+1})$. The image of locally frequent subgraphs $I(F^{k+1})$ is then passed to the master node by each worker node. Then, the second subgraph pruning step Global Pruning is initiated.

6.2.3 *Global Subgraph Pruning*

The pseudo code for performing Global Pruning is given in Algorithm 6.3. The Global Pruning function of PaGro takes as input the Image $I(F^{k+1})$ from all the worker nodes and a

global frequency threshold τ. Consider the number of workers be w For the images $I(F^{k+1})$ from w worker nodes, the subgraph pruning is performed. The global subgraph pruning is performed in similar way as local subgraph pruning was performed in Algorithm 6.2 (Lines 5-14) using degree values and DFS code. The isomorphic subgraphs of g_i are then stored in $f_{g_i}^{k+1}$ together.

Algorithm 6.3: ***GlobalPruning*** $(I(F^{k+1}), \tau)$

Input: Image of frequent subgraphs $I(F^{k+1})$ from Workers w_i, Global frequency threshold τ
Output: Global Frequent subgraph set F

1. Function **MasterCompute**(*msg*,*superstep*)
2. Sort all $I(g_i) \in I_w(F^{k+1})$ in descending order of their $d(g_i)$
3. **For each** $I(F^{k+1}) \in w_i$ **do**
4. | **For each** $I(g_i) \in I(F^{k+1})$ **do**
5. | | **If** ($d(g_i) = d(g_j)$) **then**
6. | | | **If** ($DFS(g_i) = DFS(g_j)$) **then**
7. | | | | Declare g_i and g_j be isomorphic
8. | | | | Add g_i and g_j to $f_{g_i}^{k+1}$
9. | | | **End**
10. | | **End**
11. | **End**
12. | *Support* ($f_{g_i}^{k+1}$)←*Support* (C_{gi}^{k+1})+*Support* (C_{gj}^{k+1})
13. **End**
14. **If** (*Support* ($f_{g_i}^{k+1} \geq \tau$) **then**
15. | Add all $g_i \in f_{g_i}^{k+1}$ to F
16. **End**
17. **Else**
18. | Discard subgraph g_i
19. **End**

Further, the global frequency of subgraph g_i is computed by adding the local support of g_i from all the workers that contain subgraph g_i. Formally:

$$Support(g_i) = \sum_{i=1}^{w} Support(g_i)$$

If the global frequency of frequent isomorphic subgraph g_i is greater than or equal to global frequency threshold τ, the subgraph g_i is declared as frequent globally. The global frequent subgraph set F is created by adding the images of the subgraphs that are frequent globally. The master then passes F to all the worker nodes as size k subgraph for obtaining size (k+1) subgraph using subgraph extension in phase II.

6.3 Approximate Subgraph Mining : Ap-FSM

In this section, the proposed Ap-FSM is presented. Ap-FSM discovers representative approximate frequent subgraphs from a single massive graph. The proposed approach employs a simple generation-and-verification method during each iteration on the top of Pregel. The verification is carried out in two steps in each iteration, first by Workers locally and later on Master node globally.

The block diagram of the proposed Ap-FSM is shown in Fig. 6.5. The working of Ap-FSM is divided in three phases: representative graph selection, subgraph mining and subgraph pruning. The first phase of representative graph selection phase is executed only once in the beginning whereas the second and third phases are iterative. In phase 1, AP-FSM selects a sample from the original graph to make the computations fast and easy. In second phase, first all frequent single vertices are enumerated to form size-1 subgraphs. Then, the iterative computations are performed for subgraph extension. In each iteration, Ap-FSM generates all possible subgraphs which are larger in size by one vertex and one edge from the subgraphs generated in previous iteration.

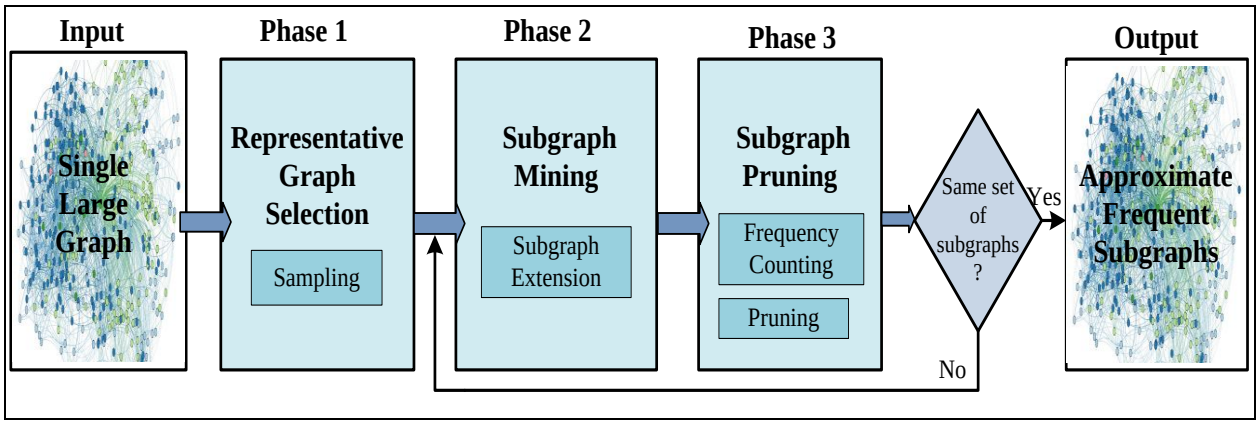


Fig. 6.5 Block Diagram of the proposed approach

Further, in the third phase, Ap-FSM adds up the frequency of every candidate subgraph and prunes those subgraphs which are not frequent. Ap-FSM terminates when no extension is performed in a particular iteration.

Representative Graph Selection: A novel graph sampling algorithm named G-Samp is proposed in order to select an appropriate representative graph from a single large graph. Representative graph *G'* is a sample of a given single large graph *G* such that *G'* holds similar graph properties of the original graph *G*. More formally:

Definition 10: The sampling of graph $G=(V,E)$ with sampling factor δ is to generate a subgraph $G'=(V',E')$ such that $|V'|=\delta|V|$ where, $V' \subseteq V$ and $E' \subseteq E$.

The real-world large graphs follow power-law-degree distribution. Thus to acquire similar properties, the sampled graph *G'* should have similar degree distribution as of graph *G*. Therefore, the algorithm starts with computing the degree of the vertices of graph *G*. All the vertices with their vertex id's and degree are stored in a list for further processing.

The vertex list is further sorted as per the descending order of their degree values. The vertices with the similar degree are aggregated and stored together in list l_i, where *i* represent the degree of vertices in list. For example, vertices with degree 6 are stored in list l_6. The precise working of G-Samp is given in Algorithm 6.4.

Algorithm 6.4: ***G-Samp* (*G*, δ)**

Input: A Graph *G* (*V*,*E*), Sampling Rate δ ,

Output: A sampled graph *G'*=(*V'*,*E'*) of size *x*

1. Rank *N* vertices of graph based on the decreasing order of their degree d_i
2. Aggregate vertices with similar degree in list l_i.
3. *G'*← ∅
4. *h*= δ *100
5. *x* = *h*% *N*
6. **While** |*V'*| ≤ *x* **do**
7. **For** each vertex list l_i **do**
8. Select *h*% vertices
9. **End**
10. l'_i ← Selected *h*% vertices
11. **End**
12. *G'*← $\sum_{i=0} l'_i$

A pre-defined sampling factor δ is used to select the vertices from the graph for sampling. For example, for δ = 0.2, 20% of the vertices are selected from each degree-wise aggregated vertex list l_i. From each vertex list l_i, *h%* vertices are selected randomly. The selected *h*% vertices from list l_i are stored in a new list l'_i. Vertices stored in lists l'_i are aggregated and are stored in sampled graph *G'* such that graph *G'* will have the same number of vertices from each degree-wise aggregated vertex list l_i.

Thus, the selected vertices will have same degree structure as of the original graph *G* =(*V*,*E*) such that the selected representative graph *G'*=(*V'*,*E'*) will also follow the similar but less dense power law degree distribution as followed by original graph *G* =(*V*,*E*). These selected vertices set *V'* with the edges incident on the vertices of *V'* together form the sampled graph *G'*.

Table 6.1: Dataset used with characteristics

Dataset	|Vertices|	|Edges|	L(v)	Type
Social Graphs				
LiveJournal	4,847,571	68,993,773	30	Directed
Twitter	11,316,811	85,331,846	25	Directed
Collaboration Networks				
US Patents	3,774,768	16,518,947	418	Directed
DBLP	317,080	1,049,866	40	Undirected

6.4 Performance Evaluation

In this section, the performance evaluation of PaGro and Ap-FSM is performed.

6.4.1 Evaluation of PaGro

The experiments are performed to evaluate the efficiency of the algorithm on 10 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 node is considered as master node, and rest are considered as slave nodes. The master node is also used as slave node. Hadoop 2.6.0 and Giraph 1.1.0 are used for performing experiments.

6.4.1.1 Dataset Characteristics

The experiments are performed over four real life datasets shown in Table 6.1. Two datasets are from social networks and other two are from collaboration networks.

LiveJournal[176]: It is on-line community which allows its members to maintain the details about journals, individual and group blogs. It also allows users to state which other members are their friends. Nodes in this dataset are users of LiveJournal, and directed edges characterize friendship among members.

Twitter[183]: The twitter graph data models the Twitter follow data based on a snapshot taken in 2009. A node represents a user and a directed edge indicates that a user is followed

by another user. This graph does not contain labels for vertices so each vertex is assigned a label randomly based on Gaussian distribution from the pool of 568 labels.

US patent [184]: It is a citation network of patents registered with the United States Patent and Trademark Office. In this dataset, each node is a patent labeled with the patent class and an edge represents a citation. The network contains self-citations also which resulted in formation of self-loops.

DBLP [176]: It is a co-authorship network of the DBLP computer science bibliography. The vertices represent authors and an undirected edge among two vertices is there if there has been a publication of at least one paper among corresponding authors together.

6.4.1.2 Effect of varying support threshold

The support threshold τ is the main estimation metric to determine when a subgraph is frequent. Decreasing the value of τ will increase the number of possible candidate frequent subgraphs exponentially. Therefore, result in exponential decrease in the performance of the graph mining algorithm. An efficient mining algorithm should be able to discover frequent subgraphs for lower support values also. The efficiency of the algorithm is determined by the total execution time.

In this experiment, the effect of varying support threshold is analyzed. This experiment is conducted over the four real life datasets mentioned above. The number of worker nodes are fixed to ten and analyzed the processing time elapsed on worker nodes and master node. The effect of varying both local support threshold τ' and global support threshold τ is examined.

(a) Local support Threshold

The processing time on worker nodes elapsed in identifying frequent subgraphs locally is shown in Fig. 6.6. Average processing time of all the worker nodes is considered by varying the local support threshold τ'. Here, the discovered subgraphs are candidate frequent

subgraphs that are frequent locally on corresponding worker nodes but might not be frequent globally.

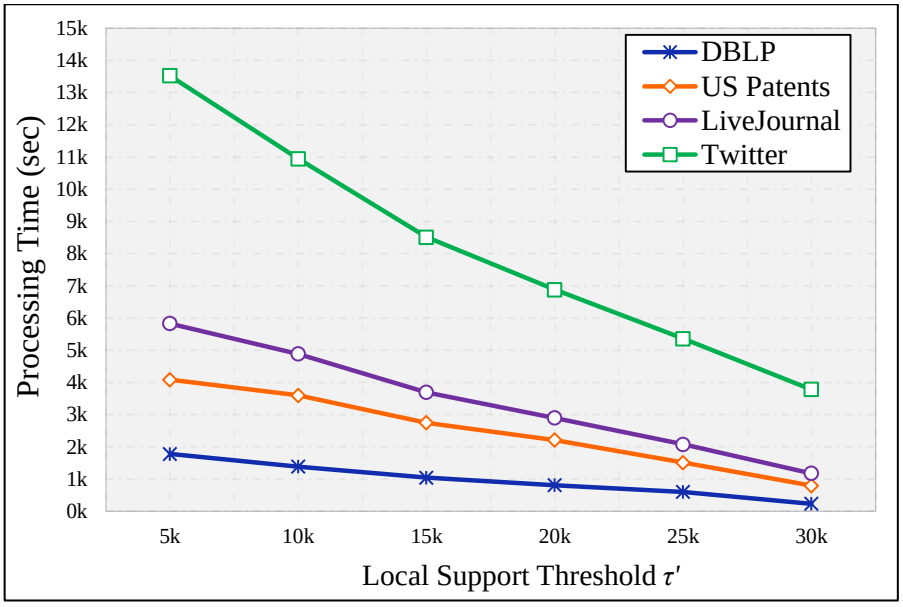


Fig. 6.6. Effect of varying local support threshold τ'

As shown, for all the datasets the processing time decreases with the increase in local support threshold τ'. For DBLP, the number of candidate frequent graphs discovered is relatively high for lower local support threshold values. The count decreases gradually when τ' = 30k. Thus, the local support threshold τ' is considered as 10k for all the subsequent experiments performed on DBLP. For US patent dataset, the number of candidate frequent subgraphs discovered locally is an order of magnitude higher than that of DBLP for smaller τ' values. For a balanced discovery of frequent subgraphs, the value of τ' = 15k for further experiments on US Patent dataset. The LiveJournal graph dataset has only 30 labels for vertices. Thus, the number of frequent subgraphs discovered is gradually high. Hence, the local support threshold for LiveJournal is kept as 20k for the rest of the experiments. Twitter is the largest dataset considered with 25 vertex labels. The number of frequent subgraphs discovered is so much higher in comparison to other datasets considered. As shown in Fig. 6.6 for τ' = 30k, the number of subgraphs are relatively comparable with other datasets. Thus the value of τ' is considered as 30k for further experiments.

