Volume 6, Issue 9, September 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Effective Mapreduces Process for Bigdata Using Load Distribution Frequent Sub Graph Mining Algorithm

[1]K. M. Padmapriya, [2]M. Keerthana
[1] M.Sc., M.Phil., Asst. Professor, PG & Research Department of Computer Science, SSM College of Arts and Science, Komarapalayam, Tamilnadu, India
[2] M.Phil Research Scholar, SSM College of Arts and Science, Komarapalayam, Tamilnadu, India

Abstract— Frequent subgraph mining (FSM) is an important task for exploratory data analysis on graph data especially when the graph is huge. In the recent years, many algorithms have been proposed to solve this task. These algorithms assume that the mining task's data structure is small enough to fit in the main memory in the systems. However, as the real-world graph data grows, both in quantity and size, such an assumption could not be met. To overcome this, some graph database-centric methods have been proposed in real problem for solving FSM however, a distributed solution using MapReduce paradigm has not been explored extensively. Since MapReduce is becoming the de-facto paradigm for computation on massive data, an efficient FSM algorithm on this paradigm is of huge demand. This paper proposes a frequent subgraph mining algorithm called FSM-H which uses an iterative MapReduce based framework. LB-FSM 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 LB-FSM algorithms adopt. The experiments with real life and large synthetic datasets validate the effectiveness of LB-FSM-H for mining frequent subgraphs from large graph datasets.

Keywords—Big Data, Frequent Subgraph Mining, Graph Database Centric method, Map Reduce, LB-FSM

I. INTRODUCTION

This Big data is a broad term for data sets so large or complex that traditional data processing applications are inadequate. Challenges include analysis, capture, search, sharing, storage, transfer, visualization, and information privacy. The term often refers simply to the use of predictive analytics or other certain advanced methods to extract value from data, and seldom to a particular size of data set. Accuracy in big data may lead to more confident decision making. And better decisions can mean greater operational efficiency, cost reduction and reduced risk.

Big Data processing mainly depends on parallel programming models like MapReduce, as well as providing a cloud computing platform of Big Data services for the public. MapReduce is a batch-oriented parallel computing model. There is still a certain gap in performance with relational databases. Improving the performance of MapReduce and enhancing the real-time nature of large-scale data processing have received a significant amount of attention with MapReduce parallel programming being applied to many machine learning and data mining algorithms.

Fig 1.1 Bigdata

Data mining algorithms usually need to scan through the training data for obtaining the statistics to solve or optimize model parameters. It calls for intensive computing to access the large-scale data frequently. To improve the efficiency of algorithms, Chu et al. proposed a general-purpose parallel programming method, which is applicable to a large number of machine learning algorithms based on the simple MapReduce programming model on multi-core processors.

Classical data mining algorithms are realized in the framework, includes locally weighted linear regression, k-Means, logistic regression, naive Bayes, linear support vector machines, the independent variable analysis, Gaussian discriminant analysis, expectation maximization, and back-propagation neural networks. With the analysis of these classical machine learning algorithms, the computational operations in the algorithm learning process could be transformed into a summation operation on a number of training data sets.

Summation operations could be performed on different subsets independently and achieve penalization executed easily on the MapReduce programming platform. Therefore, a large-scale data set could be divided into several subsets and assigned to multiple Mapper nodes. Then various summation operations could be performed on the Mapper nodes to collect intermediate results. Finally, learning algorithms are executed in parallel through merging summation on Reduce nodes.

MapReduce is useful in a wide range of applications, including distributed pattern-based searching, distributed sorting, web link-graph reversal, singular value decomposition, web access log stats, inverted index construction, document clustering, machine learning and statistical machine translation. The MapReduce model has been adapted to several computing environments like multi-core, many-core systems, desktop grids, volunteer computing environments, dynamic cloud environments, and mobile environments. At Google, MapReduce was used to completely regenerate Google's index of the World Wide Web. In this paper introducing, FSM-H, a novel iterative MapReduce based frequent subgraph mining algorithm which is complete. And design novel data structures to save and consequently propagate the states of the mining process over different iterations. The proposed novel technique is empirically demonstrate the performance of FSM-H on synthetic as well as real world large datasets.

