ISSN No. 0976-5697

Volume 6, No. 7, September-October 2015

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

Analysis of Large Graph Partitioning and Frequent Subgraph Mining on Graph Data

Appala Srinuvasu Muttipati
Research Scholar: Computer Science and Engineering
GITAM University
Andhra Pradesh, India

Dr. Poosapati Padmaja
Associate Professor: Information Technology
GITAM University
Andhra Pradesh, India

Abstract: Graph mining has attracted much attention due to explosive growth in generating graph databases. The graph database is one type of database that consists of either a single large graph or a number of relatively small graphs. Some applications that produce graph database are biological networks, semantic web and behavioural modelling. Frequent subgraph mining is playing an essential role in data mining, with an objective of extracting knowledge in the form of repeated structures. Many efficient subgraph mining algorithms have been discovered in the last two decades, yet most do not scale to the type of data, the so-called "Large-Scale Graph Data". Many problems are so large or complex that it is impractical or impossible to solve them on a single computer, especially with given limited memory. Scalable parallel computing algorithms holds the key role for solving the problem in this context. Various algorithms and parallel frameworks have been discussed for graph partitioning, frequent subgraph mining based on apriori and pattern growth approaches, and large-scale graph processing techniques. The central objective of this paper is to initiate research and development of identifying frequent subgraph mining and strategies for graph data centres in such a way that brings it parallel frameworks for achieving memory scalability, partitioning, load balancing, granularity, and technical enhancement for future generations.

Keywords: graph partitioning; frequent subgraph mining; apriori; pattern growth; parallel framework.

I. INTRODUCTION

Graph-based representation of real world problem has been obliging large due to their improving simplicity and professional use in finding solutions. A graph has a newest domain in the field of Knowledge discovery and data mining (i.e. graph mining). Graph mining [1] is one of the most investigated and growing research topics in data mining domain. The applications of data mining include Bioinformatics pattern mining [2], Network link analysis [3], Financial analysis data [4], Chemical data analysis [5,6], Drug detection, Biological network [7], Protein structure interaction [8,9] and Social networking [10]. Long-established data mining techniques such as pattern matching, classification, clustering and frequent subgraph discovery has been extended to graph scenario.

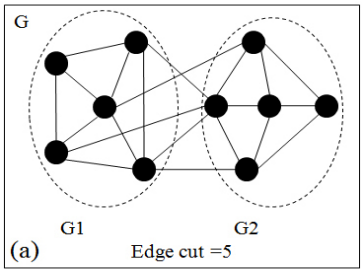
Nowadays, large graphs have increased a lot, migrating from Gigabytes to Terabytes and even up to Petabytes of data are being generated every day which gets processed online through social networking sites, biological networks. This type of data can be represented as a modeled graph, where the nodes represent user and edges represent the relation between them. Likewise, search engines manage huge amounts of data by capturing from the internet. Most common example to represent modeled graph is by considering websites as nodes and URLs as edges.

The main motive of this paper is to partition a large graph and identify frequent subgraph by mining techniques. Many efficient graph partitioning [11] and frequent subgraph mining algorithms [12] are existed that find frequent subgraph/ patterns from single graph data or multiple sets of graph data. Some of today's frequent subgraph mining source data may not fit on a single machine's hard drive. The exponential nature of the solution space compounds this problem. Scalable parallel algorithms hold a key role in addressing frequent subgraph in the context of large-scale graph data.

The rest of the paper is organized as follows. Section II presented a review on various graph partitioning methods and related software tools. Section III, describes various existed frequent subgraph mining algorithms based on apriori-based and patterns growth approach which assist to find the frequent subgraph in single graph data or multiple set of graph data. Section IV, deal with the graph processing framework which helps to process large-scale graphs. Section V, authors discuss the parallel frameworks for finding the frequent subgraph from Large-scale graphs. After discussing future research directions, we concluded in section VI.

II. GRAPH PARTITIONING

A graph G is a pair (N, E). Let N be a non-empty set of vertices and E be a nonempty set of edges $E \subseteq N \times N$ such that every edge $e \in E$ relates to the pair of vertices (N_1, N_2). The graph partitioning (GP) problem is NP-complete [13]. GP can be referred to as min-cut problem; it is defined as dividing a graph into a smaller blocks or pieces. It can be done in two ways edge based partitioning and vertex based partitioning, Figure 1 shows an example of graph partition.



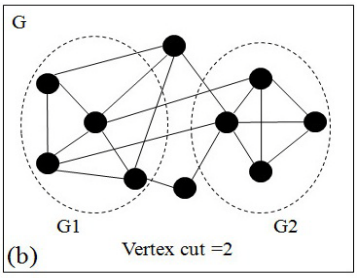


Figure 1. a) graph partitioning with edge cut b) graph partitioning with vertex cut

Consequently, in general, it is not sufficient to compute optimal partitioning for graphs of interesting size in a realistic amount of time. This statement combines the significance of the problem, which lead to the development of numerous heuristic approaches. The partitioning heuristics are divided into global and local search methods. Global methods are sometimes called as construction heuristics for the reason that they take graph description as input and generate a balanced partition**.** *Local search methods* are called improvement heuristics. They take a graph and balanced partition as input and try to improve the partition. Each partitioning algorithm has to include a *global search method*. The Optimal method produces the optimal results, although have exponential run time behaviour. The heuristic methods are Linear, Scattered and Random are exclusively depending on the node position in the given node list [14].

A. Static Graph Partitioning

Hendrickson and Leland enthused by the work of Stephen and Horst [15]. A multilevel approach to computing an eigenvector needed for a spectral partitioning algorithm. This work analyzes numerical problems in transferring eigenvectors between levels. The above problem is solved by transferring partitions between levels using multilevel graph partitioning algorithm by Hendrickson and Leland [16]. The algorithm contains three divisions: First division, a sequence of graphs is constructed by collapsing together selected vertices of the input graph in order to form a related coarser graph. This newly constructed graph then proceeds as the input graph for another round of graph coarsening and so on until an adequately small graph is obtained. Second division; computation of the spectral method was performed on the coarsest of these graphs are very fast. Third division, to project the coarse partition back through the sequence of graphs, periodically improving it with a local refinement algorithm for this local improvement phase authors use a variant of a popular algorithm originally derived by Kernighan and Lin [17], additionally it was improved by Fiduccia and Mattheyses [18], though other methods could be used very well. Figure 4 gives a detail construction of multilevel graph partitioning phase and Table I provides different methods for contraction, initial partitioning, and refinement.

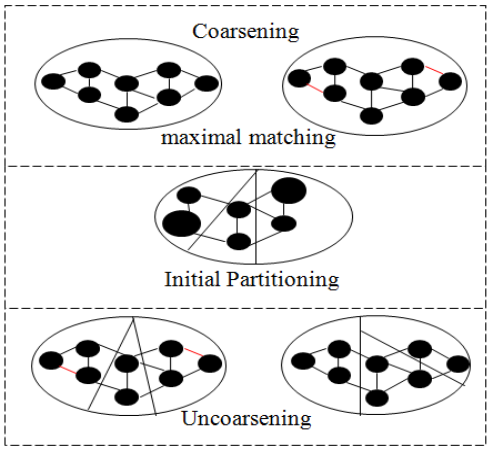


Figure 2. Multilevel Graph Partitioning

The two essential parts of multilevel approach [19-21] are coarsening strategy and local improvement method. In maintaining coarsening approach, the following aspects are required.

