A Comparative Analysis of Parallel and Distributed FSM approaches on Large-scale Graph Data

S. Ajay Kumar, Pellakuri Vidyullatha

Abstract— Graph Mining has an explosive growth or increasing use of graphs in generating graph databases of real-world applications. It contains either a single large-scale graph or set of small size graphs. A Graph is used for complex dynamic modeling objects and they are often changing in nature i.e. it may increase or decrease in their size. Frequent subgraph mining plays a vital role in data mining, with an objective of extracting useful knowledge is presented in terms of repeated structures. Graph mining is used for the chemo-informatics, Bio-informatics, sentiment analysis, social human behavior analysis, financial network analysis, web analysis, wired and wireless networks. In this paper, we describe in detail survey on frequent subgraph mining algorithms, which are helpful for retrieving knowledge from complex dynamic objects. There are different types of large-scale parallel and distributed frameworks used for performing graph partitioning, FSM based on Apriori and pattern growth strategies, and large-scale graph processing techniques to achieve load balancing memory scalability, partitioning, load balancing, granularity and technical enhancement for future generations.

Keywords: Graph partitioning, frequent subgraph mining, Apriori, pattern growth, parallel framework.

1. INTRODUCTION

Nowadays, one of the most challenging issue to be faced by government, business, scientific, and social network communities is to mine valuable information [1]. Most of the domains, there have been a success for the data that takes in terms of entities and along with their attributes, because the relationship between the entities can be helpful. A Data Mining is an analysis step in the way of Knowledge Discovery in Databases (KDD) and it should be in different forms, i.e. recurring patterns of transactions to complex patterns of transactions which are interrelated. It aims to extract the attractive patterns and knowledge from the huge amount of homogenous or heterogeneous data which is being collected within the business and the experimental world.

Basics of Graphs: A Graph is an efficient and suitable method to model the data in complex databases. It serves as an imperative data structure in various areas including social network, bioinformatics and web applications etc. In this model, a relationship should be in terms of nodes and edges, where a node acts as an object and an edge will is to make the relationships between the nodes. Graph modeling which helps us to deal with the problems that were not solved before and find the similarity between graphs is essential.

As communication networks are graphs, in which each node is a protein and each edge describes the way of an interaction, there are two possible methods to measure the similarity between the graphs. First make a pair-wise comparison among the vertices and/or links in two networks and compute an overall similarity score for the given two networks from the similarity of among the components, it requires time quadratic in terms of its edges and vertices, and it is computationally possible even for large-scale graphs and it totally neglects and defects the network shape, considered them as the collection of nodes and edges instead of graphs. In this, if they share most of the common substructures or subgraphs then we can consider two networks are similar and to solve subgraph isomorphism problem is an NP-complete, in this, the time taken for execution is expensive because this problem increases exponentially with problem size. To speed up the subgraph isomorphism with the help of novel graph canonical labelling and different types of heuristics approaches were developed.

Modern graph datasets follow two magic words i.e. “Large and Evolving”, here term large means that vertices and edges available in that graph are in terms of millions and billions. Evolving in these graphs is not static frequently, changing with addition and/or deletion of nodes and/or edges [2]. In dynamic graphs, if an insertion of subgraph that leads to more number of subgraphs and it may not be the desired subgraph, becoming the targeted subgraph pattern, removal of a single node may result in the changes of most recently required subgraph available in the current graph into non-targeted subgraph or subgraph of the targeted subgraph. Next challenge is if a graph dataset is combined into a single entity or split into more units, i.e. it may be a single connected component or disconnected. These kinds of graphs will offer different performance with different algorithms before

Revised Version Manuscript Received on March 08, 2019.

S. Ajay Kumar, Research Scholar, Department of Computer Science & Engineering, KLEF, Vaddemswaram, Guntur Dist, Andhra Pradesh, India. and Associate Professor, Department of Computer Science & Engineering, Malla Reddy Engineering College(Autonomous), Hyderabad, Telangana, India.

Pellakuri Vidyullatha, Associate Professor, Department of Computer Science & Engineering, KLEF, Vaddemswaram, Guntur Dist, Andhra Pradesh, India.

selecting the proper algorithm and choosing data set type is also very essential. FSM is a critical and crucial problem in data retrieval or mining domain. The aim of the FSM is to identify possible subgraphs will occur in the input graphs more frequently than a given user-defined threshold [3]. It is also called frequent pattern mining or frequent subgraph discovery. It has significance in intra disciplinary areas, such as protein structure analysis, transportation systems, sentiment analysis, defense and security, wireless networks, financial network analysis, and sentiment analysis. The structures relevant information is gathered from the above applications are simple trees.