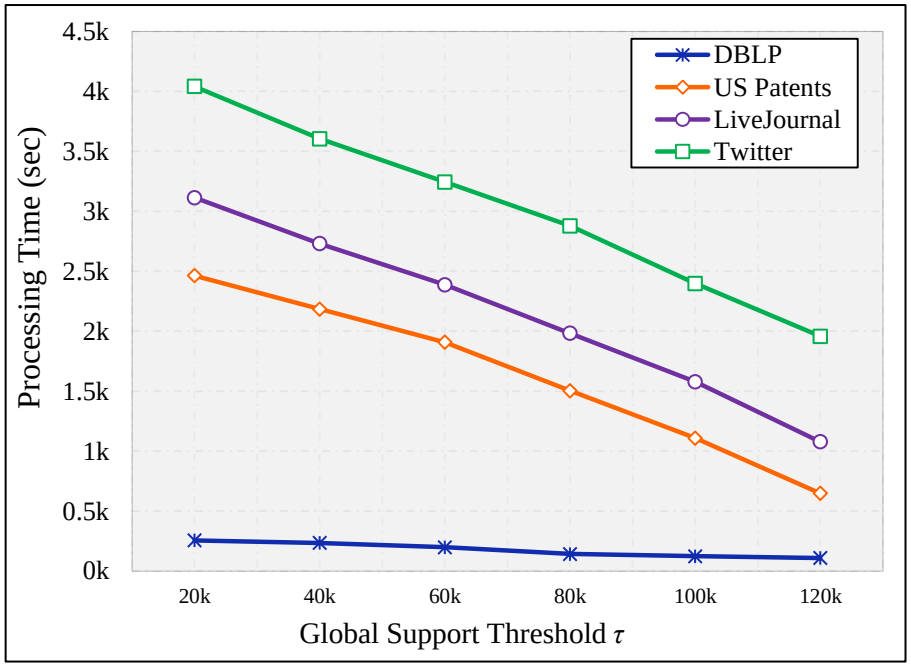


Fig. 6.7 Effect of varying global support threshold τ when local support threshold τ = 10k

(b) Global support Threshold

A subgraph is frequent globally if it is frequent in at least two of the worker partitions. Thus the value of the global support threshold τ is varied starting from 20k and the value of local support threshold τ' is considered as 10k.

Fig. 6.7 shows the effect of varying the global support threshold τ for all the considered datasets. The processing time here includes the communication cost, global subgraph isomorphism computation and pruning of subgraphs. The results by varying the global support threshold are similar as the results by varying local support threshold. The number of discovered frequent subgraphs decreases exponentially with the increase in support value. But, the difference between elapsed time in between Twitter dataset and LiveJournal dataset is smaller in comparison.

6.4.1.3 *Comparison with Existing Approaches*

For performance evaluation, proposed PaGro is compared with baseline algorithm, Arabesque [158] and GRAMI [154]. The performance of proposed PaGro along with other

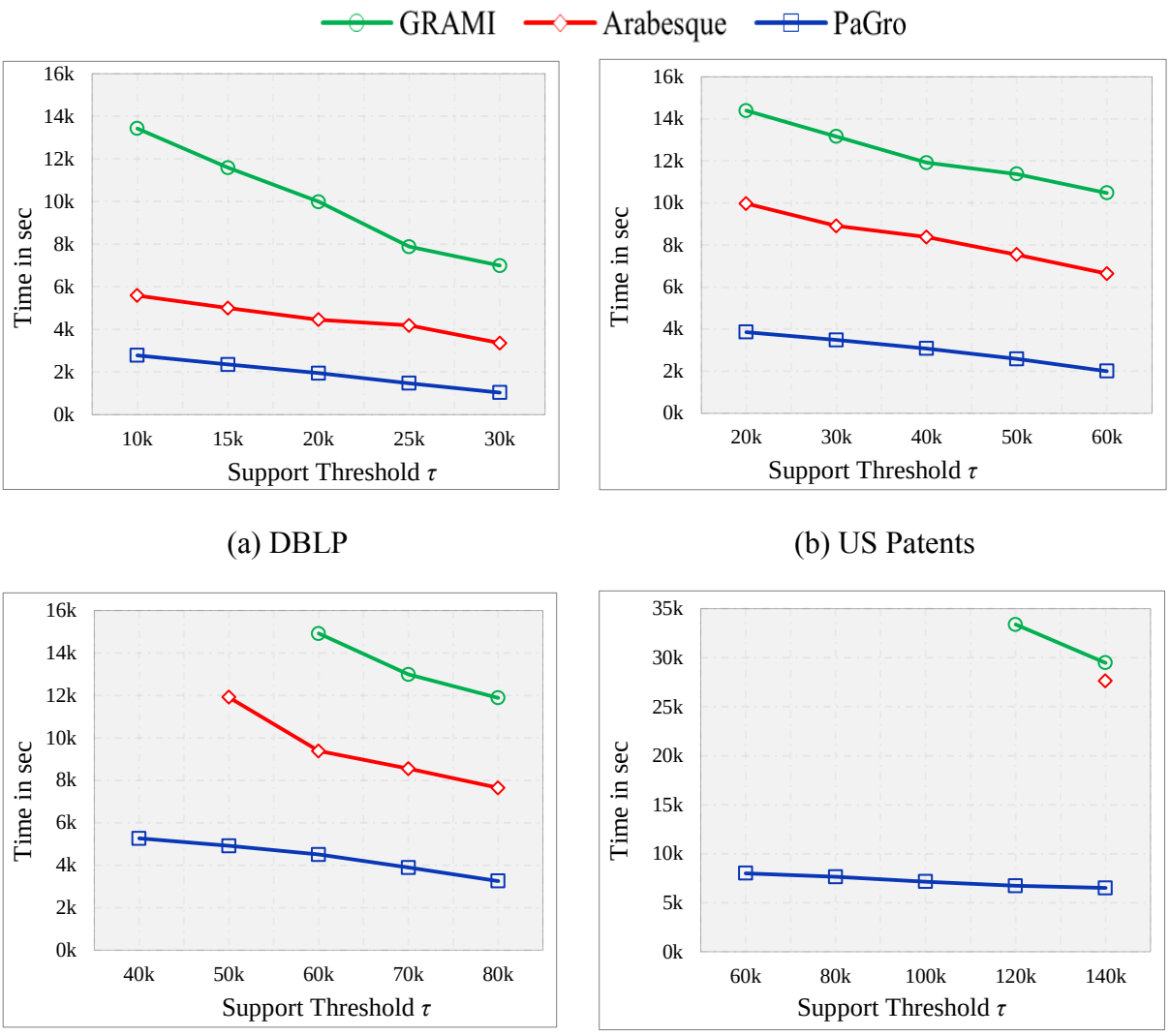
competent algorithms is compared by varying the support threshold τ. For Pagro the local support threshold τ' at each worker node is kept as 5k, 10k, 20k and 30k for DBLP, US Patent, LiveJournal and Twitter network dataset respectively.

(a) Running Time

The number of frequent subgraphs grow exponentially when the overall support threshold τ decreases. With lesser τ values, the subgraphs discovered in both intermediate levels and final level increases subsequently. Therefore, the processing time of all the algorithms also decrease exponentially with the increase in τ. PaGro outperforms all the competent algorithms particularly for smaller values of support threshold τ in all the cases as shown in Fig. 6.8.

GRAMI is a centralized algorithm thus, crashes while handling large data and lower support values. For τ = 20k, PaGro is three order of magnitude faster than the GRAMI and an order of magnitude faster than Arabesque for DBLP and US Patents dataset.

For the large datasets LiveJournal and Twitter with lower support values 50k and 80k respectively, GRAMI and Arabesque were not able to produce any result. Arabesque also smashes especially for large graphs and smaller support values because it incurs communication and storage overhead associated with the large number of embeddings. For Twitter dataset, both Arabesque and GRAMI fail to handle smaller support values as shown in Fig. 6.8 (d).



(a) DBLP (b) US Patents

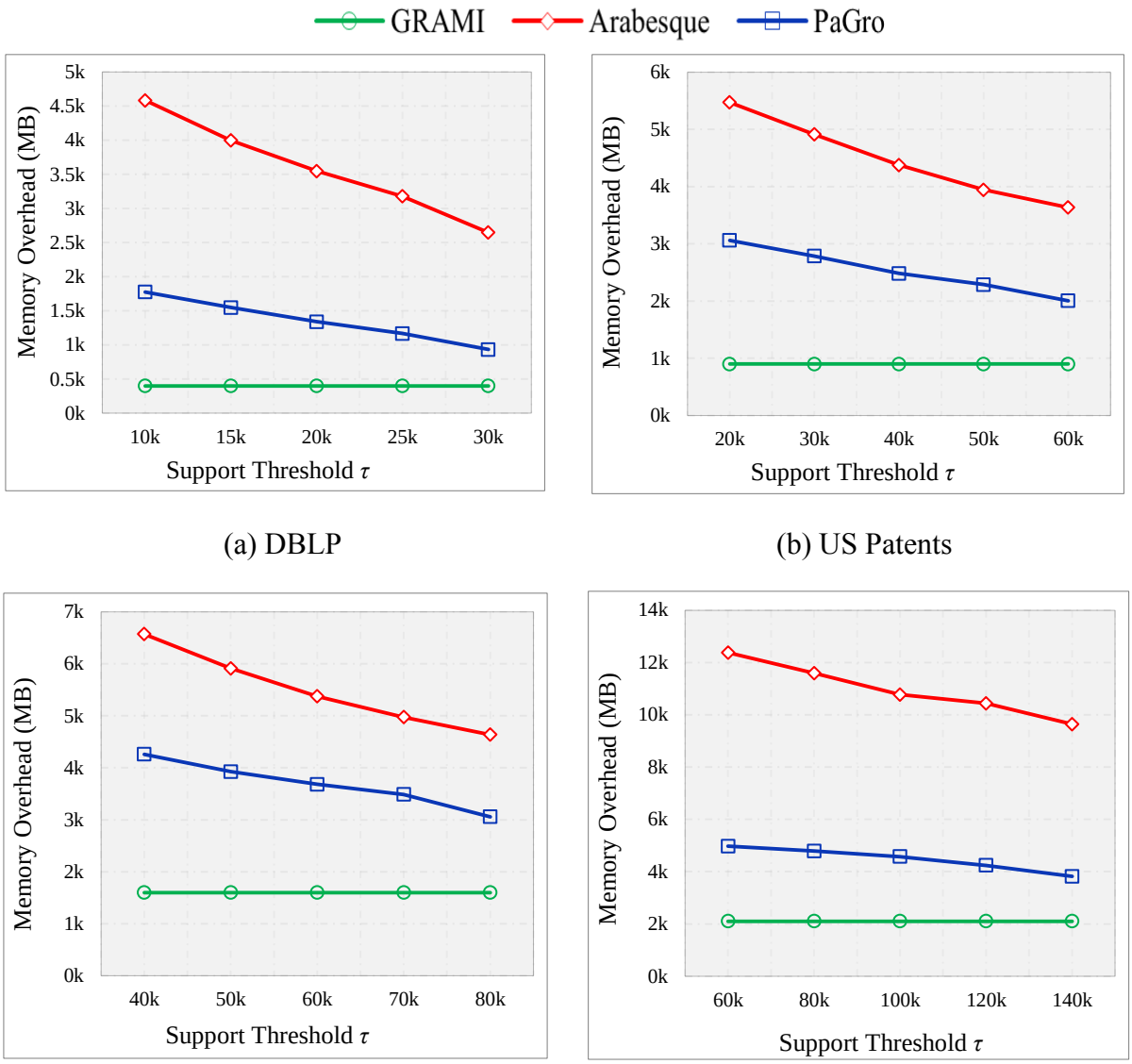
(c) LiveJournal (d) Twitter

Fig. 6.8 Comparing Performance in terms of Processing Time (sec)

Arabesque only shows result when the support value is 140k. While, GRAMI discovers frequent subgraphs only for support values 120k and 140k. The running time of both the algorithms is five orders of magnitude higher than the proposed PaGro algorithm.