- The matching algorithm has to be very fast so that it is more time efficient than standard partitioning methods applied to the initial graph.
- Since edges of high weights are usually connecting dense areas of graph, the algorithm is supposed to calculate a matching with a high edge weight in order to avoid them from appearing in the coarse graph and being cut.
- To speed up the coarsening process and to coarsen on the whole graph simultaneously, the matching's on each level should have a high cardinality. The maximum reduction is achieved by splitting number of vertices in halves on each level. This is only possible when a complete matching can be found.

Table I: different method for multilevel phases

coarsening	***partitioning***	***Uncoarsening***
Random Matching (RM), Heavy Edge Matching (HEM), Modified Heavy Edge Matching (MHEM) Light Edge Matching (LEM), Heavy Clique Matching (HCM).	Coordinate sorting, Geometric partitioning, Spectral Bisection (SB), Recursive Bisection (RB), Coordinate Nested Dissection (CND), Graph Growing Partition (GGP), Greedy Graph Growing Partitioning (GGGP).	Greedy Refinement (GR), Kernighan Lin Refinement (KLR), Boundary Greedy Refinement (BGR), Boundary Kernighan Lin Refinement (BKLR), Combination of BGR and BKLR, Hill Climbing, Helpful-Set, Simulated annealing, Tabu search, Reactive search optimization.

B. Parallel Graph Partitioning

The graph partitioning is performed in parallel implementation [22] is necessary for various reasons.

- The large graphs can't be computed in serial implementation because of memory, which is often not enough to allow the partitioning that can be now solved on massively parallel implementation and workstation clusters.

- A parallel graph partitioning algorithms can take an advantage of the extensively higher amount of memory available in parallel implementation to partition very large graphs.

Parallel graph partitioning [23] is crucial for achieving potential results in such an environment, within the context of adaptive graph partitioning, where graph is already distributed among processors, and however it must be repartitioned as a result of dynamic nature of underlying computation. In such cases, getting the graph into a single processor for repartitioning will produce a serious bottleneck that would adversely impact the measurability of the general application.

Further work on parallel graph partitioning was concentrated on geometric, spectral by Stephen and Simon [24], and multilevel partitioning schemes by Karypis and Kumar [25, 26]. Geometric graph partitioning algorithms [27, 28] tend to be slightly easy to parallelize whereas spectral and multilevel partitioners are complex to parallelize. The parallel asymptotic run times are equivalent as that of performing a parallel matrix-vector multiplication on a randomly partitioned matrix. Because of this reason the input graph is not well-distributed across the processors. If the graph is first partitioned and then distributed across the processors consequently, the parallel asymptotic run times of spectral and multilevel partitioners drop to that of performing a parallel matrix-vector multiplication on a well-partitioned matrix. Primarily, performing these partitioning schemes professionally in parallel needs a good partitioning of the input graph. The static graph partitioning cannot provide a good quality of input graph whereas the adaptive graph partitioning can provide a high quality of input graph partitioning which includes a low edge cut. For this reason, parallel adaptive graph partitioners [29, 30] attend to run considerably faster than static partitioners.

Since the run time of most parallel geometric partitioning schemes does not seem to affect the initial distribution of the graph, they will primarily be accustomed to working out a partitioning for the partitioning algorithm. That is a rough partitioning of the input graph which is often computed by a faster geometric approach. This partitioning can be used to reallocate graph before performing parallel multilevel or spectral partitioning. Use of this "boot-strapping" approach will significantly increase the parallel efficiency of the additional correct partitioning scheme by providing it with data region [23].

C. *Dynamic Graph Partitioning*

Currently, huge research is carrying on dynamic graph partitioning due to the real world applications and scalable graphs. Dynamic graph partitioning cannot do in single memory because it has the feature of dramatically increasing of node/vertices in a graph (billion nodes). For that reason it holds the concept of distributed memory system, helps to place graphs in various machines and processing. For partitioning a dynamic graph, it needs clustering, load balancing and some local heuristic methods which obtain good results. Few authors presented an articles are How to Partition a Billion-node graph presented by [31]. Streaming graph partitioning method was by [32], parallel graph partitioning for complex Networks by [33]. Spinner technique for Scalable graph partitioning for the cloud by [34]. The Figure 3 describes distributed memory system and label propagation.

- *Distributed graphs* A distributed memory system is a right communication for online query processing over a billion node graph. To organize a graph on a distributed memory system, have to divide the graph into multiple partitions and store each partition in one machine (without any overlapping). Network communication is necessary for accessing non-local partitions of the graph. Hence, how the graph is partitioned might cause the major impact on load balancing and communication [31].
- *Label Propagation (LP)* A local inhabitant of LP as follows. First assign a unique label id to each vertex. Next, update the vertex label iteratively. At every, iteration a vertex takes the label that is ubiquitous in its neighborhood as its own label. The process terminates when labels no longer change. Vertices that have the identical label belong to the identical partition.

There are two reasons to assume label propagation for partitioning [31].

1) Label propagation mechanism is lightweight. It does not cause intermediary results, and it does not need sorting or indexing the data as in many existing graph partitioning algorithms.
2) Label propagation is able to discover inherent community structures in real networks: Given the reality of local closely connected substructures, a label tends to propagate within such structures. Since most real-life networks exhibit clear community structures, a partitioning algorithm based on label propagation may divide the graph into consequential partitions. Compared to maximal match, LP is more semantic-aware and is a better coarsening scheme.

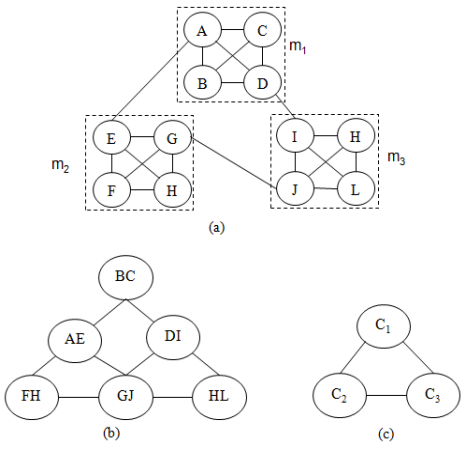


Figure 3. (a) A graph with 3 machines {m_1, m_2, m_3}, each machine carries 4 vertices and the partition graphs as {A, B, C, D}, {E, F, G, H}, {I, J, K, L}. (b) Coarsened by maximal match. (c) Coarsened by LP

D. *Classifiation of graph partitioning algorithms*

The main challenges of partitioning a graph data are: (i) quality graph partitioning (ii) multilevel paradigm (ii) load balancing. Graph partitioning algorithms use different approaches to eliminate these challenges. Figure 4 shows different algorithms are classified based on the implementation of graph partitioning approaches.