Apart from the graph analysis, FSM is essential in intra-disciplinary areas, such as privacy-preservation, protein function prediction, image processing, classification, clustering, and graph indexing. FSM in a social network is required to identify the way of social media communications. In spite of getting useful information, finding the FSM will probably provide, it may be sensitive data such as web-click graphs, trajectory graphs, mobile phone call graphs, and, release directly the FSM will cause substantial concerns on the users' privacy those who are participating in the data.

The remaining paper is described in the following strategy. In the next section, gives a review of different graph partitioning approaches. Section 3, deals with the various frequent subgraph mining techniques using Apriori-based and patterns growth methods those will be helpful for solving the FSM in small-scale graphs to large-scale graphs they depend on the candidate generation approaches and the frequency counting. Section 4, we present the parallel and distributed frameworks for finding the FSM on large-scale graphs. Section 5, discuss the conclusions and future scope.

2. GRAPH PARTITIONING APPROACHES:

Unstructured meshes are illustrated as undirected graphs and they follow the sparse matrix data structures. Various algorithms are operating on these graphs usually involve the iterative application of the same basic processing at each and every node of the graph, and each node will do a small amount of work. Unstructured mesh types of problems may be implemented on a parallel and distributed-memory computer, the graph must be divided into multiple subgraphs and each subgraph is associated with one processor. Then after, all of these subgraphs should be of very nearly equal size subgraphs or equal numbers of vertices for achieving an efficient load balancing, and the edges to be connected to the subgraphs is minimized for avoiding unnecessary communication or connection among the processors.

Let G is an undirected graph from G_1 to G_n. In this, $G_i=(V_i, E_i)$ is a tuple where V is a collection of non-empty vertices $V=\{v_1,v_2,\ldots,v_n\}$ and E is a collection of non-empty edges $E \subseteq V \times V$, such that each and every edge $e \in E$ belongs to the pair of vertices (v_1, v_2). Suppose, Graph $Gs = (V_s, E_s)$ is acts as a subgraph of $G =(V,E)$, indicated as $G_s \subseteq G$, iff $V_s \subseteq V$ and $E_s \subseteq E$. In this, bunch of graphs in G and user-mentioned support threshold is S_{min}, then G_s is said to be frequent only if it is a subgraph of at least S_{min} graphs. The support$(G, G_s) = \{G_i | G_s \subseteq G_i \cap G_i \in G\}$ denotes the supporting graphs of G_s. A subgraph G_s is frequent if $|support(G, G_s)| \geq Smin$. Suppose F denotes the collection of all FSMs and frequent subgraph $X \in F$ is said to be maximal only if it is not having the frequent supersets where M is a group of possible maximal FSMs and which is helpful to identify the maximal FSMs as connectivity is a valuable information which is collected from the connection between the nodes and permits for removing the mining disconnected combinations. The graph partitioning (GP) problem follows the NP-complete, it is also called the min-cut problem and defined as splitting the given graph into smaller chunks or blocks. Graph Partitioning can be implemented in two ways i.e. vertex based partitioning and edge-based partitioning.

For graph mining, good partitioning on large graphs is crucial because of several reasons. One of the best GP is the vital pre-processing step is to follow the divide-and-conquer algorithms, in this graph is divided into roughly same sized dense subgraphs, the same algorithm is applied on it and combines the intermediate results in the final phase. Another type of partition is to reduce the size of cross-partition edges that decreases the length of communication among the different machines at a large scale.

Large-scale graph partitioning challenges are multi-level paradigm, quality graph partitioning, and load balancing. There are various graph partitioning algorithms [4] that will use different techniques to overcome the above challenges.

There are two types of Graph partitioning techniques: 1) Static Partitioning 2) Dynamic Partitioning. Static Partitioning is further classified as serial partitioning and parallel partitioning whereas dynamic partitioning is further classified as distributed and clustered Approach. Figure1 shows the classification of algorithms depends on the implementation of graph partitioning strategies.