(b) Memory Overhead

The memory overhead shows the number of intermediate subgraphs stored in the process of discovering frequent subgraphs. Fig. 6.9 shows the memory consumption in MB of each system for different support threshold values. The memory required for running distributed algorithms PaGro and Arabesque is the average memory usage of all the worker nodes.



(a) DBLP

(b) US Patents

(c) LiveJournal

(d) Twitter

Fig. 6.9 Comparing Performance in terms of Memory Overhead (MB)

Whereas for centralized algorithm GRAMI, it is measured by the memory incurred by the single machine. GRAMI do not store the intermediate results thus; requires memory for storing the graph only. Therefore, it remains constant irrespective of the support value.

However in PaGro and Arabesque for lower support values, the candidate subgraphs tend to produce many extensions that have potential to be included in the list of candidate frequent subgraphs.

Subsequently, frequent candidate subgraphs that are large in size may have fewer valid extensions with respect to the support threshold. For small support threshold values in DBLP, there is an increase in the number of the frequent subgraphs and thus an exponential increase in the number of intermediate candidate subgraphs that need to be stored and checked for frequency. The same trend appears for all the other datasets. For Twitter dataset, when a higher number of subgraphs is discovered, Arabesque requires much more memory than the proposed PaGro algorithm as shown in Fig. 6.9 (d).

6.4.1.4 *Optimizations*

In this experiment, the effect of the optimization used in PaGro is analyzed. Both the optimizations are applied on top of the baseline algorithm. Formally, the following techniques are implemented on the same distributed environment and are compared with PaGro:

- BL: The Baseline algorithm mentioned in Algorithm 2.2 is implemented.
- ID: The optimization technique used for isomorphism discovery in PaGro is employed on the top of Baseline algorithm
- CC: It incorporates the optimization used for minimizing the communication overhead with the baseline algorithm

The effect of the optimization on the baseline algorithm is studied and all the aforementioned approaches are compared with the performance of PaGro for all the massive datasets in Fig. 6.10.

The performance is shown in the form of bar plots, where the red color boundary along a bar indicates that the algorithm do not finishes its execution in the time limit.

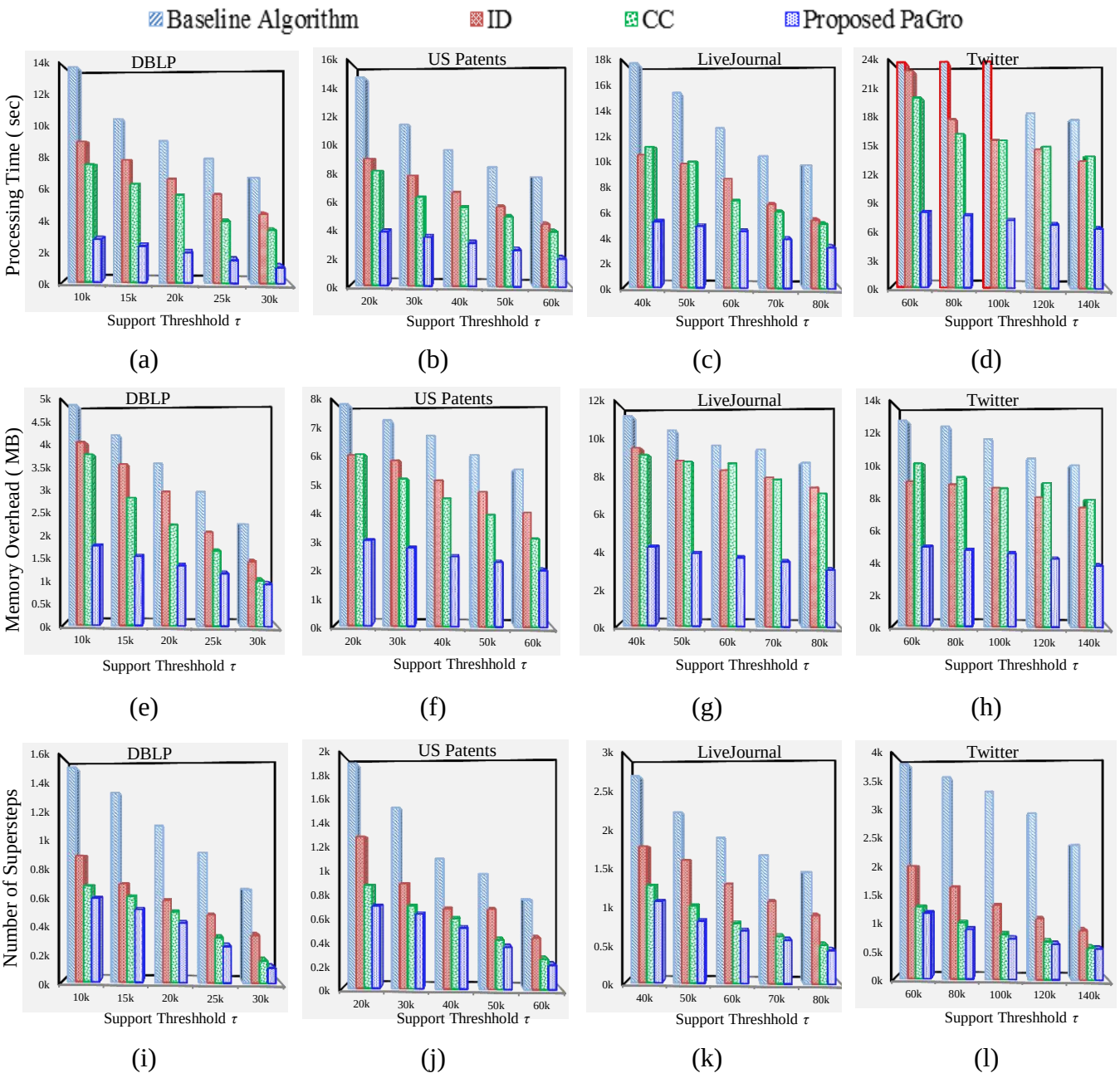


Fig. 6.10 Evaluating Performance of Optimizations: Processing Time (a)-(d), Memory Overhead (e)-(f), Number of Superstep (i)-(l)

First the processing time is evaluated of all the aforementioned techniques and Pagro shown in Fig. 6.10 (a)-(d). It is observed that the proposed PaGro outperforms the baseline algorithm for all the datasets.

Both the optimizations when applied on Baseline algorithms improve its performance significantly. ID reduce the processing time by reducing the comparisons required for identifying isomorphic graphs while, CC improves the performance by passing only the

image of all the candidate subgraphs rather than passing the whole set of candidate frequent subgraphs. For Twitter with support values 100k, 120k and 140k, the optimizations CC and ID improve the performance of baseline equally. The baseline algorithm is not able to finish its execution in even 24k seconds.

Next, the effect of these techniques on memory requirements is analyzed in Fig. 6.10 (e)-(h). The results show that Baseline requires more memory than both ID and CC. In baseline algorithm, same graph can be discovered multiple times. By employing ID on baseline the auto-isomorphism is avoided which results in less memory overhead. Also, employing CC over baseline avoid the memory overhead by storing only an image of frequent subgraph on master. However, for large graphs such as LiveJournal and Twitter ID outperforms CC for higher support values. PaGro collectively has much lesser memory overhead.

Fig. 6.10 (i)-(l) plots the number of supersteps recorded while implementing these techniques. It can be noticed that the proposed PaGro takes lesser number of supersteps in comparison to baseline algorithm.

6.4.1.5 Scalability

Fig. 6.11 shows the execution time of the proposed PaGro with the increased number of worker nodes over the considered graph datasets. As anticipated, increase in the number of processors resulted in decreased run time. As GRAMI is a centralized algorithm, it is not considered in the computation of node scalability. Thus, the performance of PaGro is compared with Baseline algorithm and Arabesque.

The support threshold is kept as 20k, 40k, 80k, and 140k for DBLP, US Patent, LiveJournal and Twitter dataset respectively. For twitter dataset τ is taken as 140, because only Arabesque gives results in that case. For all the datasets, linear run time is observed when the number of worker nodes is increased as shown in Fig. 6.11.

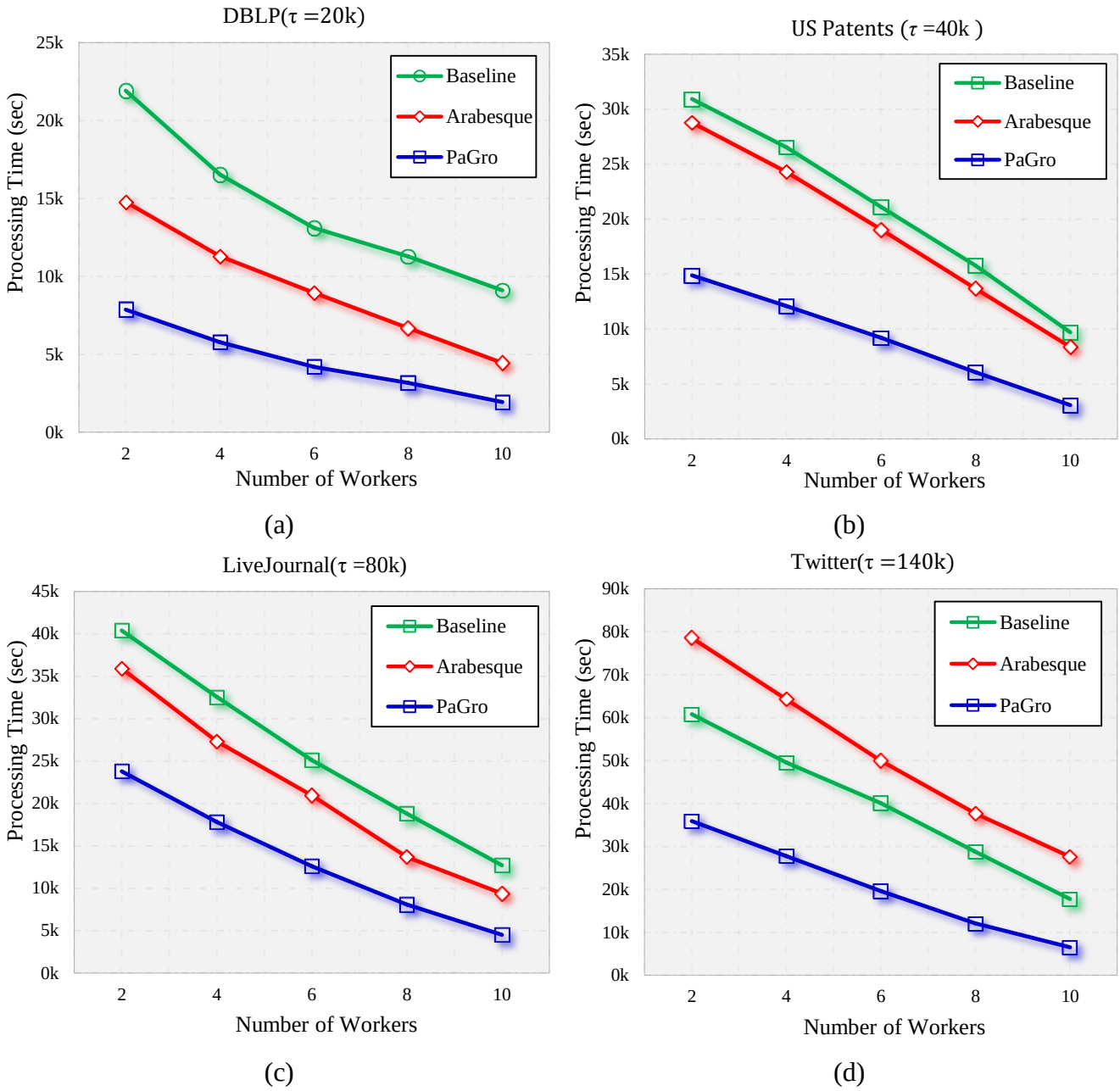


Fig. 6.11 Scalability by varying the number of Worker nodes: (a) DBLP τ=20k, (b) US Patents with τ =40k, (c) LiveJournal with τ =60k, (d) Twitter with τ =140k.

The scalability of an algorithm is measured in terms of Speedup. It is a measure for evaluating the efficiency of parallel algorithm with the increasing number of processors. It can be calculated as:

$$Speedup = \frac{Run\ Time_1}{P\ *\ Run\ Time_P}$$

where, P is the number of processors used. For ideal parallelization, Speedup = 1. Speedup behavior of PaGro is calculated using the above formulae.

PaGro achieves high Speedup efficiency of 45.8, 60.4, 61.6, and 65.9% for DBLP, US Patent, LiveJournal and Twitter dataset respectively. While Arabesque achieves 40.7, 41, 43.7 and 32.2% DBLP, US Patent, LiveJournal and Twitter dataset respectively. Therefore, PaGro achieves higher speedup efficiency and scalability for all the graph datasets.