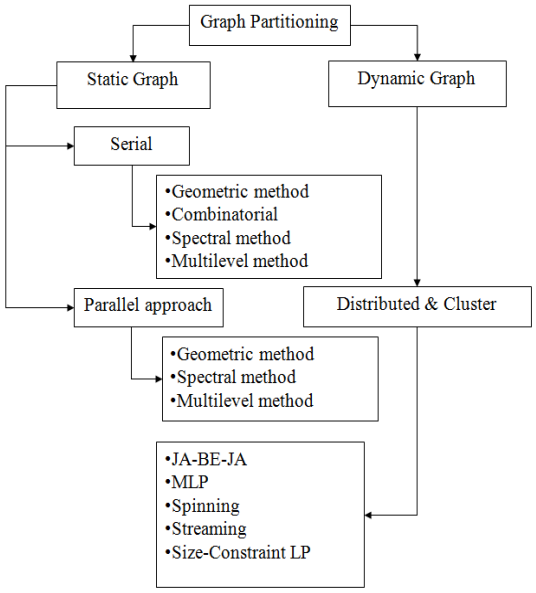


Figure 4. Classification of Graph Partitioning Methods

E. Software tools

Hendrickson and Leland [35] are the first to launch graph partitioning tool Chaco and to implement the multilevel graph partitioning. By adopting this multilevel Karypis and Kumar [36-39] has proposed parallel multilevel k-way partitioning scheme, fast and high quality multilevel schemes, parallel algorithm for multilevel graph partitioning, MeTiS, hMeTiS tool which offer good partitions. Karypis, Schloegel and Kumar [40] were developed ParMeTiS tool for parallel graph partitioning. Recently Sander and Schulz [41] was developed KaHIP tool in which Karlsruhe Fast Flow Partitioning (KaFFPa) is a multilevel graph partitioning algorithm by Sander and Schulz [42] uses novel local improvement algorithm based on max-flow and min-cut computations and more localized FM searches and, on the other hand, uses more sophisticated global search strategies transferred from multi-grid linear solvers. KaFFPa Evolutionary [43] is a distributed evolutionary algorithm to solve the graph partitioning problem. Pellegrini developed Scotch [44-46], uses recursive multilevel bisection and incorporate sequential and parallel graph partitioning methods. Chris Walshaw developed Jostle [47], a well-known sequential and parallel graph partitioning techniques. Party [48] developed by Robert, has implemented algorithms are Bubble/Sharp-optimization and Helpful Sets. Meyerhenke developed a software packages DibaP and PDibaP [49]. The tools mainly focus on hypergraph partitioning are Parkway [50] by Trifunovic, Zoltan [51] by Devine et al., and Cataiyure et al., proposed PaToH [52] produces high-quality partitioning.

III. FREQUENT SUBGRAPH MINING

The problem of frequent subgraph mining [53] is to find frequent subgraphs over a collection of graphs. Frequent subgraph mining delivers meaningfully structured information such as hot web access patterns, common protein structures, and Computational Molecular Biology [54]. Frequent subgraph mining can also be used in fraud detection to catch similar fraud translation patterns from millions of electronic payments. Furthermore, a graph is a general data structure which covers almost all previous well-researched frequent patterns, thus, it can unify the mining process into the same framework. Therefore, frequent subgraph mining has raised great interests.

In fact, this issue is precisely described here. If D is the entry dataset, the frequent subgraph mining intends to mine graphs with more support value in association with predetermined threshold. The graph support G_{sup} is denoted by $sup(G_{sup})$ and is given as

$$Sup(G_{sup}) = \frac{\sum_{i=1}^{n} G_i}{n}$$

Whereas n is the total number of graphs in the graph dataset [55].

A. Apriori based approach

Apriori-based frequent substructure mining algorithms share similar characteristics with Apriori based frequent item set mining algorithms [56]. The search used for frequent graphs starts with graphs of tiny "size" and proceeds in a bottom-up approach. Iteration at every time increases the size by one out of newly discovered frequent substructures. These latest substructures are first generated by joining two similar but slightly different frequent subgraphs that were discovered already. The frequency of newly formed graphs is then checked. The Apriori-based algorithms have considerable overhead when two size-k frequent substructures are joined to generate a size (k+1) graph candidates [57, 58].

The apriori based algorithms suffer two additional costs:

- Costly subgraph isomorphism test. Since subgraph isomorphism is an NP-Complete problem, no polynomial algorithm can solve. Thus, testing of false candidates (false test or false search) degrades the performance a lot.
- Costly candidate generation. The generation of size(k+1) subgraph candidates from size k frequent subgraphs are more complicated and costly than that of item sets as observed by Kuramochi and Karypis [59]

Apriori based algorithms include WARMER [60], AGM [61], FSG [62], FARMER [63], FFSG [64], HSIGRAM [65], GREW [66], SPIN [67], Dynamic GREW [68], ISG [69], MUSE [70], Weighted MUSE [71], MUSE-P [72] and UGRAP [73]. Few of the algorithms are discussed below Table II gives the information about remaining apriori based algorithms overview.

FFSM [64] is a novel subgraph mining algorithm by Huang et al. FFSM is utilized a vertical search schema with in an algebraic graph framework and utilizes a restricted join operation to generate candidates and stores embeddings to avoid explicit subgraph isomorphism testing. The experimental results were done on synthetic and real datasets exhibited that FFSM achieves a substantial performance over the state-of-the art subgraph mining algorithm approach.

HSIGRAM [65] uses adjacency matrix representation of graph and use iterative merging for subgraph generation. The aim of the HSIGRAM id to find the maximal independent set of graph, which are constructed out of the embeddings of a frequent subgraph so as to evaluate its frequency.

Table II: Comparison of FSM based on Apriori approach

Algorithm	*Nature of the graph*	*Search Strategy*	*Isomorphic test*	*Nature of output*	*Reference*
WARMR	Static	Breadth First	Approximate	Complete	Dehaspe et al., (1999) [60]
AGM	Static	Breadth First	exact	complete	Inokuchi et al., (2000) [61]
FSG	Static	Breadth First	Exact	incomplete	Kuramochi and Karypis, (2001) [59]
FARMER	Static	Breadth First	Approximate	complete	Nijssen and kok (2002) [62]
FFSG	Static	Depth First	exact	complete	Huan et al., (2003) [63]
HSIGRAM	Static	Breadth First	Adjustable	complete	Jiang et al., (2004) [64]
GREW	Static	Greedy	Exact	incomplete	Kuramochi and Karypis, (2004) [65]
SPIN	Static	Depth First	exact	incomplete	Huan et al., (2004) [66]
Dynamic GREW	Dynamic graphs	Depth First	Exact	incomplete	Kuramochi et al., (2006) [67]
ISG	Static	Breadth First	exact	complete	Thomas et al., (2009) [68]
MUSE	Static	Depth First	exact	complete	Li et al., (2009) [69]
Weighted MUSE	Static	Depth First	exact	complete	Jamil et al., (2011) [70]
MUSE-P	Static	Depth First	exact	complete	Li et al., (2010) [71]
UGRAP	Static	Depth First	exact	complete	Papapetrou et al., (2011) [72]

Zou et al., (2010) proposed an algorithm for Mining Frequent Subgraph Patterns from Uncertain Graph Data. In many real applications, graph data is liable to uncertainties because of incompleteness and imprecision of data. Mining such uncertain graph data is semantically different from and computationally more challenging than mining conventional exact graph data. A novel model of uncertain graphs is presented, and the frequent subgraph pattern mining problem is formalized by introducing a new measure, called expected support. An approximate mining algorithm called Mining Uncertain Sub graph patterns (MUSE) [70], is proposed to find a set of approximately frequent subgraph patterns by allowing an error tolerance on expected supports of discovered subgraph patterns. The algorithm uses efficient methods to determine whether a subgraph pattern can be output or not and new pruning method to reduce the complexity of examining subgraph patterns. Analytical and experimental results showed that the algorithm is very efficient, accurate, and scalable for large uncertain graph databases.