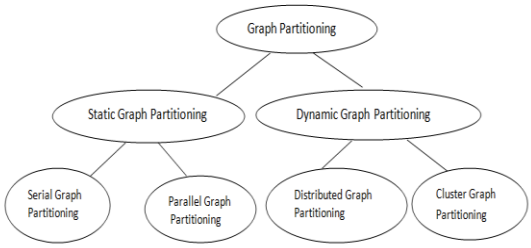

Figure1. Classification of Graph partitioning strategies

2.1. Non-Adaptive or Static Graph Partitioning:

Static partitioning is an essential way of multi level computations. Here, split a node into various m-disjoint subsets using the first stage of multi-phase calculations, in which they are active and the subgraph contains only those nodes in the first subset and its connected edges are partitioned. The subgraphs are calculated nodes are locked, it will not be changed further and accordingly, the next nodes which are available in the second subset will be appended to the subgraph. In this phase, the nodes which are from the second subset are easy to vary the subdomains and these free vertices which are allotted to the identical subdomains as their

locked neighbors. Though the next iteration of partitioning is over, the subdomains which are calculated from the second subset nodes are blocked, the third subset nodes and edges are added to the subgraph, it is further partitioned for the third iteration and the similar process is to be continued for m iterations, which gives a multi level partitioning result.

Serial Graph Partitioning: METIS is an irregular graph and which is consists of sequential programs for graph partitioning, divides the large meshes, and computes the fill-reducing orderings of sparse matrices will results in quickly produce good quality graph partitioning. METIS related algorithms are depended on the multilevel k-way, multilevel recursive-bisection, and multi-constraint partitioning approaches developed in the University of Minnesota Digital Technology Center (DTC) labs and is freely distributed.

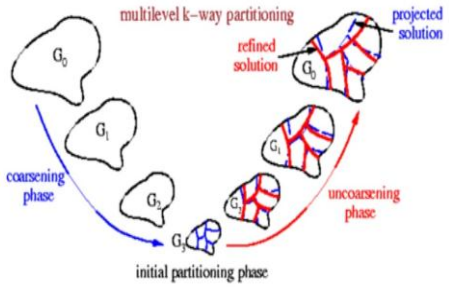

Figure2. Phases of Multilevel graph partitioning

Multilevel graph partitioning has three levels or phases that they are graph coarsening, initial partitioning, and uncoarsening. In the first phase i.e. Graph coarsening, a consecutive in sequence tiny graphs is computed from the input graph, every consecutive graph is to be built from the earlier graph by combining for a maximal size set of neighboring pairs of nodes and it should be continued until the size of the graph will have been reduced to less number of nodes. In the second phase, a partitioning of the coarsest, the smallest graph is to be identified, it is very small and which executes quickly. In the third phase, the partitioning of the smallest graph produces a useful larger graph by allocating the bunch of vertices which are combined to get the same partition as that of their relevant collapsed node. After completion of the projection step, it should be modified with the help of different heuristic methods to repetitively move the nodes between partitions till the that types moves will enhance the performance of the GP solution. Therefore, the uncoarsening phase is completed only if the partitioning solution given graph is obtained on has been projected all the way to the given graph.

METIS use modern strategies to consecutively decrease the size of the graph and refine the graph partitioning during the uncoarsening phase. At the time of coarsening, METIS applies the algorithms is simple to identify a good quality partition at the coarsest graph.

Parallel Graph Partitioning: There are various issues when the graph partitioning is performed in parallel implementation [5]. The large-scale graph data could not be computed in a consecutive manner because the memory is not enough to allow the partitioning that can be resolved on enormously concurrent implementation and a group of workstation clusters. To partitioning the very large-scale graph data in parallel graph partitioning algorithms for parallel implementation large size memory is needed.

Parallel graph partitioning is focused on geometric, spectral, and multilevel partitioning approaches. Geometric graph partitioning algorithms are easy to parallelize whereas multilevel partitions and spectral partitions are very complicated to make parallelize. The parallel asymptotic runtime is identical to which it performs a parallel matrix-vector multiplication on a randomly partitioned matrix, because of that the given graph is not properly scattered among the processors. Initially, the input graph is partitioned and distributed among the processors and the parallel asymptotic run times of multilevel partitions and spectral partitioning diminish the performance of a parallel matrix-vector multiplication on a good partitioned matrix and these partitioning approaches need to parallel well partitioning of the input graph.