6.4.2 Evaluation of Ap-FSM

The experiments are performed over one synthetic and three real-life datasets from domains such as social networks and collaboration networks. The statistics of considered graphs and the approximate graph selected are given in Table 6.2.

Table 6.2. Features of Real World Datasets used in Experiments

	Original graph description		**Approximate graph Description**		**#Node**
Dataset	**|Nodes|**	**|Edges|**	**|Nodes|**	**|Edges|**	**Labels**
US Patents	3,774,768	16,518,947	1,078,505	4,235,627	320
LiveJournal	4,847,571	68,993,773	1,310,154	17,248443	34
Twitter	41,652,230	1,468,365,182	8,010,044	271,919,547	1,053

Ap-FSM is implemented using the Giraph version 1.1.0 and Hadoop 2.6.0. Its performance has been evaluated using a Hadoop Cluster composed of 8 Machines equipped with an Intel Xeon processor with 2.9GHz processor and 8GB of RAM memory with Ubuntu 16.04 operating system installed. One of the machines is taken as Master node and rest are taken as Worker nodes. Master node also acts as a Worker node.

6.4.2.1 Performance Evaluation of G-Samp

The performance of the proposed G-Samp in representative graph selection phase is compared with the state-of-art sampling algorithms such as Random Node (RN) samples and Random Walk (RW) sampling. The characteristics of the sampled graphs are examined by varying the sampling rate δ. Let the size of the original graph as 1, sampling rate δ=0.4 means

that the size of sample is 40% of the original graph. The characteristics of the sample S_i are compared with respect to the characteristics of original graph G using:

$$\frac{p(S_i)}{p(G)} \times 100$$

where, $p(S_i)$ and $p(G)$ are particular characteristic of sample S_i and original graph G respectively. The performance of all the algorithms is compared on the basis of quality of samples prepared in terms of Degree similarity, Clustering-Coefficient similarity and Betweenness similarity and is shown in Fig. 6.12, Fig. 6.13 and Fig. 6.14 respectively.

A. Degree Similarity

Degree of a vertex is the number of links incident upon a vertex. The degree similarity characteristics of sampling algorithms with the original graph are shown in Fig. 6.12. In US Patents dataset, for δ = 0.1, the best degree similarity is generated by the proposed approximate phase. However, for δ = 0.5, Random walk sampling produces better samples in terms of the degree distribution. Whereas, the performance of proposed approach remains consistently efficient. For LiveJournal graph dataset, the proposed approach outperforms all the competent algorithms for all the sampling rates. In this case, the random node sampling performs worst of all the algorithms.

For the Twitter Dataset, the proposed approach again performs well as shown in Fig. 6.12. It can be observed from the Fig. 6.12 that, the random walk sampling has the most inconsistent results in terms of degree distribution and proposed approach depict high degree similarity.

A. Clustering Coefficient

Clustering Coefficient is a measure of the degree to which the vertices of a graph tends to form a cluster. In the case of average global clustering coefficient, there is a significant

variation in the performance of the considered sampling algorithms. The plot in Fig. 6.13 shows the average fraction of triangles existing around the vertices of degree *d*.

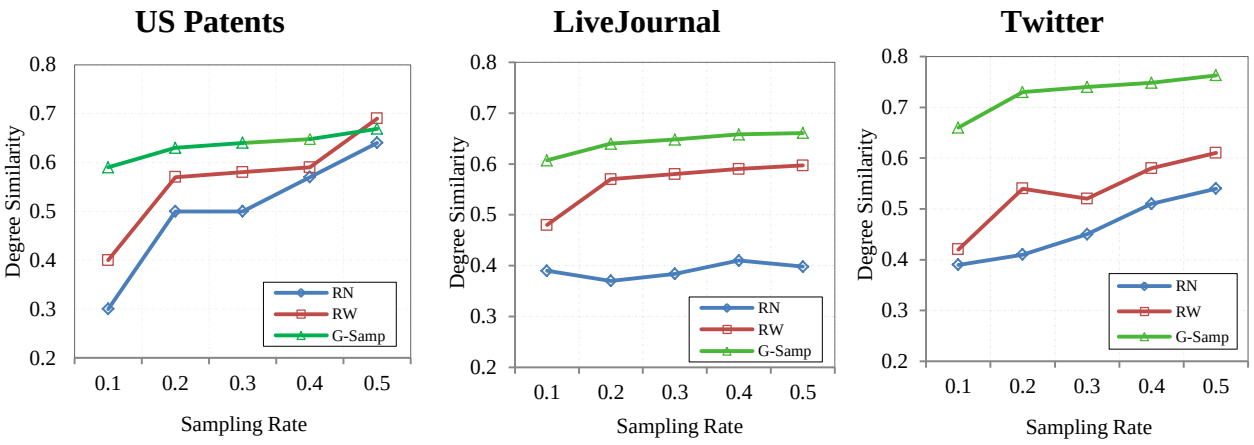


Fig. 6.12 Performance of sampling algorithms in terms of Degree Similarity

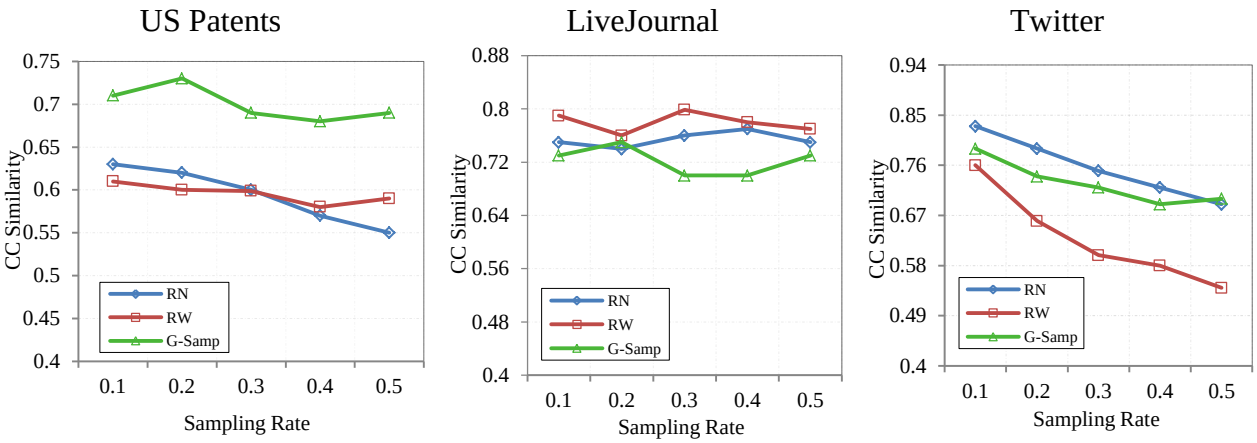


Fig. 6.13 Performance of sampling algorithms in terms of Clustering Coefficient

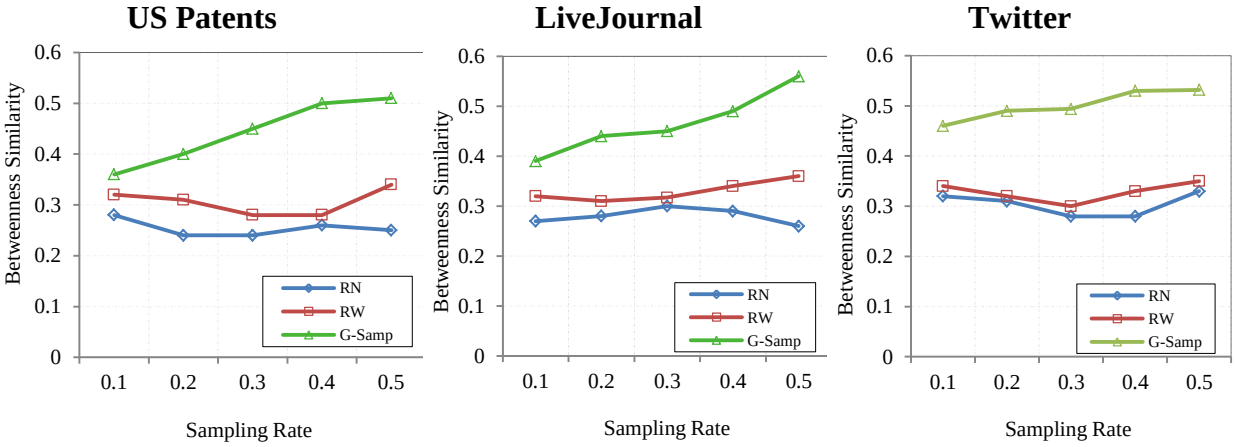


Fig. 6.14 Performance of sampling algorithms in terms of Betweenness similarity

Instinctively, it gives an idea about the community structure of the graph. Random node sampling surprisingly has very high clustering coefficient accuracy for the Twitter graph and has low accuracy for US Patent graph. However, the proposed G-Samp provides more consistent results for all the sampling rates and is highly accurate for all the considered datasets.

B. Betweenness similarity

Betweenness computes the shortest path in between the vertices of the graph. As the proposed approach considers the degree of vertices for sampling thus, has smaller diameter and better betweenness as depicted in Fig. 6.14. For all the datasets, proposed approach performs better than the random walk sampling and random node sampling.

6.4.2.2 Comparison with Existing Approaches

For performance evaluation, proposed Ap-FSM is compared with baseline algorithm, Arabesque [158], GRAMI [154], AGRAMI [185] and gApprox [136] . Baseline algorithm, Arabesque and

GRAMI are exact subgraph mining algorithms while AGRAMI and gApprox are approximate subgraph mining algorithm. The performance of proposed Ap-FSM along with other competent algorithms is compared by varying the support threshold τ. For fair evaluation, the exact subgraph mining algorithms are evaluated for sample sizes 20%, 30% and 40% as shown in Fig. 6.15.

Ap-FSM outperforms the competent algorithms particularly for smaller values of support threshold τ in all the cases. Arabesque smashes especially for larger samples and lesser support values because it incurs communication and storage overhead associated with the large number of embeddings.

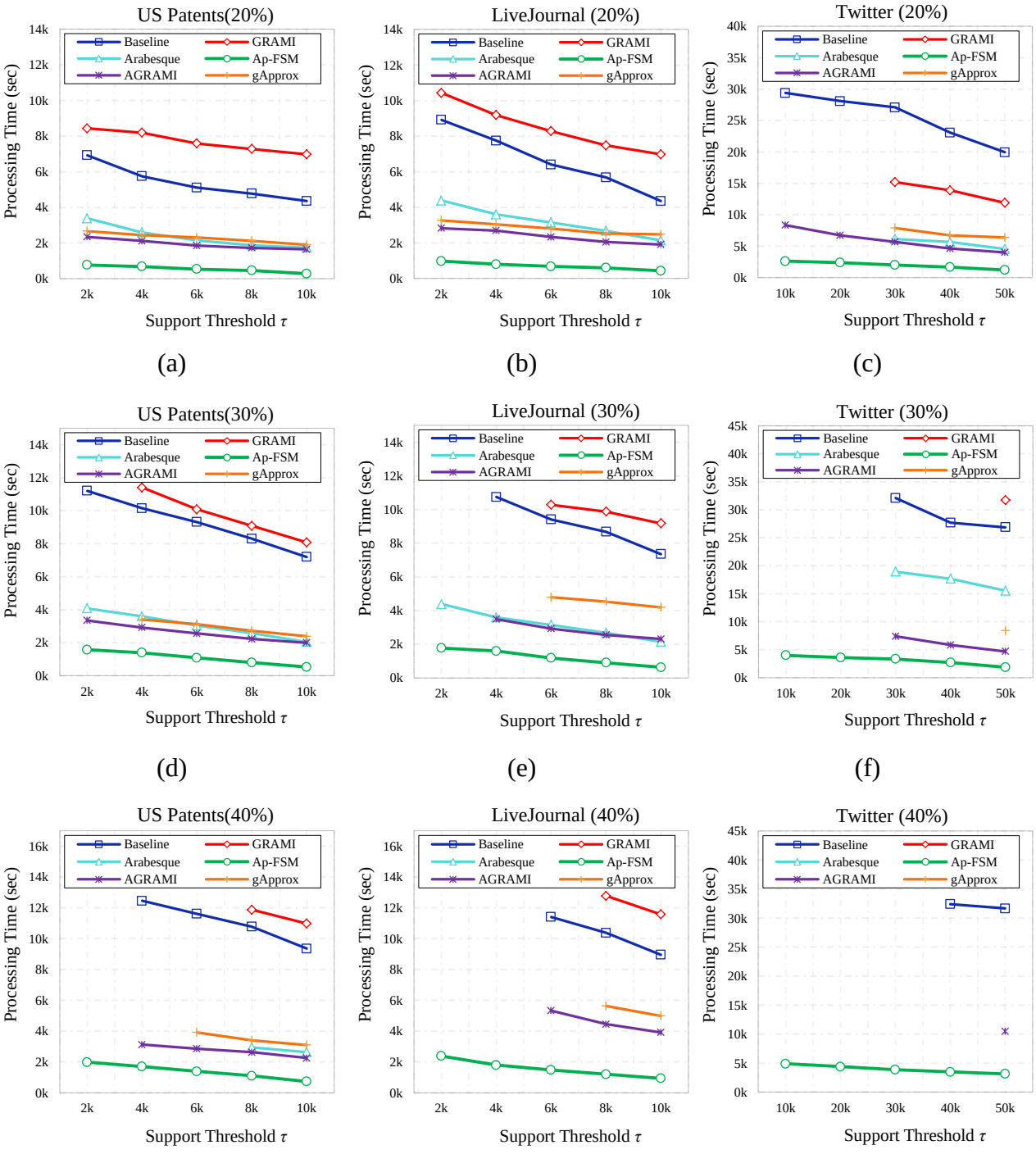


Fig. 6.15 Evaluating Performance in terms of Processing Time (sec) on various sample sizes: (a)-(c) 20%, (d)-(f) 30%, (g)-(i) 40%