Khan et al., (2011) proposed a Weighted MUSE [71] by modifying the MUSE by assigning weights factor w (0, 1) to the edges of embeddings includes in the identified frequent subgraph pattern.

Another author investigated on frequent subgraph mining on uncertain graphs under probabilistic semantics [72]. Specifically, a measure called ϕ-frequent probability is introduced to evaluate the degree of recurrence of subgraphs. The goal is to find quickly all the subgraphs with frequent probability. The extensive experiments on real uncertain graph data verify that the algorithm is efficient and that the mining results have very high quality.

Papapetrou et al., (2011) proposed a method that uses an index of the uncertain graph database to reduce the number of comparisons needed to find frequent subgraph patterns. The algorithm depends on the apriori property for enumerating candidate subgraph patterns efficiently. Then, the index is used to reduce the number of comparisons required for computing the expected support of each candidate pattern. It also enables additional optimizations with respect to scheduling and early termination, that further increase the efficiency of the method. The evaluation of our approach on three real-world datasets and on synthetic uncertain graph databases exhibits the significant cost savings with respect to the state-of-the-art approach [73].

B. Pattern growth approch

In order to avoid limitations of apriori algorithms, Pattern growth algorithms have been developed, most of which adopt the depth-first search strategy. The pattern growth mining algorithm extends a frequent graph by adding a new edge, in each and every possible position. A possible issue with the edge extension is that the similar graph can be exposed many times. The gSpan algorithm solves this problem by introducing a rightmost extension technique, where the only extensions take place on the right-most paths. A rightmost path is the straight path from the straight vertex V_0 to the last vertex V_n, according to a depth-first search on the graph.

Pattern growth based approaches based on multiple small graphs included algorithms are Jianzhong, Yong and Hong were proposed algorithms are RP-FP, RP-GD [88], Yong, Jianzhong and Hong was proposed JPMiner [85], Yuhua et al. were proposed MSPAN [82], HybridGMiner [79], PATH [80], SEUS[81] Yiping, James, Jeffrey was proposed algorithm FCPMiner [83], Shijie, Jiong, Shirong was proposed by RING [84], Sayan, Ambuj was proposed GraphSig [86], Hsun-Ping, Chengp-Te was proposed TSP [87], , for more details [55]. Four well-known algorithms are discussed below and Table III gives the information about remaining pattern growth algorithms overview.

Holder et al., (1994) proposed a Substructure Discovery in the SUBDUE system. The SUBDUE [74] system, which uses the minimum description length principal to discover substructures that compress the database and represent structural concept in the data. By replacing previously discovered substructures in the data, multiple passes of SUBDUE produces a hierarchical description of the structural regularities in the data. The optimal background knowledge guides SUBDUE towards appropriate substructures for a particular domain. The use of an inexact graph matching allows a controlled amount of deviations in the instance of a substructure concept. The large amount of structural information that can be added to non-structural data collection on physical phenomena provides a large

testbed for comparing an integrated discovery system based on SUBDUE to other non-structural systems.

Borgelt and Berthold were proposed a MoFa algorithm [75] which is referred as Molecular Fragment Miner. The MoFa algorithm keeps all the embedding list of formerly found subgraphs with an edge and the extension function is restricted only to these embedding lists. Isomorphism tests can be done inexpensively by testing whether an embedding list can be polished in the similar way. The algorithm also uses structural pruning and environment knowledge to reduce support computation and to remove duplicates uses benchmark isomorphism testing.

Table III: Comparison of FSM based on Pattern Growth approach

Algorithm	*Nature of graphs*	*Search method*	*Isomorphic test*	*Nature of output*	*Reference*
GBI	Static	greedy	exact	complete	Yoshida et al., (1994) [73]
SUBDUE	Static	greedy	approximate	complete	Cook and Holder et al., (1994) [74]
MOFA	Static	Depth First	exact	complete	Borgelt and Berthold (2002) [75]
gSpan	Static	Depth First	exact	complete	Yan and Han (2002) [76]
ClosedGraph	Static	Depth First	exact	incomplete	Yan and Han (2003) [77]
GASTON	Static	Depth First	exact	complete	Nijssen and Kok (2004) [78]
HybridGMiner	Static	Depth First	exact	complete	Meinl et al (2004) [79]
SEUS	Static	Depth First	exact	complete	Gudes et al (2006) [80]
MSPAN	Static	Depth First	exact	complete	Li et al., (2009) [81]
FCPMiner	Static	Depth First	exact	complete	Ke et al (2009) [82]
RING	Static	Depth First	exact	complete	Zhang et al., (2009) [83]
JPMiner	Static	Depth First	exact	incomplete	Liu et al., (2009) [84]
GraphSig	Static	Depth First	exact	complete	Ranu et al., (2009) [85]
TSP	Dynamic	Depth First	exact	incomplete	Li and Hsieh (2010) [86]
RP-GD	Static	Depth First	exact	incomplete	Li et al., (2011) [87]
RP-FP	Static	Depth First	exact	incomplete	Li et al., (2011) [87]

Yan and Han were proposed CloseGraph [77] algorithm refer as Mining Closed Frequent Graph Pattern. It is the extension of the gSpan algorithm. A graph G is closed in a database if there exists no proper subgraph of G that has the same support as G. CloseGraph, is developed by discovering several interesting looping methods. The performance study shows that, CloseGraph not only dramatically reduces unnecessary subgraphs to be generated, but also significantly increases the efficiency of mining, particularly in the presence of large graph patterns.

Nijssen and Kok were presented a GASTON [78] algorithm which is referred as GrAph/Sequence/Tree extractiON. GASTON algorithm puts together frequent path, sub-tree, and subgraph, owing to the surveillance that most frequent substructures in molecular datasets are open trees. The algorithm suggests an explanation by divide the frequent subgraph mining procedure into the path, then sub-tree, and at last generate subgraph. As a result, the generated subgraphs are invoked when needed. Hence, GASTON functions optimally when graphs are generally trees or paths because the most expensive subgraph isomorphism testing is finding out subgraph mining phase. GASTON keeps all the embeddings in order to generate only new subgraphs that actually appear; thus saving on unnecessary isomorphism detection. GASTON can compute the frequency of a subgraph either with isomorphism tests or embedding lists

C. Classifiation of Frequent Subgraph Mining

Three main challenges of subgraph generation process are: (i) isomorphic subgraphs, (ii) infrequent subgraphs and (iii) the subgraphs that not exist in the graph database. Frequent subgraph mining algorithms use different approaches to remove or reduce these challenges. Figure 5 shows the frequent subgraph mining algorithms have been classified based on Apriori based and Pattern growth-based approaches. Grouping the similar approaches based on their nature of the input graph data.