2.2. Dynamic Graph Partitioning

Most of the static graph partitioning strategies is not give good quality of input graphs. But the dynamic graph partitioning will give the quality of input graph partitioning, including low edge cut that's the way the parallel adaptive graph partitions will run considerably faster than static partitions and it can be applicable to the scalable graphs and real-world applications.

Dynamic graph partitioning is not possible to do in a single memory because it has the aspect of radically increasing graph nodes and it may be billions of nodes, for this reason, it uses the distributed memory system concept which helps to place graphs in various machines and processing. For dynamic partitioning of a given large-scale graph, it requires load balancing, clustering and few heuristic techniques which obtain good results. Some of the other dynamic partitioning approaches are the parallel graph partitioning or streaming graph partitioning for the complex networks and spinner technique for scalable graph partitioning for cloud networks.

Most of the parallel geometric partitioning approaches will not affect the execution at the initial distribution of the graph, which will be helpful for the partitioning algorithm i.e. a rough partitioning is a faster geometric approach. Rough partitioning is useful for the reallocation of the graph before performing the spectral partitioning or parallel multilevel. This approach considerably enhances the concurrent efficts of the additional correct partitioning task.

Parallel graph partitioning [6] is decisive for achieving preeminent results in the circumstance of dynamic GP, in this, the input graph is shared earlier among the processors and it is the necessity of repartitioned as a consequence of adaptive nature of underlying computation. In a dynamic GP approach, if one processor is used for repartitioning of the given large-scale graph will generate a severe bottleneck problem that should be negative impact measurability. The widespread approach is to perform on the large graph that uses the

simultaneous computational systems to compute the algorithms and graph dataset is shared among the machines and different concurrent computation techniques will be useful to handle these types of graphs. There are various computationally efficient and good quality algorithms are used for graph partition problem, but the results are may or may not be optimal, such as spectral bisection method, Kernighan-Lin algorithm, and multilevel graph partitioning, however, these techniques are not scalable to the graph data with large sets. One of the eminent techniques is the graph stream partitioning for large scale distributed graphs is to accept the graph datasets sequentially from the disk onto a cluster heuristics techniques. In most situations, it is identified as impracticable for the distributed system, in which graph data is performed in a parallel way.

In distributed graphs, a distributed memory system was helpful for online query processing over large-scale graph which may be a billion nodes. For organizing the large-scale graph on distributed systems, segregate into more partitions and each and every partition is stored in one processor without any conflicts or overlapping. Dynamic multi-phase graph partitioning will cause major effects on load balancing and inter-process communication in distributed graphs. To maintain the consistency in this environment, fulfill the load balance constraints and minimize the inter-process communication.

3. FREQUENT SUBGRAPH MINING

FSM is used to calculate all the possible subgraphs which occur in the input graphs most frequently than a given user-defined threshold. It is generating the candidates and the task of candidate generation is done either in a depth-first or breadth-first strategy. From the generated candidate subgraphs, the frequently occurring subgraphs are found i.e. support counting. There are two issues of given FSM, first it is to generate the candidate subgraphs without including the duplicates and second is to calculate the eminent processing approaches that produce the necessary and desired frequent subgraphs or candidate subgraphs. FSM is used to check the isomorphism checking strategy. The best candidate subgraph generation technique will produce the candidates and it avoids the duplicate candidates.

Every FSM should follow the three characteristics that they are:

Graph Representations: A graph is described as an adjacency matrix or adjacency list and difficult to find the isomorphism testing using adjacency matrix and adjacency list. Canonical labelling describes a unique code for a given graph and it makes possible to check isomorphism because it makes sure that given pair of graphs is isomorphic, then their corresponding canonical labelling is same.

Subgraph Enumeration: Enumeration is a task of calculating or counting subgraphs one by one. Graph enumeration to be done by using either join operation or extension operation. With single join operation, multiple candidates are identified and with many join operations single candidate which can be duplicated. With extension operation, it can restrict nodes which include to the newly introduced edge.

Frequency Counting: Frequency counting for graphs can be done using either embedding lists or recomputed embedding.

3.1. Categorization of Frequent Subgraph Mining Algorithms: There are three major challenges of subgraph generation process that they are infrequent subgraphs, isomorphic subgraphs, and the subgraphs that do not available in the graph database. FSM algorithms use various strategies to reduce and eliminate these issues or challenges. Grouping the similar approaches depends on their nature of the given input graph data which may be small-scale graphs to large-scale graphs.