GRAMI, AGRAMI and gApprox are centralized algorithm thus, crashes while handling large data and lower support values. For δ= 20%, Ap-FSM is three order of magnitude faster than the GRAMI and an order of magnitude faster than Arabesque, AGRAMI and gApprox for all the considered datasets. For Twitter dataset with δ= 20%, Arabesque, GRAMI, AGRAMI and

gApprox fails to handle smaller support values as shown in Fig. 6.15(c). For 30% sampled US Patent dataset, GRAMI crashes for τ =2k while AGRAMI and gApprox are found to be computationally expensive. For δ=30%, the baseline algorithm also crashes for smaller support values for LiveJournal and Twitter dataset. Approximate subgraph mining algorithms AGRAMI and gApprox crashed for most of the support threshold values.

However, the proposed Ap-FSM performs well for all the datasets and is computationally very less expensive in comparison to the other counterparts. When δ= 40% is taken, all the competent algorithms suffers in handling such a large amount of data. However, the proposed Ap-FSM performs exceptionally well as shown in Fig. 6.15(g), Fig. 6.15(h) and Fig. 6.15(i).

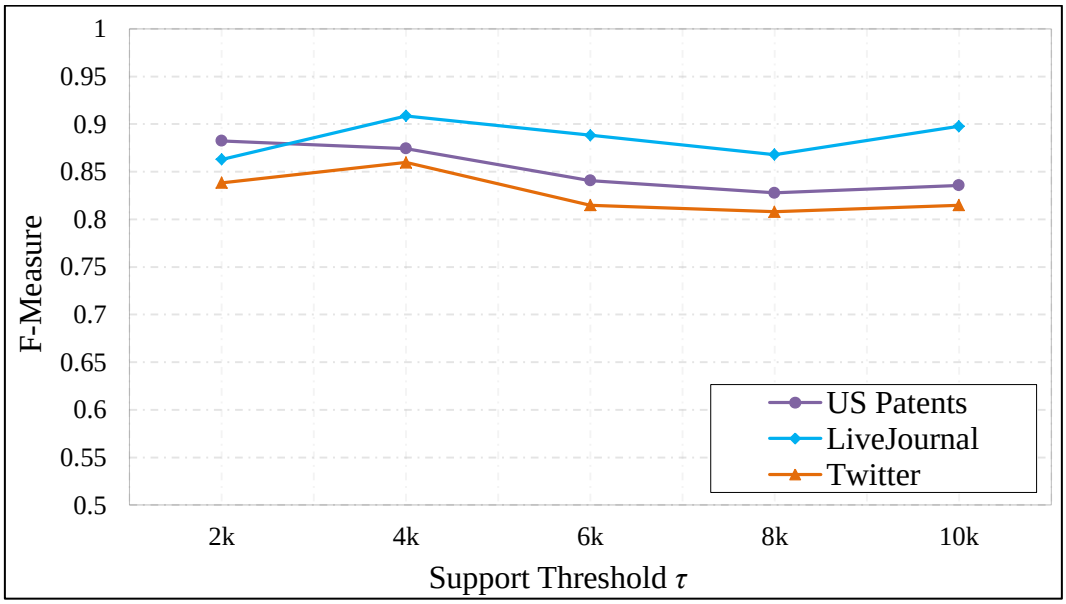


Fig. 6.16 Accuracy of proposed Ap-FSM in terms of F-Measure

6.4.2.3 Accuracy

The accuracy of the subgraphs discovered by Ap-FSM is assessed by comparing the subgraphs produced by the Approximate subgraph mining Ap-FSM have sample rate 0.4 with the exact subgraph mining (ESM). F-measure is considered for the evaluation of accuracy of Ap-FSM. If the subgraph is detected as frequent in both Ap-FSM and ESM, it is considered as true positive. If the subgraph is frequent in ESM but not in AP-FSM, it will be treated as false

negative and subgraphs frequent in Ap-FSM but not in ESM are considered as false positives. Fig. 6.16 depicts the F-measures for US Patents and LiveJournal dataset for various support values.

It can be seen that Ap-FSM has high F-Measure for all the datasets. For LiveJournal dataset, value of F-Measure 0.9 when τ = 4k. Ap-FSM despite of taking much less time in comparison to exact subgraph mining algorithm discovers frequent subgraphs with high accuracy.

7. CONCLUSION AND FUTURE DIRECTIONS

This chapter presents the key findings, the research contribution and possible future directions in the domain of big data and graph mining. This chapter addresses the contribution in section 7.1 and future work in section 7.2

7.1 Main Contribution

In this thesis, the main focus is on designing models and algorithms for mining patterns from a single large graph. A parallel and distributed computing environment Pregel is used for performing analysis. Pregel based Giraph which runs on the top of the Hadoop cluster is used to process large graph datasets. An efficient algorithm named 'PGFC' is proposed for finding fuzzy clusters by amending the structure of the classical Fuzzy C-means algorithm for large graph data. Instead of selecting the cluster centers randomly, degree of the vertices is taken as the selection criteria for the initialization of cluster centers. The proposed PGFC is scalable and is able to handle graph dataset of various sizes efficiently. The performance of the proposed algorithm is compared with Pregel based semi-clustering, K-means and FCM algorithms by applying it to synthetic and real-life graph datasets in a distributed environment. It is observed that PGFC show better performance for fuzzy clustering in terms of partition coefficient and conductance. 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, optimal clustering results are achieved by PGFC with high CPU utilization. However, the proposed PGFC has certain limitations. PGFC requires a pre-defined number of clusters to be mentioned in starting. But,

for large graphs of real-world, if the number of clusters in such networks is predefined, the actual clusters might break up or get merged.

To overcome the aforementioned limitation of PGFC, an influence based fuzzy clustering algorithm named ‘PFCA’ is designed. The proposed PFCA first selects the candidate cluster heads based on their influence in the network and then determines the number of clusters by analyzing the graph structure using PageRank algorithm and modularity. PFCA identifies both disjoint and fuzzy clusters efficiently and finds membership of only those vertices which lie in more than one cluster. The performance of proposed PFCA is compared with the existing clustering counterparts in terms of run time, scalability, F-measures and modularity. The results are validated using six real life datasets of complex networks. Through experimental analysis, it is proved that PFCA is efficient and scale up linearly with the size of network. Accuracy of the proposed PFCA is measured using the ground truth data. It is observed that PFCA efficiently finds the clusters of varying sizes with high accuracy. But, PFCA also has a drawback. While analyzing the structure of the graph, it may result in the loss of certain vertices which do not lie in any of the cluster. Such information can be crucial in some applications and can be used for some other tasks such as outlier detection.

To overcome the information loss, distributed autoencoder based layered model named DFuzzy is proposed. 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-tune 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. 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. Thus, DFuzzy exploits both

approaches so that the shortcomings of either approach can be compensated by the advantages of the other. 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.

For finding frequent subgraphs two approaches are proposed. First, an exact subgraph mining algorithm named PaGro is proposed by leveraging the operative communication primitives for better scalability. A two-step hybrid approach is developed in PaGro for optimization of subgraph pruning task at both local and global levels to avoid the excess communication overhead. The NP-Complete graph isomorphism problem is optimized for fast graph processing. The storage overhead is also minimized by reducing the number of embeddings to be stored. The experiments are performed over real life graph datasets and comparison of proposed PaGro is performed with baseline algorithm, GRAMI and Arabesque. The results show that PaGro outperforms all the state-of-art subgraph mining algorithms in terms of processing time and is at least an order of magnitude faster than the competent algorithms. The efficiency of optimizations proposed in PaGro is also shown by comparing with the baseline algorithm. Furthermore, it is shown that PaGro has high data scalability and can efficiently handle graphs with billions of edges.

However, in many applications, getting the faster approximate results is significant than getting the exact results. If that large graph is analyzed at small scale, the useful insights can be extracted in lesser time. Thus, the working of PaGro is amended for performing approximate frequent subgraph mining in Ap-FSM by exploiting sampling for faster processing. A novel sampling approach named G-Samp is proposed in Ap-FSM for the selection of an approximate subgraph while capturing the original graph properties for

convenient and relatively easy analysis. The experiments were performed over real life datasets and compared the performance of proposed Ap-FSM with baseline algorithm, GRAMI, Arabesque and with two approximate subgraph mining algorithm AGRAMI and gApprox. The results show that Ap-FSM outperforms the considered state-of-art subgraph mining algorithms in terms of processing time. The efficiency of optimizations used in Ap-FSM is also shown by comparing with its various variants. Furthermore, it is shown that Ap-FSM has high data scalability and can efficiently handle graphs with billions of edges.

7.2 Future Directions

The research described in the thesis has a number of promising directions for future research. A suitable partitioning technique could be applied to the proposed Ap-FSM for better load balancing. Also, the work can be extended to handle queries in distributed graph environment. Further, the proposed algorithm can be applied to various applications like credit card fraud detection, recommendation system, etc. for obtaining interesting patterns.

- **Pattern mining in dynamic graphs:** Graphs generated from domains like social networks, collaboration networks tend to change dynamically with time. Mining real-time graphs is a very active area of research. Graph mining approaches such as clustering and frequent subgraph mining can be used to identify interesting patterns in dynamic graphs. Some examples include analyzing the properties of time-evolving graphs, mining dynamic frequent subgraphs and evolving recommendation system.
- **Mining images using interesting points:** Many images can be modeled as graphs where the vertices can represent the interesting points or key entities from images and the edges represent relation between the entities. In such applications, the similarity measures between the entities can be represented by allocating edge weights. The future challenge

is to mine key patterns from images by providing compact graph representations for images with effective edge and vertex labeling. It can help in image classification.

- **Mining patterns from multi-attributed graphs:** Big data can incorporate many features therefore, the relationships among the vertices of large graphs can represent multiple features. Selecting key features and assigning weight to features are challenging tasks that can play a vital role in the whole process. It is also possible to incorporate some feature selection approaches in graph mining. It can be used to group the entities with similar features or to find outliers.
- **Graph based Text/Document Mining:** Mining patterns from large collection of documents or from text is an active area of research. It covers applications such as web mining and text mining. Approaches for keyword or semantic based graph representations can be extended for large graph setting. Moreover, the accurate and meaningful labeling of vertices and edges is vital to achieve effective graph based text mining.

BIBLIOGRAPHY

[1] X. Wu, X. Zhu, and S. Member, "Data Mining with Big Data," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[2] O. Mason and M. Verwoerd, "Graph Theory and Networks in Biology," *IET systems biology*, vol. 1, no. 2, pp. 89–119, 2007.

[3] M. Ventresca and D. Aleman, "Efficiently Identifying Critical Nodes in Large Complex Networks," *Computational Social Networks*, vol. 2, no. 6, pp. 1–16, 2015.

[4] L. Wang and J. Hopcroft, "Community Structure in Large Complex Networks," in *the Proceedings of International Conference on Theory and Applications of Models of Computation*, 2010, pp. 455–466.

[5] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang, "Complex networks : Structure and dynamics," *Physics reports*, vol. 424, no. 4, pp. 175–308, 2006.

[6] J. C. Long, F. C. Cunningham, P. Carswell, and J. Braithwaite, "Patterns of collaboration in complex networks: the example of a translational research network," *BMC Health Services Research*, vol. 14, no. 225, pp. 1-10, 2014.

[7] C. S. Thaker, D. S. Jat, A. Machanja, and E. Nepolo, "Big data: Emerging technological paradigm and challenges," in *the Proceedings of IEEE International Conference on Emerging Trends in Networks and Computer Communications*, 2015, pp. 138–143.

[8] V. Morabito, "Big data and analytics: Strategic and organizational impacts," *Big Data and Analytics: Strategic and Organizational Impacts*, pp. 1–183, 2015.