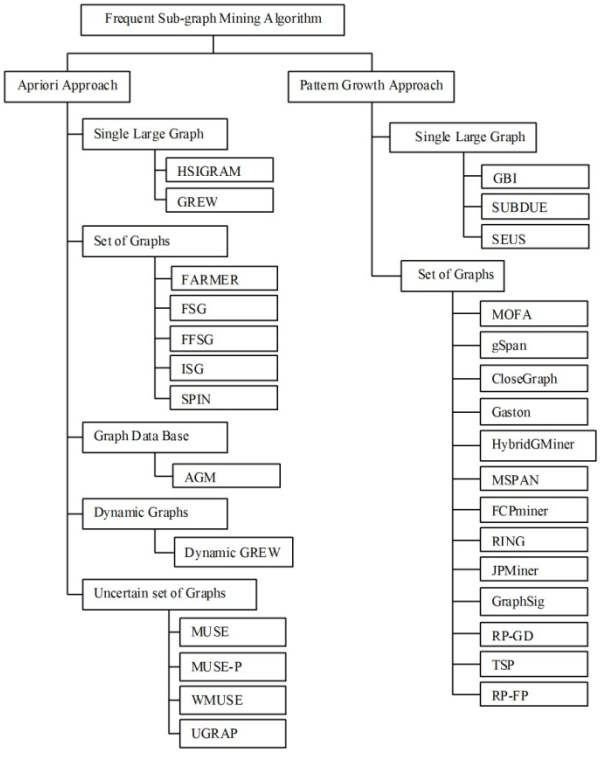


Figure 5. Classification of Frequent Subgraph Mining Algorithm

IV. PARALLEL FRAMEWORKS FOR LARGE SCALE GRAPHS

In this section describes an implementation for processing and generating large graph data sets with parallel and distributed algorithms on a cluster.

A. MapReduce

MapReduce [88] is a programming framework for processing the massive amount of data in a distributed approach. MapReduce provides users a simple interface for programming distributed algorithm, and it handles all the details of data distribution, replication, fault tolerance, and load balancing. A typical MapReduce task consists of three abstract functions: map, shuffle, reduce. At the "map" function, the raw data is read and processed to output (key, value) pairs. At the "shuffle" function, the out of the map phase is sent to reducers passing through the network so that the pair with the same key is grouped together. At the" reduced" function, the (key, value) pairs with the same key are processed to output another (key, value) pairs with the same key are processed to output another (key, value) pair. An iterative MapReduce program runs several MapReduce tasks, by feeding the output of the current task as the input of the next task.

B. Hadoop

Hadoop [89] is an open-source implementation of MapReduce programming model written in Java language. Hadoop uses the Hadoop Distributed File System (HDFS) [90] for its file system. There are several packages that run on top of Hadoop, including PIG [91], a high-level language for Hadoop, and HBASE, column-oriented data storage on top of Hadoop. Due to the simplicity, scalability and fault tolerance, big graph mining using Hadoop attracted significant attentions in the research community. The examples of these systems are Pegasus [92]: A peta-scale graph mining system – Implementation and observations was introduced by Kang et al., in 2009, Mahout [93], HaLoop [94] was introduced by Bu et al., in 2010, iMapReduce [95], Surfer [96] was introduced by Chen et al., in 2010 and Twister [97] was introduced by Ekanayake et al., in 2010.

C. Message Passing Interface

The Message Passing Interface (MPI) is a library particular for message-passing. MPI [98] is a standard Application Program Interface (API) that can be utilized to make application. The objective of the MPI is to give a broadly used standard for writing message-passing programs. The interface endeavors to be: practical, portable, efficient, flexible. MPI was designed for distributed memory architecture, which were becoming increasingly popular at time 1980s -1990s. An architecture trend changed, shared memory was combined over networks creating hybrid distributed memory/shared memory system. Today, MPI runs on virtually any hardware platform: Distributed Memory, Shared Memory, and Hybrid. The programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs.

There are several executions of MPI [99-101] which can be used to actualize parallel message-passing graph algorithms in various programming languages. MPI consists of very low-level communication primitives that do not provide any consistency or fault-tolerance. Programmers must build another level of deliberation all alone, which makes programming harder than bulk synchronous message-passing systems.

D. Bulk-Synchronize Parallel

Leslie Valiant (1990) developed a Bulk-Synchronize Parallel in Harvard University. In the middle of 1990 -1992, valiant and McColl dealt with thoughts for a distributed memory BSP programming model. Somewhere around 1992 and 1997, McColl leads a large research team at Oxford and developed different BSP programming libraries, languages, tools and also various massively parallel BSP algorithms. BSP Model for massage passing and collective communications. A BSP program consists of a sequence of supersteps. Each superstep comprises of three phases: Local computation, Process communication, Barrier synchronization. BSP programming enables you to write high-performance parallel computing algorithms for a wide range of scientific problems.

Pregel [102] presented the first bulk synchronous distributed message-passing framework, from which graph processing system has drawn. A few different frameworks are based on Pregel, including Giraph [103], GoldenOrb [104], Phoebus [105], Hama [106], JPregel [107], Bagel [108]. Giraph is the most well known and advanced of these systems. Giraph adds several features beyond the basic Pregel model, including master computation, shared aggregators, edge-oriented input, out-of-core computation and more. In 2012, apache Giraph is dispatched as an open partner to Pregel and an iterative graph processing system constructed for high scalability. Giraph can run as a typical Hadoop job that uses the Hadoop clustering infrastructure. Giraph model is appropriate for distributed implementation because it doesn't demonstrate any mechanism for detecting the order of execution within a superset, and all communication ids from superstep S to superstep S+1. During program execution, graph vertices are partitioned and assigned to workers. The default partition mechanism is hash-partitioning, but the custom partition also supports.

E. Other systems

Krepska et al., (2010) proposed a HipG [109] is a framework in which each vertex is a java object and the computation is done sequentially starting from a particular vertex. The code is expressed as if the graph is in a single machine, and the reads and writes to vertices residing in other machines are translated as RPC calls. HipG incurs significant overhead from RPC calls when executing algorithms, such as PageRank, that compute a value for each vertex in the graph. Zaharia et al., (2010) developed Spark [110] framework is a general clustering system, whose API is designed to express generic iterative components. As a result, programming graph algorithms on Spark required signification more coding effort than on graph processing system. GraphX [111] is apache Spark's API for graphs and graph-parallel computation. The requirement for instinctive, scalable tools for graph computation has lead to the advancement of new graph parallel system like Pregel and GraphLab[112] which are intended to proficiently execute graph algorithms. Unfortunately, these systems do not address the challenges of graph construction and transformation and offer restricted fault tolerance and support for interactive analysis.

V. PARALLEL FRAMEWORKS ON FSM

Bhuiyan et al., (2013) presented a novel iterative MapReduce framework based on Frequent subgraph mining algorithm called MIRAGE[113]. MIRAGE is complete as it returns all the frequent subgraphs for a given user-defined support, and it is efficient as it applies all the optimizations that the latest FSM algorithms adopt. The experiments with

real life and large synthetic datasets validate the effectiveness of MIRAGE for mining frequent subgraphs from large graph datasets.

Bhuiyan et al., (2014) proposed a frequent subgraph mining algorithm called FSM-H [114] which handles real world graph data grows both in size and quantity. FSM-H is a distributed frequent subgraph mining method over a MapReduce-based framework. The framework consists of three phases: data partition, a preparation phase, and mining phase. In data partition phase, FSM-H creates the partitions of input data along with the omission of infrequent edges from the input graph. Preparation and mining phase performs the actual mining task. FSM-H generates a complete set of frequent subgraphs for a given minimum threshold support, and it is efficient as it applies all the optimizations that the latest FSM algorithm adopt. The experiments with real life and large synthetic datasets validate the effectiveness of FSM-H for mining frequent subgraphs from large graph datasets.