There are possible ways to find out the frequent subgraph mining search strategy i.e. either using Breadth First Search (BFS) or Depth First Search (DFS) approach. The algorithms basically depend on the input correctness then one is to accept the exact graph for a certain group of graphs as input and another is an uncertain set of graphs as input. In this, an input may be a bunch of small graphs or a single large graph as input. After multiple FSGs are identified then it will produce a complete set of FSGs or a partial set of FSGs.

Apriori-based Approach: The basic scheme of this frequent substructure mining algorithms [15] is to combine two size-k frequent sub structures and to produce (k+1) size candidate graphs, searching for frequent graphs and starts with the small size and continues in a bottom-up approach. In this, isomorphism testing is an expensive i.e. NP-Complete problem, test the false candidates or false search efficiency is minimized.

Apriori-based FSM approaches are further classified as follows: Suppose given graph is a large single graph then GREW and HSIGRAM algorithms are used. If the type of the graph input is a graph database then it uses the AGM algorithm. Suppose input is a set of graphs then FARMER, FSG, FFSG, ISG, and SPIN algorithms are used. If the input is a dynamic graph then it uses Dynamic GREW. Finally, the given input graph is an uncertain set of graphs then we use MUSE, MUSE-P, WMUSE, and UGRAP algorithms.

GREW [7]: It is a sparse matrix representation of a static graph and uses repetitive merging for subgraph candidate generation. It follows a greedy search strategy. The aim of the GREW is to calculate maximal independent set of a graph, which is collected from the FSM so as to calculate the frequency which missed many interesting patterns. GREW is efficient for large-scale graphs and for finding non-trivial patterns which allow the maximum portion of the input graph.

HSIGRAM : It is an adjacency matrix representation of a static graph and uses the iterative merging for subgraph candidate generation which follows a breadth-first search strategy. The aim of the HSIGRAM is to calculate the maximal independent group of a graph, which is built from the FSM embeddings to evaluate its frequency.

AGM [8]: It uses adjacency matrix representation of a static graph and uses the vertex extension for subgraph candidate generation which follows a breadth-first search strategy. It is used to find the FSMs using graph database as input, which is gathered from the frequent subgraph so as to

assess the Canonical labelling. It increases the substructure size iteratively one vertex at each step.

FARMER: It uses trie structure representation of a static graph and uses the level-wise search (ILP) for subgraph candidate generation which follows a breadth-first search strategy. FARMER is to find the FSM using a group of input graphs and it not efficient for most of the output generations.

FSG: It uses adjacency-list way of a static graph and uses the one edge extension for subgraph candidate generation which follows a breadth-first search strategy. FSG aim is to identify the frequently connected subgraphs using the bunch of graphs as an input, which is to be the frequent subgraph and estimate the Transaction identifier (TID) lists and it follows NP-Complete approach.

FFSG [9]: It describes the matrix adjacency representation of a static graph and uses the merging and extension for subgraph candidate generation which follows a DFS strategy. FFSG is needed to calculate the FSMs using graphs as input, which is cumulated from FSM so as to find the sub-optimal canonical adjacency matrix tree and it follows NP-Complete approach.

ISG: It uses Edge triplet representation of a static graph and uses the edge triplet extension for subgraph candidate generation which follows a breadth-first search strategy. ISG is to compute the maximal FSM using the input of a set of graphs and it is build from the FSM for evaluating the Transaction identifier (TID) lists and it follows the NP-complete approach, each and every FSM supports group of transaction identifiers.

SPIN [10]: It is the adjacency matrix representation of a static graph and uses the join operation for subgraph candidate generation which follows a breadth-first search strategy. SPIN is used to find the maximal FSM using a set of graphs as input, to evaluate the Canonical spanning tree and non-maximal graphs are to be found but needs an entire database scan. It works efficiently for compressing a huge number of FSM to a very small number of maximal subgraphs and allows large scalability to large graph databases.

Dynamic GREW: It is a sparse matrix representation of a dynamic graph and uses the iterative merging for the subgraph candidate generation which follows the depth-first search strategy. In this, FSM is constructed from dynamic patterns in FSM to collect the suffix trees and it needs more burden to identify dynamic patterns.