In this study, developing algorithms that discover all the frequently occurring subgraphs in a large graph database is particularly challenging and computationally intensive as graph and subgraph isomorphism plays a vital role entire computations. This research proposes an effective solution in the form of a subgraph mining approach, to one of the main challenges for graph-based knowledge discovery and data mining systems, which is to scale up their data interpretation abilities to discover frequent subgraphs in large datasets of any distributed or parallel resources.

II. REVIEW

Jeffrey Dean and Sanjay Ghemawat et al [1] describe the MapReduce and its a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/ value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication.

Jie Tang et al [2] describe the large social networks, nodes (users, entities) are influenced by others for various reasons. For example, the colleagues have strong influence on one's work, while the friends have strong influence on one's daily life. How to differentiate the social influences from different angles (topics). How to quantify the strength of those social influences. How to estimate the model on real large networks and implement tTo address these fundamental questions, we propose Topical Affinity Propagation (TAP) to model the topic-level social influence on large networks.

U Kang, Charalampos et al [3] In this describe PEGASUS, an open source Peta Graph Mining library which performs typical graph mining tasks such as computing the diameter of the graph, computing the radius of each node and finding the connected components. As the size of graphs reaches several Giga-, Tera- or Peta-bytes, the necessity for such a library grows too. To the best of our knowledge, PEGASUS is the first such library, implemented on the top of the H ADOOP platform, the open source version of M APREDUCE. Many graph mining operations (PageRank, spectral clustering, diameter estimation, connected components etc.) are essentially a repeated matrix-vector multiplication. In this paper we describe a very important primitive for PEGASUS, called GIM-V (Generalized Iterated Matrix-Vector multiplication). GIM-V is highly optimized, achieving (a) good scale-up on the number of available machines (b) linear running time on the number of edges, and (c) more than 5 times faster performance over the non-optimized version of GIM-V. Our experiments ran on M45, one of the top 50 supercomputers in the world. We report our findings on several real graphs, including one of the largest publicly available Web Graphs, thanks to Yahoo!, with ≈ 6,7 billion edges.

U Kang Brendan Meeder et al [4] set a graph with billions of nodes and edges, how can we find patterns and anomalies? Are there nodes that participate in too many or too few triangles? Are there close-knit near-cliques? These questions are expensive to answer unless we have the first several eigenvalues and eigenvectors of the graph adjacency matrix. However, eigensolvers suffer from subtle problems (e.g., convergence) for large sparse matrices, let alone for billion-scale ones. We address this problem with the proposed HEIGEN algorithm, which we carefully design to be accurate, efficient, and able to run on the highly scalable MAPREDUCE (HADOOP) environment. This enables HEIGEN to handle matrices more than 1000 larger than those which can be analyzed by existing algorithms.

Siddharth Suri Sergei et al [5] describe the clustering coefficient of a node in a social network is a fundamental measure that quantifies how tightly-knit the community is around the node. Its computation can be reduced to counting the number of triangles incident on the particular node in the network. In case the graph is too big to fit into memory, this is a non-trivial task, and previous researchers showed how to estimate the clustering coefficient in this scenario. A different avenue of research is to perform the computation in parallel, spreading it across many machines. In recent years MapReduce has emerged as a de facto programming paradigm for parallel computation on massive data sets. The main focus of this work is to give MapReduce algorithms for counting triangles which we use to compute clustering coefficients.

III. FREQUENT SUBGRAPH MINING

Frequent Subgraph Mining (FSM) is the essence of graph mining. The objective of FSM is to extract all the frequent subgraphs, in a given data set, whose occurrence counts are above a specified threshold. The straightforward idea behind FSM is to "grow" candidate subgraphs, in either a breadth first or depth first manner (candidate generation), and then determine if the identified candidate subgraphs occur frequently enough in the graph data set for them to be considered interesting (support counting). The two main research issues in FSM are thus how to efficiently and effectively (i) generate the candidate frequent subgraphs and (ii) determine the frequency of occurrence of the generated subgraphs.

Effective candidate subgraph generation requires that the generation of duplicate or superfluous candidates is avoided. Occurrence counting requires repeated comparison of candidate subgraphs with subgraphs in the input data, a process known as isomorphism checking. FSM, in many respects, can be viewed as an extension of Frequent Itemset Mining (FIM) popularised in the context of association rule mining. Consequently, many of the proposed solutions to addressing the main research issues effecting FSM are based on similar techniques found in the domain of FIM.