VI. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we have described various discoveries of graph partitioning and frequent subgraph mining algorithms. Based on the different graph partitioning algorithms, we have classified graph partitioning approaches into static, dynamic and parallel implementations. In case of frequent subgraph mining for the large graph, a multiple sets of small graph data, dynamic graph data and uncertain graph data. The algorithms have been classified based on the apriori and pattern growth approaches. To handle a large scale graphs we presented various graph processing techniques and parallel frameworks for frequent subgraph mining are discussed in detail.

The future directions for identifying frequent subgraphs are:

1) For handling large graph data, very few methodologies are there for FSM. So, by adopting graph partitioning algorithms a large graph can be decomposed into a subset of graphs and then to the smaller graphs either apriori-based or pattern growth approach algorithms can be implemented to identify frequent subgraph mining.

2) GraphLab, Giraph and GraphX parallel frameworks provide good results while comparing with other different frameworks, thus one of the above-mentioned frameworks can be adopted for identifying frequent subgraphs in a large graph data.

VII. REFERENCES

[1] D. Chakrabarti and C. Faloutsos, “Graph mining: Laws, generators, and algorithms,” ACM comput. Surv. 38, 1, Article 2, jun. 2006, doi:10.1145/1132952.1132954.

[2] C. Wang and S. Parthasarathy, “Parallel Algorithms for Mining Frequent Structural Motifs in Scientific Data,” Proc. ACM International Conference on Supercomputing (ICS 04). Jun. 2004, pp. 31-40, doi:10.1145/1006209.1006215.

[3] J. R. Punin, M. Krishnamoorthy, and M. J. Zaki, “LOGML-Log Markup Language for Web Usage Mining,” WEBKDD Workshop: Mining Log Data across All Customers Touch Points, 2001, pp. 88-112.

[4] H. Mannilla, H. Toivonen and I. Verkamo, “Discovering Frequent Episodes in Sequences.” Proc. IEEE International Conference on Knowledge Discovery and Data Mining (ICKDD 95), 1995, pp. 210-215.

[5] M. Deshpande, M. Kuramochi and G. Karypis, “Frequent sub-structure based approaches for classifying chemical compounds,” IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 8, 2005, pp. 1036-1050.

[6] M. A. Srinuvasu, P. Padmaja and Y. Dharmateja, “Subgraph relative frequency approach for extracting interesting substructures from molecular data,” International Journal of Computer Engineering & Technology, 2013, vol.4 no. 4, pp. 400-411.

[7] H. Haiyan Hu, Xifeng Yan, Yu Huang1, Jiawei Han, Xianghong Jasmine Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. Bioinformatics, vol. 21, no. 1, 2005, pp. 213-221.

[8] G. Ciriello and C. Guerra, “A review on models and algorithms for motif discovery in protein-protein interaction networks,” Briefings Functional Genomics & Proteomic, vol. 7, no. 2, 2008, pp. 147–156.

[9] J. Huan, W. Wang, J. Prins, and J. Yang, “Spin: Mining maximal frequent subgraphs from graph databases,” UNC Technical Report TR04-018, 2004.

[10] P. Raghavan, “Social Networks on the Web and in the Enterprise,” Proce. First Asia-Pacific Conference on Web Intelligence, 2001, pp. 58-60.

[11] C. Bichot and P. Siarry, “Graph Partitioning,” Wiley, New York, 2011.

[12] D. J. Cook and L. B. Holder, “Mining Graph Data,” Wiley, New Jersey, 2007.

[13] M. R. Garey and D. S. Johnson, “Computers and Intractability: A Guide to the Theory of Np-Completeness,” W. H. Freeman & Co., New York, NY, USA. 1990.

[14] R. Preis and R. Diekmann, “PARTY- a software library for graph partitioning,” Advances in Computational Mechanics with Parallel and Distributed Processing, CIVIL COMP PRESS, 1997, pp. 63-71.

[15] S. T. Barnard and H. D. Simon, “A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems,” Concurrency: Practice and Experience, 1994, vol.6, pp. 101-117.

[16] B. Hendrickson and R. Leland, “A multilevel algorithm for partitioning graphs,” Proce. ACM/IEEE Conference on Supercomputing. 1995, pp. 28-28.

[17] B. W. Kernighan and S. Lin “An efficient heuristic procedure for partitioning graphs,” The Bell System Technical Journal, 1970, vol. 49, no. 0, pp. 291–307.

[18] C. M. Fiduccia and R. M. Mattheyses, “A linear time heuristic for improving network partitions,” Proce. IEEE Design Automation Conference, 1982, pp.175–181.

[19] B. Monien, R. Preis and R. Diekmann, “Quality matching and local improvement for multilevel graph-partitioning,” Parallel Computing, vol. 26, no. 12, 2000, pp. 1609-1634.

[20] C. Chevalier and I. Safro, "Comparison of Coarsening Schemes for Multilevel Graph Partitioning," Proce. International Conference on Learning and Intelligent Optimization. 2009, pp. 191–205.

[21] I. Safro, P. Sanders and C. Schulz, "Advanced coarsening schemes for graph partitioning," Proce. International Symposium on Experimental Algorithms (SEA'12). 2012, pp. 369–380.

[22] A. Grama, G. Karypis and V. Kumar, "Introduction to Parallel Computing," Addison-Wesley, 2nd edition, 2003.

[23] 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.

[24] S.T Barnard and H. Simon. "A parallel implementation of multilevel recursive spectral bisection for application to adaptive unstructured meshes," Proce. SIAM conference on Parallel Processing for Scientific Computing.1995, pp. 627–632.

[25] G. Karypis and V. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," Journal of Parallel and Distributed Computing, vol. 48, no. 1, 1998, pp. 71-95.

[26] G. Karypis and Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs," SIAM Review, vol. 41, no. 2, 1999, pp. 278-300.

[27] G. L. Miller, S. H. Teng and S. A. Vavasis, "A unified geometric approach to graph separators," Proce. Annual Symposium on Foundations of Computer Science, 1991, pp. 538–547.

[28] H. Heath and P. Raghavan, "A Cartesian parallel nested dissection algorithm," SIAM Journal of Matrix Analysis and Applications, vol. 16, no. 1, 1995, pp. 235-253.

[29] K. Schloegel, G. Karypis and V. Kumar, "Wavefront division and LMSR: algorithms for dynamic repartitioning of adaptive meshes," Technical Report TR 98-034, Dept. of Computer Science and Engineering, Univ. of Minnesota. 1998.

[30] C. Walshaw, M. Cross and M. Everett, "Parallel dynamic graph partitioning for adaptive unstructured meshes," Journal of Parallel and Distributed Computing, vol. 47, no. 2, 1997, pp. 102-108.

[31] Lu Wang, Yanghua Xiao, Bin Shao, Haixun Wang. "How to partition a billion-node graph," Proce. IEEE 30th International Confernce on Data Engineering (ICDE). 2014, pp. 568-579.

[32] C. E. Tsourakakis, C. Gkantsidis, B. Radunovic and M. Vojnovic, "FENNEL: Streaming Graph Partitioning for Massive Scale Graphs," Technical Report MSR-TR-2012-113, 2012.

[33] H. Meyerhenke, P. Sanders and C. Schulz, "Parallel Graph Partitioning for Complex Networks," CoRR, 2014, abs/1404.4797.