[9] W. Fan and A. Bifet, "Mining Big Data : Current Status , and Forecast to the Future," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 1–5, 2013.

[10] A. Suinesiaputra, P. Medrano-Gracia, B. R. Cowan, and A. A. Young, "Big Heart Data: Advancing Health Informatics Through Data Sharing in Cardiovascular Imaging," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 4, pp. 1283–1290, 2015.

[11] C. F. Sun L., Yamin M., Mushi C., Liu K., Alsaigh M., "Information analytics for healthcare service discovery," *International Journal of Ambient Computing and Intelligence*, vol. 9, no. 1, pp. 457–477, 2014.

[12] X. Hong-lin, Y. Han-bing, G. Cui-fang, and Z. Ping, "Social Network Analysis Based

on Network Motifs," *Journal of Applied Mathematics*, vol. 2014, pp. 1–6, 2014.

[13] S. Pippal, V. Sharma, S. Mishra, and D. S. Kushwaha, "An efficient schema shared approach for cloud based multitenant database with authentication and authorization framework," in *the Proceedings of IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 2011, pp. 213–218.

[14] H. Zhu, J. Zhang, M. T. Kim, A. Boison, A. Sedykh, and K. Moran, "Big data in chemical toxicity research: The use of high-throughput screening assays to identify potential toxicants," *Chemical Research in Toxicology*, vol. 27, no. 10, pp. 1643–1651, 2014.

[15] L. Duan and Y. Xiong, "Big data analytics and business analytics," *Journal of Management Analytics*, vol. 2, no. 1, pp. 1–21, 2015.

[16] H. N. Rothberg and G. S. Erickson, "Big data systems: knowledge transfer or intelligence insights?," *Journal of Knowledge Management*, vol. 21, no. 1, pp. 92-112, 2017.

[17] G. Chetty and M. Yamin, "A distributed smart fusion framework based on hard and soft sensors," *International Journal of Information Technology*, vol. 9, no. 1, pp. 19–31, 2017.

[18] D. Che, M. Safran, and Z. Peng, "From Big Data to Big Data Mining: Challenges, Issues, and Opportunities," in *the Proceedings of International Conference* on *Database Systems for Advanced Applications*, 2013, pp. 1–15.

[19] X. Wu, X. Zhu, G. Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[20] S. Schneeweiss, "Learning from Big Health Care Data," *Perspective*, vol. 363, no. 1, pp. 1–3, 2010.

[21] U. Kang and C. Faloutsos, "Big Graph Mining : Algorithms and Discoveries," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 29–36, 2013.

[22] Z. Deng, X. Zhu, D. Cheng, M. Zong, and S. Zhang, "Efficient kNN classification algorithm for big data," *Neurocomputing*, vol. 195, pp. 143-148, 2016.

[23] S. Moens, E. Aksehirli, and B. Goethals, "Frequent Itemset Mining for Big Data," in *the Proceedings of IEEE International Conference on Big Data*, 2013, pp. 111–118.

[24] X. Gao, Shang; Chen, Alan; Rahmani, Ali; Zeng, Jia; Tan, Mehmet; Alhajj, Reda; Rokne, Jon; Demetrick, Douglas; Wei, "Multi-scale modularity and motif distributional effect in metabolic networks," *Current Protein and Peptide Science*, vol. 17, no. 1, pp. 82–92, 2016.

[25] Rahmani, A., Afra, S., Zarour, O., Addam, O., Koochakzadeh, N., Kianmehr, K., Alhajj, R. and Rokne, J.., "Graph-based approach for outlier detection in sequential data and its application on stock market and weather data," *Knowledge-Based Systems*, vol. 61, pp. 89–97, 2014.

[26] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel Graph Partitioning for Complex Networks -Balanced Graph Partitioning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2625–2638, 2017.

[27] W. X. Lu, C. Zhou, and J. Wu, "Big social network influence maximization via recursively estimating influence spread," *Knowledge-Based Systems*, vol. 113, pp. 143–154, 2016.

[28] M. Wang, C. Wang, J. X. Yu, and J. Zhang, "Community Detection in Social Networks : An In-depth Benchmarking Study with a Procedure-Oriented Framework," *Proceedings of the VLDB Endowment*, vol. 8, no. 10, pp. 998–1009, 2015.

[29] N. S. Ketkar, L. B. Holder, and D. J. Cook, "Subdue: compression-based frequent pattern discovery in graph data," in *the Proceedings of 1st International Workshop on Open source data Mining: Frequent Pattern Mining Implementations*, 2005, pp. 71–76.

[30] C. Borgelt and M. R. Berthold, "Mining molecular fragments: finding relevant substructures of molecules," in *the Proceedings of IEEE International Conference on Data Mining*, 2002, pp. 51–58.

[31] S. E. Schaeffer, "Graph Clustering," *Computer Science Review*, vol. 1, pp. 27–64, 2007.

[32] S. Malek, M. Golsefid, M. Hossien, and F. Zarandi, "Fuzzy Community Detection Model in Social Networks," *International Journal of Intelligent Systems*, vol. 30, pp. 1227–1244, 2015.

[33] H. Zhou, J. Li, J. Li, F. Zhang, and Y. Cui, "A graph clustering method for community detection in complex networks," *Physica A: Statistical Mechanics and its Applications*, vol. 469, pp. 551–562, 2017.

[34] A. Ghosh, N. S. Mishra, and S. Ghosh, "Fuzzy clustering algorithms for unsupervised change detection in remote sensing images," *Information Sciences*, vol. 181, no. 4, pp. 699–715, 2011.

[35] Y. Zhou, H. Cheng, and J. X. Yu, "Graph Clustering Based on Structural / Attribute Similarities," in *Proceedings of the VLDB Endowment*, 2009, vol. 2, no. 1, pp. 718–729.

[36] D. Arthur, “k-means ++ : The Advantages of Careful Seeding,” in *the Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027–1035.

[37] J. C. Bezdek, R. Ehrlich, and W. Full, “FCM: The Fuzzy C-means clustering algorithm,” *Computers & Geosciences*, vol. 10, no. 2–3, pp. 191–203, Jan. 1984.

[38] H. Wang, Z. Xu, and W. Pedrycz, “An overview on the roles of fuzzy set techniques in big data processing: Trends, challenges and opportunities,” *Knowledge-Based Systems*, vol. 118, pp. 15–30, 2016.

[39] G. Palla and I. Dere, “Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society,” *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.

[40] S. Fortunato, “Community detection in Graphs,” *Physics Reports*, vol. 486, pp. 75–174, 2010.

[41] C. Laclau and M. Nadif, “Hard and fuzzy diagonal co-clustering for document-term partitioning,” *Neurocomputing*, vol. 193, pp. 133–147, 2016.

[42] G. Bello-Orgaz, J. J. Jung, and D. Camacho, “Social Big Data: Recent Achievements and New Challenges,” *Information Fusion*, vol. 28, pp. 45–59, 2016.

[43] Y. Özbay, R. Ceylan, and B. Karlik, “A fuzzy clustering neural network architecture for classification of ECG arrhythmias,” *Computers in Biology and Medicine*, vol. 36, no. 4, pp. 376–388, 2006.

[44] O. Kesemen, Ö. Tezel, and E. Özkul, “Fuzzy C-means clustering algorithm for directional data (FCM4DD),” *Expert Systems with Applications*, vol. 58, pp. 76–82, 2016.

[45] A. Király, Á. Vathy-fogarassy, and J. Abonyi, “Geodesic distance based fuzzy c-medoid clustering – searching for central points in graphs and high dimensional data,” *Fuzzy Sets and Systems*, vol. 286, pp. 157–172, 2015.

[46] P. G. Sun, L. Gao, and S. S. Han, “Identification of overlapping and non-overlapping community structure by fuzzy clustering in complex networks,” *Information Sciences*, vol. 181, pp. 1060–1071, 2011.

[47] T. Nepusz, A. Petrczi, L. Ngyessy, and F. Bazs, “Fuzzy Communities and the concept of Bridgeness in Complex Networks,” *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 77, no. 1, pp. 1–13, 2008.

[48] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A.Y. Zomaya , S. Foufou and A. Bouras, “A Survey of Clustering Algorithms for Big Data : Taxonomy and Empirical Analysis,” *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 3, 2014.

[49] J. Baumes, M. Goldberg, M. Krishnamoorthy, M. Magdon-Ismail, and N. Preston,

"Finding communities by clustering a graph into overlapping subgraphs," in *the proceedings of the International Conference on Applied Computing*, 2005, pp. 97–104.

[50] J. Huan, W. Wang, J. Prins, and J. Yang, "SPIN: Mining Maximal Frequent Subgraphs from Graph Databases," in *the Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, no. 1, pp. 581–586.

[51] F. Schreiber and H. Schw, "Frequency Concepts and Pattern Detection for the Analysis of Motifs in Networks," in *the Proceedings of Transactions on Computational Systems Biology III*, 2005, pp. 89–104.

[52] A. Inokuchi, T. Washio, and H. Motoda, "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data," *Principles of Data Mining and Knowledge Discovery*, vol. 1910, pp. 13–23, 2000.

[53] H. P. Hsieh and C. Te Li, "Mining temporal subgraph patterns in heterogeneous information networks," in *the Proceedings of 2nd IEEE International Conference on Privacy, Security, Risk and Trust*, 2010, pp. 282–287.

[54] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," *Journal of Chemical Information and Modeling*, vol. 53, no. 9, pp. 1689–1699, 2002.

[55] A. Dhiman and S. K. Jain, "Optimizing Frequent Subgraph Mining for Single Large Graph," *Procedia Computer Science*, vol. 89, pp. 378–385, 2016.

[56] J. Li, Z. Zhong, J. Z. Huang, and S. Feng, "Balanced Parallel FP-Growth with MapReduce," in *the Proceedings of IEEE International Conference on Granular Computing (GrC)*, 2011, pp. 875–878.

[57] K. Wang, X. X. B, H. Jin, P. Yuan, F. Lu, and X. Ke, "Frequent Subgraph Mining in Graph Databases Based on MapReduce," in *the Proceedings of 10th Asia-Pacific Services Computing Conference on Advances in Services Computing, APSCC*, 2016, pp. 464–476.

[58] W. Lin, X. Xiao, and G. Ghinita, "Large-Scale Frequent Subgraph Mining in MapReduce," in *the Proceedings of IEEE International Conference on Data Engineering*, 2014, pp. 844–855.

[59] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *the Proceedings of the Symposium on Operating Systems Design and Implementation*, 2004, pp. 137–149.

[60] G. Malewicz *et al.*, "Pregel : A System for Large-Scale Graph Processing," in *the Proceedings of the ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–145.

[61] C. Aggarwal and H. Wang, *Managing and mining graph data*, vol. 40. 2010.

[62] D. J. Cook and L. B. Holder, *MINING GRAPH DATA*. Wiley, 2007.

[63] A. Seth, H. Agarwal, and A. R. Singla, "Reliability Estimation of Services Oriented Systems Using Adaptive Neuro Fuzzy Inference System," *Journal of Software Engineering and Applications*, vol. 7, pp. 581–591, 2014.

[64] M. Kaya and R. Alhajj, "A clustering algorithm with genetically optimized membership functions for fuzzy association rules mining," in *the Proceedings of 12th IEEE International Conference on Fuzzy Systems*, 2003, vol. 2, pp. 881–886.

[65] A. Bonato and Y. Tian, "Complex Networks and Social Networks," *Advances in Network Analysis and its Applications*, pp. 1–18, 2013.

[66] I. X. Y. Leung, P. Hui, P. Liò, and J. Crowcroft, "Towards real-time community detection in large networks," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 79, no. 6, pp. 1–10, 2009.

[67] S. Halder and Y. K. Lee, "Supergraph based periodic pattern mining in dynamic social networks," *Expert Systems with Applications*, vol. 72, pp. 430–442, 2017.

[68] S. . Hill, B. . Srichandan, and R. . Sunderraman, "An iterative MapReduce approach to frequent subgraph mining in biological datasets," in *the Proceedings of ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, 2012, pp. 661–666.

[69] A. Karci, "Efficient automatic exact motif discovery algorithms for biological sequences," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7952–7963, 2009.

[70] L. Wang, Z. Wang, S. Zhao, and S. Tan, "Stock market trend prediction using dynamical Bayesian factor graph," *Expert Systems with Applications*, vol. 42, no. 15–16, pp. 6267–6275, 2015.

[71] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang, "Boosting for graph classification with universum," *Knowledge and Information Systems*, vol. 50, no. 1, pp. 53–77, 2017.

[72] S. Agarwal, A. S. Bisht, D. Singh, and N. P. Pathak, "A novel neural network based image reconstruction model with scale and rotation invariance for target identification and classification for Active millimetre wave imaging," *Journal of Infrared, Millimeter, and Terahertz Waves*, vol. 35, no. 12, pp. 1045–1067, 2014.