MUSE [11]: (Mining Uncertain Subgraph patterns). It follows the adjacency matrix representation of a static graph and uses the Disjunctive normal forms for subgraph candidate generation which follows a DFS strategy. The main aim is to find the FSMs use the input as an uncertain set of graphs, which is constructed from frequent subgraphs for evaluation of the DFS coding scheme. It found that the given sub-graph pattern is an output or not. Various pruning techniques are helpful to decreases the complexity of verifying subgraph patterns. MUSE is an accurate, efficient algorithm for massive uncertain graph databases. MUSE algorithm is further enhanced by MUSE-P and WMUSE (Weighted MUSE) algorithms as per application requirements.

Pattern Growth Approach: The basic idea of a pattern Growth FSM algorithm is to extend the frequent graph keep on including the new edge, in every feasible position and it uses edge-extension technique, the problem with this approach is that relevant sub-graph is identified the number of times. To overcome the problems of Apriori-based algorithms, many pattern growth strategies use the depth-first search technique.

Pattern growth FSM approaches are further classified as follows: Suppose the input graph is a single large graph then SUBDUE, GBI and SEUS algorithms are used. Some situations, the input graph is a set of graphs then we use CloseGraph, Gaston, gSpan, MOFA, MSPAN, JPMiner, TSP, RP-GD, and RP-FP algorithms.

SUBDUE: It uses the adjacency matrix representation and uses the level-wise search for subgraph candidate generation which follows a greedy search strategy. SUBDUE is to gather the minimum description code length and it uses very less number of patterns.

GBI : It uses the single large static graph as an input and which follows a greedy search strategy. The isomorphic test is exact for GBI.

SEUS : It uses the single large static graph as an input and which follows a depth-first search strategy. The isomorphic test is exact for SEUS.

CloseGraph: It basically follows the adjacency list representation of a static graph and uses the rightmost extension for subgraph candidate generation which follows a depth-first search strategy. The CloseGraph aim is to compute the depth-first search lexicographic order and it required a lot of time for failure detection.

Gaston: It uses hash table representation of a static graph and uses the extension for subgraph candidate generation which follows a DFS strategy. Gaston is used to find the embedded lists and interesting patterns may be lost.

GSpan: It uses the adjacency list representation of a static graph and uses the rightmost extension for subgraph candidate generation which follows the depth-first search strategy. GSpan performs the depth- first search lexicographic order and frequent graphs generated can't be scalable.

MOFA: It uses the adjacency list representation of a static graph and uses the rightmost extension for subgraph candidate generation. It is to identify the depth-first search lexicographic order and frequent graphs produced may or may not be exactly frequent.

MSPAN: It uses adjacency list representation of a static graph and uses the rightmost extension for subgraph candidate generation which follows depth-first search strategy. MSPAN follows the depth-first search lexicographic order.

JPMiner : It follows the adjacency list representation of a static graph and uses the rightmost extension for subgraph candidate generation. It is to examine the DFS lexicographic order, to evaluate the frequent jump patterns and sometimes there may be less number of jump patterns.

TSP : It is the adjacency list representation of a dynamic graph and uses the rightmost extension for subgraph

candidate. The aim of the TSP is to find the depth-first search lexicographic order, to find the representative graphs and time for summarizing the pattern is more than for mining.

RP-GD: It uses the adjacency list representation of a static graph and uses the rightmost extension for subgraph candidate generation. The aim of the RP-GD is to find the depth-first search lexicographic order, and frequent graphs generated can't be scalable.

RP-FP: It uses the rightmost extension for subgraph candidate generation which follows a depth-first search strategy. It is used to find the depth-first search lexicographic order, and time for summarizing the pattern is more than for mining.

4. PARALLEL AND DISTRIBUTED FRAMEWORK FOR GENERATION OF LARGE SCALE FSM:

We describe a framework for processing and generating large scale frequent subgraph mining with parallel and distributed approaches on a cluster. Scalable graph processing consists of five phases that they are reading the data, pre-processing, partitioning, computation and writing data, in each phase handle the errors effectively. Graph processing architecture will be one of the graph programming models such as Distributed Architecture, Share-Memory Architecture, and Heterogeneous Architecture. For the implementation of this framework, we need to use the following aspects, they are MapReduce, Hadoop, GraphLab, Giraph, and GraphX techniques.