[34] C. Martella, D. Logotheti and G. Siganos, "Spinner: Scalable Graph Partitioning for the Cloud," arXiv: 1404.3861v1, 2014.

[35] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[36] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," SIAM J on Scientific Computing, vol. 20, no. 1, 1998, pp. 359-392.

[37] G. Karypis and V. Kumar, "hMeTiS 1.5: A hypergraph partitioning package," Technical report, Dept. of Computer Science and Engineering, Univ. of Minnesota, 1998.

[38] G. Karypis and V. Kumar, "MeTiS 4.0: Unstructured graph partitioning and sparse matrix ordering system," Technical report, Dept. of Computer Science and Engineering, Univ. of Minnesota, 1998.

[39] G. Karypis and V. Kumar, "Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs," Proce. ACM/IEEE Supercomputing'96 conference. 1996.

[40] G. Karypis, K. Schloegel and V. Kumar, "ParMeTiS: Parallel graph partitioning and sparse matrix ordering library," Technical report, Dept. of Computer Science and Engineering, University of Minnesota, 1997.

[41] P. Sanders and C. Schulz, "Engineering Multilevel Graph Partitioning Algorithms," Proce. European Symposium on Algorithms, 2011, pp. 469–480

[42] P. Sanders and C. Schulz, "Think Locally, Act Globally: Highly Balanced Graph Partitioning," Experimental Algorithms Lecture Notes in Computer Science, vol. 7933, 2013, pp. 164-175.

[43] M. Holtgrewe, P. Sanders and C. Schulz, "Engineering a Scalable High Quality Graph Partitioner," Proce. IEEE International Symposium on Parallel & Distributed Processing (IPDPS). 2010, pp.1–12.

[44] C. Chevalier and F. Pellegrini, "Improvement of the Efficiency of Genetic Algorithms for Scalable Parallel Graph Partitioning in a Multi-level Framework," Euro-Par 2006 Parallel Processing Lecture Notes in Computer Science, 2006, vol. 4128, pp. 243-252.

[45] C. Chevalier and F. Pellegrini, "PT-Scotch: A tool for efficient parallel graph ordering," Parallel Computing, 2008, vol. 34, no. 6, pp. 318-331.

[46] F. Pellegrini and J. Roman, "SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," HPCN-Europe, Springer LNCS 1067, 1996, pp. 493-498.

[47] C. Walshaw and M. Cross, "JOSTLE: Parallel Multilevel Graph-Partitioning Software - An Overview," In Mesh Partitioning Techniques and Domain Decomposition Techniques, Civil-Comp Ltd., 2007, pp. 27-58 .

[48] R. Preis and R. Diekmann, "PARTY- a software library for graph partitioning," Advances in Computational Mechanics with Parallel and Distributed Processing, CIVIL COMP PRESS, 1997,pp. 63-71.

[49] H. Meyerhenke, B. Monien and T. Sauerwald ,"A new diffusion-based multilevel algorithm for computing graph partitions," Journal of Parallel and Distributed Computing, 2009, vol. 69, no. 9, pp. 750-761.

[50] Trifunovic and W. J. Knottenbelt, "Parallel Multilevel Algorithms for Hypergraph Partitioning," Journal of Parallel and Distributed Computing, 2008, vol. 68, no. 5, pp. 563-581.

[51] Umit V Catalyurek, Erik G Boman, Karen D Devine, Doruk Bozdag, Robert T. Heaphy, Lee Ann Riesen. "A repartitioning Hypergraph model for dynamic load balancing," Journal of Parallel Distribution and Computing , 2009, vol. 69, no. 8, pp. 711-724.

[52] U. V. Catalyure and C. Aykanat, "PaToH: Partitioning Tool for Hypergraphs," 2011.

[53] C. C. Aggarwal and H. Wang, editors. Managing and Mining Graph Data, volume 40 of Advances in Database Systems. Springer, 2010.

[54] S. Aluru, "Handbook of Computational Molecular Biology," Chapman and Hall/CRC. 2006.

[55] S. U. Rehman, S. Asghar, Y. Zhuang, and S. Fong. "Performance evaluation of frequent subgraph discovery techniques," Mathematical Problems in Engineering, 2014, pp. 1-5.

[56] 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 Knowledge Discovery (PKDD 00), 2000, pp. 13-23.

[57] K. Lakshmi and T. Meyyappan , "A comparative study of frequent subgraph mining algorithms," International Journal of Information Technology Convergence and Services (IJITCS), 2012, vol. 2, no. 2, pp. 23-39.

[58] K. Lakshmi and T. Meyyappan, "Frequent subgraph mining algorithms - a survey and framework for classification," Proce. Conference on Innovations in Theoretical Computer Science (ITCS 12), 2012, pp. 189–202,.

[59] M. Kuramochi and G. Karypis, "Frequent Subgraph Discovery," Proce. IEEE International Conference on Data Mining (ICDM 01). 2001, pp. 313-320.

[60] L. Dehaspe and H. Toivonen, "Discovery of FrequentDatalog Patterns", Data Mining and Knowledge Discovery, 1999, pp.7-36.

[61] 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.

[62] S. Nijssen and J. Kok, "Faster association rules for multiple relations," Proce. International Joint Conference on Artificial Intelligence (IJCAI'01), 2001, pp. 891–896.

[63] J. Huan, W. Wang and J. Prins, "Efficient mining of frequent subgraph in the presence of isomorphism," UNC computer science techonology report TR03-021, 2003

[64] 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.

[65] M. Kuramochi and G. Karypis, "GREW: A Scalable Frequent Subgraph Discovery Algorithm", Proce. International Conference on Data Mining (ICDM'04), Brighton, pp.439–442, 2004.

[66] J. Huan, W. Wang, J. Prins and J. Yang, "Spin: Mining maximal frequent subgraphs from graph databases," UNC Technical Report TR04-018, 2004.

[67] 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.

[68] Thomas L, Valluri S, Karlapalem K. "Isg: Itemset based subgraph mining," Technical Report, Center for Data Engineering, IIIT, Hyderabad, 2009.

[69] Z. Zou and J. Li, "Mining Frequent Subgrph Patterns from Uncertain Graph Data," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 9, 20 may. 2010, pp. 1203-1218, doi:10.1109/TKDE.2010.80

[70] 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.

[71] Z. Zou, H. Gao and J. Li "Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics," Proce. ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 10), 2010, pp. 633–642, doi:10.1145/1835804.1835885.

[72] O. Papapetrou, E. Loannou, D. Skoutas, "Efficient Discovery of Frequent Subgraph Patterns in Uncertain Graph Database," Proce. International Conference on Extending Database Technology (EDBT/ICDT '11), Anastasia Ailamaki, Sihem Amer-Yahia, Jignesh Pate, Tore Risch, Pierre Senellart, and Julia Stoyanovich (Eds.). ACM, New York, NY, USA, pp. 355-366, doi:10.1145/1951365.1951408.

[73] K. Yoshida, H. Motoda and N. Indurkhya, "Graph-based Induction as a Unified Learning Framework", Journal of Applied Intelligence, pp.297–328.

[74] L. B. Holder, D. J. Cook and S. Djoko, "Substructure Discovery in the Subdue System", Proce. AAAI'94 workshop knowledge discovery in databases (KDD'94), WA, 1994, pp 169–180.