[73] H. Chun *et al.*, "A graph clustering method for community detection in complex networks," *Knowledge and Information Systems*, vol. 469, no. 1, pp. 718–729, 2017.

[74] C. Nabti and H. Seba, "A Simple Algorithm for Subgraph Queries in Big Graphs," *rXiv preprint arXiv:1703.05547* (2017).

[75] M. A. Bhuiyan and M. Al Hasan, “An Iterative MapReduce based Frequent Subgraph Mining Algorithm,” *Transactions on Knowledge and Data Engineering*, vol. 4347, no. OCTOBER, pp. 1–17, 2013.

[76] I. Journal, O. F. Advanced, and R. Science, “Analysis of Large Graph Partitioning and Frequent Subgraph Mining on Graph Data,” vol. 6, no. 7, pp. 29–40, 2015.

[77] A. Perez-Suarez, J. F. Martinez-Trinidad, J. A. Carrasco-Ochoa, and J. E. Medina-Pagola, “OClustR: A new graph-based algorithm for overlapping clustering,” *Neurocomputing*, vol. 121, pp. 234–247, 2013.

[78] U. Kang, C. Tsourakakis, and C. Faloutsos, “PEGASUS: A Peta-Scale Graph Mining System-Implementation and Observations,” in *the Proceedings of 9th IEEE International Conference on Data Mining*, 2009, pp. 229–238.

[79] “Apache Giraph,” 2016. [Online]. Available: http://giraph.apache.org/. [Accessed: 10-Nov-2017].

[80] S. Lee, S. Kang, H. Kim, and J. Min, “An Efficient Parallel Graph Clustering Technique Using Pregel,” in *the Proceedings of International Conference on Big Data and Smart Computing (BigComp)*, 2016, pp. 370–373.

[81] P. Agarwal, M. Ramanath, and G. Shroff, “Relationship Queries on Large graphs using Pregel,” in *the Proceedings of International Conference on Management of Data (COMAD)*, 2017, pp. 1–19.

[82] S. Salihoglu and J. Widom, “Optimizing graph algorithms on pregel-like systems,” in *the Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2014, pp. 577–588.

[83] L. Liu *et al.*, “An influence power-based clustering approach with PageRank-like model,” *Applied Soft Computing Journal*, vol. 40, pp. 17–32, 2016.

[84] C. C. Aggarwal and H. Wang, “Text Mining in Social Networks,” in *Social Network Data Analytics*, Springer US, 2011, pp. 353–378.

[85] K. Avrachenkov, S. K. Pham, and E. Smirnova, “PageRank Based Clustering of Hypertext Document Collections Categories and Subject Descriptors,” in *the Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, 2008, pp. 873–874.

[86] O. Shafiq, R. Alhajj, and J. G. Rokne, “On personalizing Web search using social network analysis,” *Information Sciences*, vol. 314, pp. 55–76, 2015.

[87] U. N. Raghavan, R. Albert, and S. Kumara, “Near linear time algorithm to detect community structures in large-scale networks,” *Physical Review E -Statistical,*

Nonlinear, and Soft Matter Physics, vol. 76, no. 3, pp. 1–12, 2007.

[88] J. Xie and B. K. Szymanski, "Community detection using a neighborhood strength driven Label Propagation Algorithm," in *the Proceedings of the IEEE 1st International Network Science Workshop*, 2011, pp. 188–197.

[89] J. M. Pujol and J. Delgado, "Clustering algorithm for determining community structure in large networks," *Physical review E*, vol. 74, no. 1, pp. 1–9, 2006.

[90] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," *World Wide Web Internet And Web Information Systems*, vol. 54, no. 1999–66, pp. 1–17, 1998.

[91] P. Wang, B. Ribeiro, J. Zhao, J. C. S. Lui, D. Towsley, and X. Guan, "Practical Characterization of Large Networks Using Neighborhood Information," *arXiv preprint arXiv:1311.3037*, 2013.

[92] X. Zhu and Z. Ghahramini, *Learning from Labeled and Unlabeled Data with Label Propagation*, Technical Report CMU-CALD-02-107, Carnegie Mellon Univ, 2002.

[93] F. Tian, B. Gao, Q. Cui, E. Chen, and T. Liu, "Learning Deep Representations for Graph Clustering," in *the Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 2015, pp. 1293–1299.

[94] J. Yang and J. Leskovec, "Overlapping community detection at scale: A Nonnegative Matrix Factorization Approach," in *the Proceedings of the 6th ACM international conference on Web search and data mining*, 2013, p. 587.

[95] S. Gregory, "Finding overlapping communities in networks by label propagation," *New Journal of Physics*, vol. 12, no. 10, pp. 1–21, 2010.

[96] T. Schäfer and P. Mutzel, "StruClus: Structural Clustering of Large-Scale Graph Databases," *arXiv preprint arXiv:1609.09000*, 2016.

[97] I. Timón, J. Soto, H. Pérez-sánchez, and J. M. Cecilia, "Parallel implementation of fuzzy minimals clustering algorithm R," *Expert Systems with Applications*, vol. 48, pp. 35–41, 2016.

[98] G. Paiva, E. Ogasawara, E. Bezerra, and G. Xexeo, "Discovering top- k non-redundant clusterings in attributed graphs," *Neurocomputing*, vol. 210, pp. 45–54, 2016.

[99] L. Liu *et al.*, "An Influence Power-Based Clustering Approach with PageRank-like Model," *Applied Soft Computing Journal*, vol. 40, pp. 17–32, 2016.

[100] S. A. Ludwig, "MapReduce-based Fuzzy C-means clustering algorithm : implementation and scalability," *International Journal of Machine Learning and Cybernetics*, vol. 6, no. 6, pp. 923–934, 2015.

[101] J. Yang and J. Leskovec, “Overlapping community detection at scale: A Nonnegative Matrix Factorization Approach,” in *the Proceedings of the 6th ACM international conference on Web search and data mining.*, 2013, p. 587.

[102] Z. Wu, G. Gao, Z. Bu, and J. Cao, “SIMPLE: a simplifying-ensembling framework for parallel community detection from large networks,” *Cluster Computing*, vol. 19, no. 1, pp. 211–221, 2016.

[103] X. Pan, D. Papailiopoulos, S. Oymak, B. Recht, K. Ramchandran, and M. I. Jordan, “Parallel Correlation Clustering on Big Graphs,” *arXiv:1507.05086v2*, pp. 1–22, 2015.

[104] P. T. Sivasankar and M. RamaKrishnan, “An Energy-Efficient Based on Heterogeneous Cluster Head Selection Algorithm for Wireless Sensor Networks,” *Journal of Computational and Theoretical Nanoscience*, vol. 14, no. 9, pp. 4520–4527, 2017.

[105] E. Estrada and J. a. Rodriguez-Velazquez, “Subgraph Centrality in Complex Networks,” *Physical Review E*, vol. 71, no. 5, pp. 1-5, 2005.

[106] S. P. Borgatti, K. M. Carley, and D. Krackhardt, “On the robustness of centrality measures under conditions of imperfect data,” *Social Networks*, vol. 28, no. 2, pp. 124–136, 2006.

[107] L. C. Freeman, “A Set of Measures of Centrality Based on Betweenness,” *Sociometry*, vol. 40, no. 1. pp. 35–41, 1977.

[108] A. Bavelas, “Communication Patterns in Task-Oriented Groups,” *The Journal of the Acoustical Society of America*, vol. 22, no. 6, pp. 725–730, 1950.

[109] L. Liu, L. Sun, S. Chen, M. Liu, and J. Zhong, “K -PRSCAN : A Clustering Method Based on PageRank,” *Neurocomputing*, vol. 175, pp. 65–80, 2015.

[110] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, “An Efficient k -Means Clustering Algorithm : Analysis and Implementation,” *IEEE Transactions on Pattern Analysis and Machine Learning*, vol. 24, no. 7, pp. 881–892, 2002.

[111] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, “Scalable K-Means ++,” in *the Proceedings of 38th International Conference on Very Large Data Bases*, 2012, pp. 622–633.

[112] Y. Xu *et al.*, “Efficient k -Means ++ Approximation with MapReduce,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3135–3144, 2014.

[113] W. Liu, X. Jiang, M. Pellegrini, and X. Wang, “Discovering communities in complex

networks by edge label propagation," *Scientific Reports*, vol. 6, no. 1, pp. 1–10, 2016.

[114] H. Avron, "Community Detection Using Time-Dependent Personalized PageRank," 2015.

[115] S. A. Tabrizi, A. Shakery, M. Asadpour, and M. Abbasi, "Personalized PageRank Clustering : A Graph clustering Algorithm based on Random Walks," *Physica A*, vol. 392, pp. 5772–5785, 2013.

[116] J. H. Chin and K. Ratnavelu, "Detecting community structure by using a constrained label propagation algorithm," *PLoS ONE*, vol. 11, no. 5, pp. 1–21, 2016.

[117] B. Wang, Z. Tu, and J. K. Tsotsos, "Dynamic label propagation for semi-supervised multi-class multi-label classification," in *the Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 425–432.

[118] D. Zhang, M. Ji, J. Yang, Y. Zhang, and F. Xie, "A Novel Cluster Validity Index for Fuzzy Clustering based on Bipartite Modularity," *Fuzzy Sets and Systems*, vol. 253, no. c, pp. 122–137, 2014.

[119] M. Shao, S. Li, Z. Ding, and Y. Fu, "Deep linear coding for fast graph clustering," in *the Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2015, pp. 3798–3804.

[120] P. Hu, K. C. C. Chan, and T. He, "Deep Graph Clustering in Social Network.," in *the Proceedings of the 26th International Conference on World Wide Web Companion.*, 2017, pp. 1425–1426.

[121] K. Zhang and X. W. Chen, "Large-scale deep belief nets with mapreduce," *IEEE Access*, vol. 2, pp. 395–403, 2014.

[122] S. Wikaisuksakul, "A multi-objective genetic algorithm with Fuzzy C-means for automatic data clustering," *Applied Soft Computing*, vol. 24, pp. 679–691, 2014.

[123] E. Egrioglu, C. Hakan, and U. Yolcu, "Fuzzy time series forecasting with a novel hybrid approach combining Fuzzy C-means and neural networks," *Expert Systems With Applications*, vol. 40, no. 3, pp. 854–857, 2013.

[124] M. A. Khalilia, J. Bezdek, M. Popescu, and J. M. Keller, "Improvements to the relational fuzzy c -means clustering algorithm," *Pattern Recognition*, vol. 47, pp. 3920–3930, 2014.

[125] C. Vehlow, S. Member, T. Reinhardt, D. Weiskopf, and I. C. Society, "Visualizing Fuzzy Overlapping Communities in Networks," *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, vol. 19, no. 12, pp. 2486–2495, 2013.

[126] X. Chen, “A new clustering algorithm based on near neighbor influence,” *Expert Systems with Applications*, vol. 42, pp. 7746–7758, 2015.

[127] R. Bhatnagar, V. Bhattacharjee, and M. . Ghose, “A proposed Novel Framework for early effort estimation using fuzzy logic techniques,” *Global Journal of Computer Science and Technology*, vol. 10, no. 14, pp. 66–72, 2010.

[128] J. Zhou, S. Member, C. L. P. Chen, and L. Chen, “A Collaborative Fuzzy Clustering Algorithm in Distributed Network Environments,” *IEEE Transactions on Fuzzy Systems*, vol. 22, no. 6, pp. 1443–1456, 2014.

[129] J. Xie, B. K. Szymanski, and X. Liu, “SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process,” in *the Proceedings of IEEE International Conference on Data Mining*, 2011, pp. 344–349.

[130] A. Stetco, X. Zeng, and J. Keane, “Fuzzy C-means ++ : Fuzzy C-means with effective seeding initialization,” *Expert Systems with Applications*, vol. 42, pp. 7541–7548, 2015.

[131] R. Pears, S. Pisalpanus, and Y. S. Koh, “A graph based approach to inferring item weights for pattern mining,” *Expert Systems with Applications*, vol. 42, no. 1, pp. 451–461, 2015.

[132] L. T. Thomas, S. R. Valluri, and K. Karlapalem, “MARGIN: Maximal Frequent Subgraph Mining,” *ACM Transactions on Knowledge Discovery from Data*, vol. 4, no. 3, pp. 1–42, 2010.

[133] D. Wu, J. Ren, and L. Sheng, “Uncertain maximal frequent subgraph mining algorithm based on adjacency matrix and weight,” *International Journal of Machine Learning and Cybernetics*, https://doi.org/10.1007/s13042-017-0655-y, pp. 1–11, 2017.

[134] M. Flores-Garrido, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, “Mining maximal frequent patterns in a single graph using inexact matching,” *Knowledge-Based Systems*, vol. 66, pp. 166–177, 2014.

[135] A. J. Moussaoui M., Zaghdoud M., “POSGRAMI: Possibilistic Frequent Subgraph Mining in a Single Large Graph,” in *the Proceedings of International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2016, pp. 549–561.