MapReduce Framework: MapReduce [12] is a programming framework used for the processing and generating a huge volume of data in a distributed manner. It provides an efficient user interface for handling the data distribution, replication, and load balancing. It performs three abstract functions i.e. map, shuffle, and reduce. The raw data is used for reading and processing to the output (key, value) pairs is done by map function. The output of the map function is forwarded to the reducers propagated through the more secure network so that the pair with the identical key is to be grouped together done by the shuffle function. The key-value pair with the identical key is to be processed as output, key and value pair is done by reduce function. In this MapReduce iterative program which runs on several MapReduce tasks, then the output of the current task should be the input of the next task.

Hadoop: Hadoop provides a scalable idea to store and process the data sets with a huge amount in distributed and parallel fashion. It is an open source implementation of the MapReduce programming model and for its file management which follows the Hadoop Distributed File System (HDFS). Various tools which are used on top of Hadoop such as Hive, PIG, Spark, HBASE, iMapReduce [13], Scoop and Flume. Because of the accuracy, big graph mining, efficiency, scalability simplicity, and fault tolerance using Hadoop.

GraphX: It is a data model which contains the property graphs and immutable collections. It was developed on top of the Apache Spark [14], represents transformation on graphs and each and every operating action creates a new graph. It is apache spark API for graphs and graph parallel computations which intends to execute the large-scale graph algorithms proficiently. The immutability condition will clarify the abstraction and helpful for fault tolerance and reuse of data. It consists of unordered tuples or key-value pairs which are represented as unstructured data. In this, a key may be null and not be unique, and value might be a random object. The unordered data collection view is important for processing the raw input, graph computation results are to be evaluated. This framework uses a programming abstraction method is said to be Resilient Distributed Graph (RDG) interface, it constructed on Spark's in-memory storage abstraction i.e. Resilient Distributed Datasets (RDD). It follows the directed adjacency structure with user specified attributes connected to each and every vertex and edge, and they are treated as RDGs.

Giraph: Apache Giraph [15] is the most popular and powerful open-source platform for the parallel and distributed environment. It uses Map-only Hadoop jobs to coordinates the vertex-centric workers and it takes help of HDFS for storing and processing graph datasets. It has a faster input loading time compared to Pregel and it inherits the benefits and deficiencies of the Pregel vertex-centric programming method.

GraphX, Giraph, and GraphLab parallel frameworks provide good results when compare to conventional frameworks, thus one of the above-mentioned frameworks can be adopted for identifying FSM in a large scale graph data and implemented further using either R language or Python.

5. EXPERIMENTAL RESULTS:

Time complexity analysis of the PFSG-Miner is to divide the analysis of the FStG algorithm is on master processor and PFSG algorithm is on the slaver processors in parallel environment. All the experiments have been conducted on a Downing 4000L platform at Gansu Computing Center. The platform contain 48 2.2GHz CPUs and 24 2.2GHz Core 2 CPUs. It has 14 146G Ultra30 SCSI hard disks. In order to reflect the effectiveness of the algorithm, compare this algorithm with the gSpan and FFSM algorithms. The

gSpan and FFSM execute on single processor, while this algorithm will run on four slave processors at parallel phase.

The experiment chooses real data and simulation data. The real data is DTP (therapeutic program developmental) which provides the collection of compounds. The compounds in DTP can be divided into 3 datasets, which include CA (active confirmed), CM (moderately confirmed) and CI (confirmed Inactive). Figure 3, Figure 4 and Figure 5 shows the running results of PFSG-Miner, FFSM, and gSpan on different datasets.