[75] C. Borgelt and M. R. Berhold, “Mining molecular fragments: Finding relevant substructures of molecules,” Proce. IEEE International Conference on Data Mining (ICDM 02), IEEE Press, 2002, pp. 51-58, doi:10.1109/ICDM.2002.1183885.

[76] X. Yan and J. Han, “gSpan: Graph-based substructure pattern mining,” Proce. IEEE International Conference on Data Mining (ICDM 02), IEEE press, 2002, pp. 721-724, doi:10.1109/ICDM.2002.1184038.

[77] X. Yan and J. Han, “CloseGraph: Mining closed frequent graph patterns,” Proce. ACM SIGKDD International Conference Knowledge Discovery and Data Mining (KDD 03). Aug. 2003, pp. 286-295, doi:10.1145/956750.956784.

[78] S. Nijssen and J. Kok, “A quickstart in frequent structure mining can make a difference,” Proce. ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 04), Aug. 2004, pp. 647–652, doi:10.1145/1014052.1014134.

[79] T. Meinl, and M. R. Berthold, “Hybrid fragment mining with *MoFa* and FSG,” Proce. IEEE International Conference on Systems, Man and Cybernetics. 2004, pp. 4559-4564, doi: 10.1109/ICSMC.2004.1401250.

[80] E. Gudes, E. Shimony and N. Vanetik, “Discovering Frequent Graph Patterns Using Disjoint Paths”, IEEE Transactions on Knowledge and Data Engineering, Los Angeles, 2006, pp.1441–1456, doi:10.1109/TKDE.2006.173.

[81] Y. Li, Q. Lin, G. Zhong, D. Duan, Y. Jin and W. Bi. “A directed labeled graph frequent pattern mining algorithm based on minimum code,” Proce. International Conference on Multimedia and Ubiquitous Engineering (ICMUE ’09), 2009, pp. 353-359, doi:10.1109/MUE.2009.67.

[82] Y. Ke, J. Cheng and J. X. Yu. “Efficient Discovery of Frequent Correlated Subgraph Pairs,” Proce. IEEE International Conference on Data Mining (ICDM ’09), IEEE press, 2009, pp. 239-248.

[83] S. Zhang, J. Yang and S. Li. “RING: An Integrated Method for Frequent Representative Subgraph Mining, Proce. International Conference on Data Mining (ICDM ’09), 2009, pp. 1082-1087, doi:10.1109/ICDM.2009.96.

[84] Y. Liu, J. Li and H. Gao. “JPMiner: Mining Frequent Jump Patterns from Graph Databases,” Proce. International Conference on Fuzzy Systems and Knowledge Discovery (ICFSKD ’09), 2009, pp. 114-118.

[85] S. Ranu, K. Ambuj K, Singh. “GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases,” Proce. IEEE International Conference on Data Engineering (ICDE ’09), IEEE press, 2009, pp. 844-855, doi:10.1109/ICDE.2009.133.

[86] H. P. Hsieh and C. T. Li. “Mining Temporal Subgraph Patterns in Heterogeneous Information Networks,” Proce. IEEE International Conference on Social Computing / International Conference on Privacy, Security, Risk and Trust, 2010, pp. 282-287, doi:10.1109/SocialCom.2010.47.

[87] J. Li, Y. Liu and H. Gao. “Efficient algorithms for summarizing graph patterns,” IEEE Transactions on Knowledge and Data Engineering, 2011, vol. 23, no.9, pp. 1388-1405, doi:10.1109/TKDE.2010.48.

[88] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Proce. Symposium on Operating System Design and Implementation, pp. 137–150, 2004.

[89] Apache Hadoop. http://hadoop.apache.org/.

[90] Hadoop Distributed File System. http://hadoop.apache.org/hdfs/.

[91] C. Olston, B. Reed, U.Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: a Not-So-Foreign Language for Data Processing," Proce. ACM/SIGMOD International Conference on Management of Data (SIGMOD), 2008, pp. 1099-1110, doi:10.1145/1376616.1376726.

[92] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "PEGASUS: A peta-scale graph mining system – Implementation and observations," Proce. IEEE International Conference on Data Mining, 2009, pp. 229–238, doi:10.1109/ICDM.2009.14.

[93] Apache Mahout. http://mahout.apache.org/.

[94] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: Efficient iterative data processing on large clusters," Proce. International Conference on Very Large Databases, pp. 285–296, 2010.

[95] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "iMapreduce: A distributed computing framework for iterative computation," DataCloud, 2011.

[96] R. Chen, X. Weng, B. He, and M. Yang, "Large graph processing in the cloud," Proce. International Conference on Management of Data, pp. 1123–1126, 2010.

[97] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," Proce. ACM International Symposium on High Performance Distributed Computing, HPDC ’10, pp. 810–818, New York, NY, USA, 2010. ACM.

[98] Open MPI. http://www.open-mpi.org/.

[99] MPICH2. http://www.mcs.anl.gov/research/projects/mpich2/.

[100] pyMPI. http://pympi.sourceforge.net/.

[101] OCaml MPI. http://forge.ocamlcore.org/projects/ocamlmpi/.

[102] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A System for LargeScale Graph Processing," Proce. ACM SIGMOD International Conference on Management of Data, pp. 155–166, 2011.

[103] Apache Incubator Giraph. http://incubator.apache.org/giraph//.

[104] GoldenOrb. http://www.raveldata.com/goldenorb/.

[105] Phoebus. https://github.com/xslogic/phoebus.

[106] Apache Hama. http://incubator.apache.org/hama/

[107] JPregel. http://kowshik.github.com/JPregel/.
[108] Bagel Programming Guide. https://github.com/mesos/spark/wiki/BagelProgramming-Guide/.
[109] E. Krepska, T. Kielmann, W. Fokkink, and H. Bal, "A high-level framework for distributed processing of large-scale graphs," Proce. International Conference on Distributed Computing and Networking, pp. 1123–1126, 2010.
[110] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I.Stoica, "Spark: cluster computing with working sets," Proce. USENIX conference on Hot topics in cloud computing, HotCloud'10, pp. 10–10, Berkeley, CA, USA, 2010. USENIX Association.
[111] Graphx. http://spark.apache.org/graphx/
[112] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "GraphLab: A New Framework for Parallel Machine Learning," CoRR, abs/1006.4990, 2010.
[113] M. A. Bhuiyan and M. A. Hasan, "MIRAGE: An iterative MapReduce based frequent subgraph mining algorithm," arXiv:1307.5894v1 [cs.DB] 22 Jul 2013.
[114] M. A. Bhuiyan and M. A. Hasan, "FSM-H: Frequent subgraph mining algorithm in Hadoop," Proce. IEEE International Conference on Big Data (ICBD 14), pp. 9-16, doi:10.1109/BigData.Congress.2014.12

Short Bio data for the Authors

Appala Srinuvasu Muttipati is a Research Scholar in Computer Science and Engineering from GITAM University, Visakhapatnam, Andhra Pradesh, India. He received his M.Tech Degree in Computer Science and Technology from GITAM University in 2011. MCA Degree from Andhra University in 2008. His Current research areas include Data mining and Graph mining.

Padmaja Poosapati is a Associate Professor in Department of Information Technology at GITAM University, Visakhapatnam, Andhra Pradesh, India. She received her Master degree in Computer Science and Engineering from Andhra University in 1999 and PhD degree in Computer Science and Engineering from Andhra University 2010. Her current research interests include Clustering and Classification in Data mining and Graph mining.