[136] Y. Jia, J. Zhang, and J. Huan, “An efficient graph-mining method for complicated and noisy data with real-world applications,” *Knowledge and Information Systems*, vol. 28, no. 2, pp. 423–447, 2011.

[137] N. Acosta-Mendoza, A. Gago-Alonso, and J. E. Medina-Pagola, “Frequent

approximate subgraphs as features for graph-based image classification," *Knowledge-Based Systems*, vol. 27, pp. 381–392, 2012.

[138] C. Chen, X. Yan, F. Zhu, and J. Han, "gApprox: Mining frequent approximate patterns from a massive network," *the Proceedings of IEEE International Conference on Data Mining*, pp. 445–450, 2007.

[139] M. Flores-Garrido, Carrasco-Ochoa, J. Ariel, and J. F. Martnez-Trinidad, "AGraP: an algorithm for mining frequent patterns in a single graph using inexact matching," *Knowledge and Information Systems*, vol. 44, no. 2, pp. 385–406, 2014.

[140] Y. Yuan, G. Wang, L. Chen, and B. Ning, "Efficient pattern matching on big uncertain graphs," *Information Sciences*, vol. 339, pp. 369–394, 2016.

[141] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in *the Proceedings of 20th International Conference on Very Large Data Bases, VLDB*, 1994, pp. 487–499.

[142] S. Nijssen and J. Kok, "Faster Association Rules for Multiple Relations," in *the Proceedings of International Joint Conference on Artificial Intelligence*, 2001, pp. 891–896.

[143] M. Kuramochi and G. Karypis, "An Efficient Algorithm for Discovering Frequent Subgraphs," *IEEE transactions on knowledge and data engineering*, vol. 16, no. 9, pp. 1038–1051, 2004.

[144] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *the Proceedings of IEEE International Conference on Data Mining*, 2001, pp. 313–320.

[145] M. Kuramochi and G. Karypis, "GREW - A scalable frequent subgraph discovery algorithm," in *the Proceedings of 4th IEEE International Conference on Data Mining*, 2004, pp. 439–442.

[146] L. Thomas, S. R. Valluri, and K. Karlapalem, "ISG: Itemset based Subgraph Mining," Technical Report, IIIT Hydrabad, India, 2009.

[147] M. Kuramochi and G. Karypis, "Discovering frequent geometric subgraphs," *Information Systems*, vol. 32, no. 8, pp. 1101–1120, 2007.

[148] T. Ramraj and R. Prabhakar, "Frequent subgraph mining algorithms - A survey," *Procedia Computer Science*, vol. 47, no. C, pp. 197–204, 2014.

[149] D. J. Cook and L. B. Holder, *Mining Graph Data*. Wiley, 2007.

[150] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *the Proceedings of 3rd IEEE International Conference on Data Mining*, 2003, pp. 2–5.

[151] S. Nijssen and J. N. Kok, “A Quickstart in Frequent Structure Mining Can Make a Difference,” in *the Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 647–652.

[152] X. Yan and J. Han, “CloseGraph: mining closed frequent graph patterns,” in *the Proceedings of 9th ACM SIGKDD International Conference on Knowledge discovery and data mining*, 2003, pp. 286–295.

[153] Y. Liu, J. Li, and H. Gao, “JPMiner: Mining frequent jump patterns from graph databases,” in *the Proceedings of 6th International Conference on Fuzzy Systems and Knowledge Discovery*, 2009, vol. 5, pp. 114–118.

[154] S. Reinhardt and G. Karypis, “A multi-level parallel implementation of a program for finding frequent patterns in a large sparse graph,” in *the Proceedings of 21st International Symposium on Parallel and Distributed Processing*, 2007, pp. 1-8.

[155] M. Kuramochi and G. Karypis, “Finding Frequent Patterns in a Large Sparse Graph,” *Journal of Data Mining and Knowledge Discovery*, vol. 11, no. 3, pp. 243–271, 2005.

[156] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and Kalnis. Panos, “GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph,” *Proceedings of VLDB Endowment*, vol. 7, no. 7, pp. 517--528, 2014.

[157] N. Babu, “A Distributed Approach to Weighted Frequent Subgraph Mining,” in *the Proceedings of IEEE International Conference on Emerging Technology Trends*, 2016, pp. 1–7.

[158] S. Aridhi, L. D’Orazio, M. Maddouri, and E. Mephu Nguifo, “Density-based data partitioning strategy to approximate large-scale subgraph mining,” *Information Systems*, vol. 48, pp. 213–223, 2015.

[159] E. Abdelhamid, I. Abdelaziz, P. Kalnis, Z. Khayyat, and F. Jamour, “ScaleMine: Scalable Parallel Frequent Subgraph Mining in a Single Large Graph,” in *the Proceedings of IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 1–12.

[160] C. H. C. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, “Arabesque: A System for Distributed Graph Mining,” in *the Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 425–440.

[161] W. Lu, G. Chen, A. K. H. Tung, and F. Zhao, “Efficiently extracting frequent subgraphs using MapReduce,” in *the Proceedings of IEEE International Conference on Big Data*, 2013, pp. 639–647.

[162] X. Zhao, Y. Chen, C. Xiao, and Y. Ishikawa, “Frequent Subgraph Mining Based on

Pregel," *The Computer Journal*, vol. 58, no. 8, pp. 1113-1128, 2016.

[163] B. Mozafari, P. Sarkar, M. Franklin, M. Jordan, S. Madden, and U. C. Berkeley, "Scaling Up Crowd-Sourcing to Very Large Datasets : A Case for Active Learning," *Proceedings of the VLDB Endowment* vol. 8, no. 2, pp. 125–136, 2015.

[164] L. Edwards, L. Johnson, M. Milosavljevic, V. Gadepally, and B. A. Miller, "Sampling Large Graphs for Anticipatory Analytics," in the proceedings of IEEE International Conference on *High Performance Extreme Computing,*2015, pp. 1-6.

[165] B. Ribeiro and D. Towsley, "Estimating and Sampling Graphs with Multidimensional Random Walks," in *the Proceedings of the 10th ACM SIGCOMM Conference on Internet measurement,*2010, pp. 390-403.

[166] E. Voudigari, N. Salamanos, T. Papageorgiou, and E. J. Yannakoudakis, "Rank Degree : An Efficient Algorithm for Graph Sampling," in *the Proceedings of IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2016, pp. 120–129.

[167] M. Lahiri and T. Y. Berger-Wolf, "Mining Periodic Behavior in Dynamic Social Networks," in the *Proceedings of 8th IEEE International Conference on Data Mining*, 2008, pp. 373–382.

[168] M. Berlingerio, F. Pinelli, and F. Calabrese, "ABACUS: Frequent pattern mining-based community discovery in multidimensional networks," *Data Mining and Knowledge Discovery*, vol. 27, no. 3, pp. 294–320, 2013.

[169] C. K. S. Leung and C. L. Carmichael, "Exploring social networks: A frequent pattern visualization approach," in *the Proceedings of 2nd IEEE International Conference on Social Computing, PASSAT 2010,* 2010, pp. 419–424.

[170] C. K. M. Lee, H. C. W. Lau, G. T. S. Ho, and W. Ho, "Design and development of agent-based procurement system to enhance business intelligence," *Expert Systems with Applications*, vol. 36, no. 1, pp. 877–884, 2009.

[171] M. Kuhn, C. von Mering, M. Campillos, L. J. Jensen, and P. Bork, "STITCH: Interaction networks of chemicals and proteins," *Nucleic Acids Research*, vol. 36, no. 1, pp. 684–688, 2008.

[172] M. S. Hossain and R. A. Angryk, "GDClust: A Graph-Based Document Clustering technique," in *the Proceedings of IEEE International Conference on Data Mining*, 2007, pp. 417–422.

[173] M. Al Hasan, "Mining interesting subgraphs by output space sampling," *ACM SIGKDD Explorations Newsletter*, vol. 12, no. 1, p. 73, 2010.

[174] S. Ranu and A. K. Singh, "GraphSig: a scalable approach to mining significant subgraphs in large graph databases," in *the Proceedings of International Conference on Data Engineering*, 2009, pp. 844–855.

[175] S. Zhang, S. Li, and J. Yang, "GADDI: Distance Index Based Subgraph Matching in Biological Networks," in *the Proceedings of 12th International Conference on Extending Database Technology*, 2009, pp. 192–203.

[176] S. . Tanev, G. . Liotta, and A. . Kleismantas, "A business intelligence approach using web search tools and online data reduction techniques to examine the value of product-enabled services," *Expert Systems with Applications*, vol. 42, pp. 7582–7600, 2015.

[177] E. Voudigari, J. Pavlopoulos, and M. Vazirgiannis, "A Framework for Web Page Rank Prediction", IFIP Advances in Information and Communication Technology, vol 364. Springer, Berlin, Heidelberg, pp. 1–10.

[178] J. Leskovec and K. Andrej, "Stanford Large Network Dataset Collection," 2014. [Online]. Available: https://snap.stanford.edu/data/. [Accessed: 10-Mar-2017].

[179] J. Kunegis, "The Koblenz Network Collection," 2016. [Online]. Available: http://konect.uni-koblenz.de/. [Accessed: 10-Jun-2017.

[180] R. Noldus and P. Van Mieghem, "Assortativity in Complex Networks," *Journal of Complex Networks*, vol. 3, no. 4, pp. 507–542, 2014.

[181] M. E. J. Newman, "Modularity and community structure in networks," *the Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[182] K. L. Wu, "Analysis of parameter selections for Fuzzy C-means," *Pattern Recognition*, vol. 45, no. 1, pp. 407–415, 2012.

[183] M. Fiedler and C. Borgelt, "Support computation for mining frequent subgraphs in a single graph," *the Proceedings of the 5th International Workshop on Mining and Learning with Graphs*, pp. 25–30, 2007.

[184] B. Bringmann and S. Nijssen, "What Is Frequent in a Single Graph?," in *the proceedings of International Conference on Advances in Knowledge Discovery and Data Mining*, 2008, pp. 1–4.

[185] C. Meeyoung, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring User Influence in Twitter: The Million Follower Fallacy," in *the Proceedings of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2010.

[186] Hall, B. H., A. B. Jaffe, and M. Trajtenberg, "The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools.," *NBER Working Paper 8498*, 2001. .

[187] M. Elseidy, E. Abdelhamid, and S. Skiadopoulos, “GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph,” *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 517--528, 2014.

LIST OF PUBLICATIONS

Journals

1) Vandana Bhatia and Rinkle Rani, "A Parallel Fuzzy Clustering algorithm for Large Graphs using Pregel", Expert Systems with Applications, Vol-78, pp. 135-144, 2017. [SCIE Indexed, Impact Factor-3.928] doi- 10.1016/j.eswa.2017.02.005.
2) Vandana Bhatia and Rinkle Rani, "DFuzzy: A Deep learning based Fuzzy Clustering Model for Large Graphs", Knowledge and Information Systems [SCIE Indexed, Impact Factor-2.004] doi- 10.1007/s10115-018-1156-3.
3) Vandana Bhatia and Rinkle Rani, "Ap-FSM: A Parallel algorithm for Approximate Frequent Subgraph Mining using Pregel", Expert Systems with Applications, Vol-106, pp. 217-232, 2018. [SCIE Indexed, Impact Factor-3.928]
   doi- 10.1016/j.eswa.2018.04.010.
4) Vandana Bhatia and Rinkle Rani, "PFCA: An Influence based Parallel Fuzzy Clustering algorithm for Large Complex Networks", Expert Systems-SCIE Indexed, Impact factor: 1.18. [In Press]
5) Vandana Bhatia and Rinkle Rani, "PaGro: A Distributed Pattern Growth based Frequent Subgraph Mining algorithm for Large Graphs", IEEE Transactions on Parallel and Distributed Computing-SCIE Indexed, Impact factor: 4.181. [Under Review]
6) Vandana Bhatia and Rinkle Rani, "PSGC: Parallel Structural Graph Clustering algorithm based on Subgraph Similarity", The Journal of Supercomputing -SCIE Indexed, Impact Factor-1.326 [Communicated].

Conferences

1) Vandana Bhatia and Rinkle Rani, “An Efficient Influence based Label Propagation algorithm for Clustering large graphs”, in the proceedings of IEEE International Conference on Infocom Technologies and Unmanned Systems (ICTUS'2017), pp. 1-7, December 2017.
2) Vandana Bhatia and Rinkle Rani, “An Efficient algorithm for Sampling of Single Large Graph”, in the proceedings of IEEE 10th International Conference on Contemporary Computing (IC3’2017), August 2017.
3) Vandana Bhatia and Rinkle Rani, “INGC: Graph Clustering & Outlier Detection algorithm using Label Propagation”, in the proceedings of IEEE International Conference on Machine Learning and Data Science, December 2017.