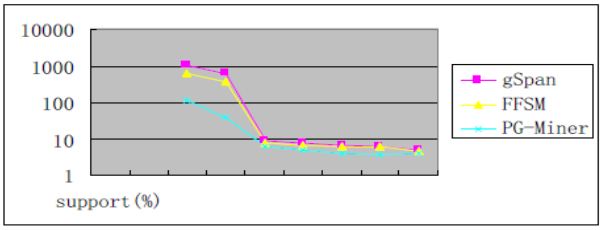

Figure 3: CA Data Set

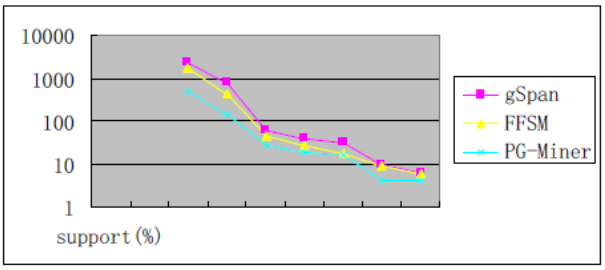

Figure 4: CM Data Set

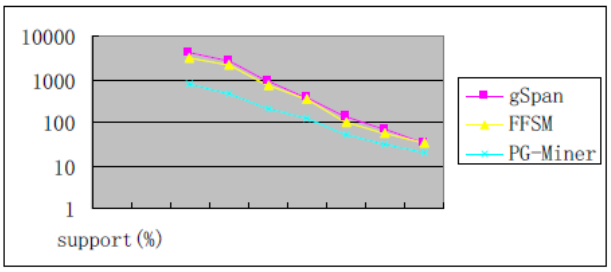

Figure 5: CM Data Set

5. CONCLUSION

In this paper, we present a comparative analysis of the several FSM algorithms with their merits, demerits, behavior, and application in a parallel and distributed environment. FSM is useful for multi disciplinary areas like bioinformatics, chemo-informatics etc. where digging of iterative patterns among the huge collection of networks. Based on the various graph partitioning algorithms, we have classified graph partitioning approaches into static, dynamic, distributed and parallel implementations. The algorithms have been classified depends on the Apriori and Pattern growth approaches. Generally, the existing algorithms focus only on static graph datasets. Still, most of the research work was not done on the complex dynamic or evolving large-scale graphs for mining pattern. If it applies on distributed and parallel implementation will also save the more processing time.

6. REFERENCES

1. Susanna Thomas and Jyothisha. J. Nair, “A Survey on extracting Frequent Subgraphs”, Conference on Advances in Computing, Communications and Informatics (ICACCI), Sept. 21-24, 2016, Jaipur, India.
2. Chandra Prakash Dixit and Nilay Khare, “A survey of frequent subgraph mining algorithms”, International journal of Engineering and Technology, Issue 7, page No: 58-62, 2018.
3. K.lakshmi and Dr. T Meyyappan, “A Comparative study of frequent subgraph mining algorithms”, International Journal of Information Technology Convergence and Services (IJITCS), Vol.2, No.2, April 2012.
4. Appala Srinuvasu Muttipati and Dr. Poosapati Padmaja, “Analysis of Large Graph Partitioning and Frequent Subgraph Mining on Graph Data”, International Journal of Advanced Research in Computer Science, Volume 6, No. 7, September-October 2015.
5. A. Grama, G. Karypis and V. Kumar, “Introduction to Parallel Computing”, Addison-Wesley, 2nd edition, 2003.
6. K. Schloegel, G. Karypis and V. Kumar, “Graph partitioning for high-performance scientific simulations,” In: Sourcebook of parallel computing, Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003, pp. 491-541.
7. J. Huan, W. Wang, J. Prins and J. Yang, “Spin: Mining maximal frequent subgraphs from graph databases,” UNC Technical Report TR04-018, 2004.
A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," Proce. European Conference on Principles of Data Mining and Kn owledge Discovery (PKDD 00), 2000, pp. 13-23.
8. M. Kuramochi.M and G. Karypis, “Finding Frequent Patterns In a Large Sparse Graph”, in Proceedings of the4th SIAM International Conference on Data Mining (SDM 2004), USA, 2004.
9. B. Wackersreuther, P. Wackersreuther, A. Oswald, C. Bohm and K. M. Borgwardt, “Frequent subgraph discovery in dynamic networks,” Proce. Eighth Workshop on mining and Learning with Graphs (MLG 10), 2010, pp. 155-162, doi:10.1145/1830252.1830272.
10. S. Jamil, A. Khan, Z Halim, A. R. Baig, “Weighted MUSE for Frequent Subgraph Pattern Finding in Uncertain DBPL Data,” Proce. International Conference on Internet Technology and applications (iTAP), 16-18 Aug. 2011, pp. 1-6.
11. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Proce. Symposium on Operating System Design and Implementation, pp. 137–150, 2004.
12. Y. Zhang, Q. Gao, L. Gao, and C. Wang, "iMapreduce: A distributed computing framework for iterative computation," DataCloud, 2011.
13. Daniel Crankshaw, Ankur Dave and Reynold S. Xin, “The GraphX Graph Processing System”, UC Berkeley AMPLab.
14. Avery Ching, Sergey Edunov and Maja Kabiljo, “One Trillion Edges: Graph Processing at Facebook-Scale” Proceedings of the VLDB Endowment, Volume 8.