The existing system involves mining frequent sub-graphs from the given graph database (A database with 'N' graphs). Here MapReduce approach is used which is a programming model that enables distributed computation over massive data. In special, Iterative MapReduce is used here which can be defined as a multi staged execution of map and reduce function pair in a cyclic fashion, i.e. the output of the stage i reducers is used as an input of the stage i + 1 mappers. An external condition decides the termination of the job. Graph isomorphism is also considered.

A fundamental feature for exact search based algorithms is that the mining is complete, i.e. the mining algorithms are guaranteed to find all frequent subgraphs in the input data. Complete mining algorithms perform efficiently only on sparse graphs with a large amount of labels for vertexes and edges. Due to this completeness restriction, these algorithms undertake extensive subgraph isomorphism comparison, either explicitly or implicitly, resulting in a significant computational overhead. The effectiveness of integrating constraints into the FSM process is influenced by many aspects, including the properties of the data and the pruning cost. Constraint based mining algorithms therefore need to take into account the trade-off between the pruning cost and any potential benefit

- Before sending input graph data to nodes, they are not balanced. For example, one node may be assigned with more big graphs and other node with small graphs.
- Nodes have to wait for Reduce phase and can start the process only after all the mapper processes are completed in all nodes.
- Overall time efficiency is poor.
- Candidate generation strategy poor.
- The mechanism for traversing the search space and low occurrence counting process.
- Single graph based FSM applied.
- Graph isomorphism is neither known to be solvable in polynomial time nor NP-complete.

IV. ENHANCED FREQUENT SUBGRAPH MINING

Frequent pattern mining has been a focused theme in data mining for over a decade. Abundant literature was dedicated to this area, making tremendous progress, including frequent itemset mining, sequential pattern mining and so forth. Frequent subgraphs are subgraphs found from a collection of graphs or single large graph with support no less than a user-specified threshold. Frequent subgraphs are useful at characterizing graph datasets, classifying and clustering graphs, and building structural indices. Differentiate the two aforementioned scenarios multi-graph and single-graph, and this paper focuses on frequent subgraph mining (FSM) on the single-graph setting.

Distributed FSM from single massive graphs is challenging, due to not only the special constraints of FSM algorithm design, but also the deficient support from existing distributed programming frameworks. First, an FSM algorithm computes the support of a candidate subgraph over the entire input graph. In a distributed platform, if the input graph is partitioned over various worker nodes, the local support of the subgraph is not much useful for deciding whether subgraph is globally frequent. Also, the support computation cannot be delayed arbitrarily, since candidate frequent subgraphs can be generated only from frequent subgraphs as per Apriori principle.

Additionally, although there are several existing models, including MapReduce , the de facto big data processing framework, they also do not accommodate graph algorithms. Among the distributed graph processing frameworks, Pregel is recognized for its scalability, flexibility, fault tolerance and a number of other attractive features. It is a vertex-centric programming model such that developers usually only need to submit processing scripts on vertices to the framework, which will handle the remaining issues such as graph partition and synchronization. However, it is suggested that structure-related computation may not fit Pregel naturally. Therefore, mapping a structure mining algorithm onto Pregel requires non-trivial efforts, since Pregel does not specify implementation details for self-defined functions.

Advantages of the proposed system

The thesis focuses on efficient implementation of FSM over single massive graphs on a Pregel-like extensible computing platform. To the best of our knowledge, this is among the first attempts to address the problem at scale under a modern distributed programming framework.

1. Ppropose a systematic solution for FSM in single massive graphs using Pregel-like distributed programming paradigm.
2. To devise two optimization techniques to enhance the baseline algorithm, reducing communication cost and distribution overhead, respectively.

3. To evaluate the resulting algorithm pegi with extensive experiments on public real-life data. The experiment results confirm the efficiency and scalability of the proposed methods.

The proposed system involves all the existing system approaches. In addition, before nodes are assigned with graphs for Map process, the graphs are balanced such that all the nodes get correct number of graphs with nodes count. For example, two small graphs are given to Node A and one big graph is given to Node B. So, the map processes are completed in less interval in all the nodes so that reduce phase can be started immediately.

- Before sending input graph data to nodes, they are balanced. For example, two nodes are equal number of nodes and edges.
- Nodes complete the Mapper process in less intervals so that Reduce phase can be started with minimum delay.
- Overall time efficiency is increased.

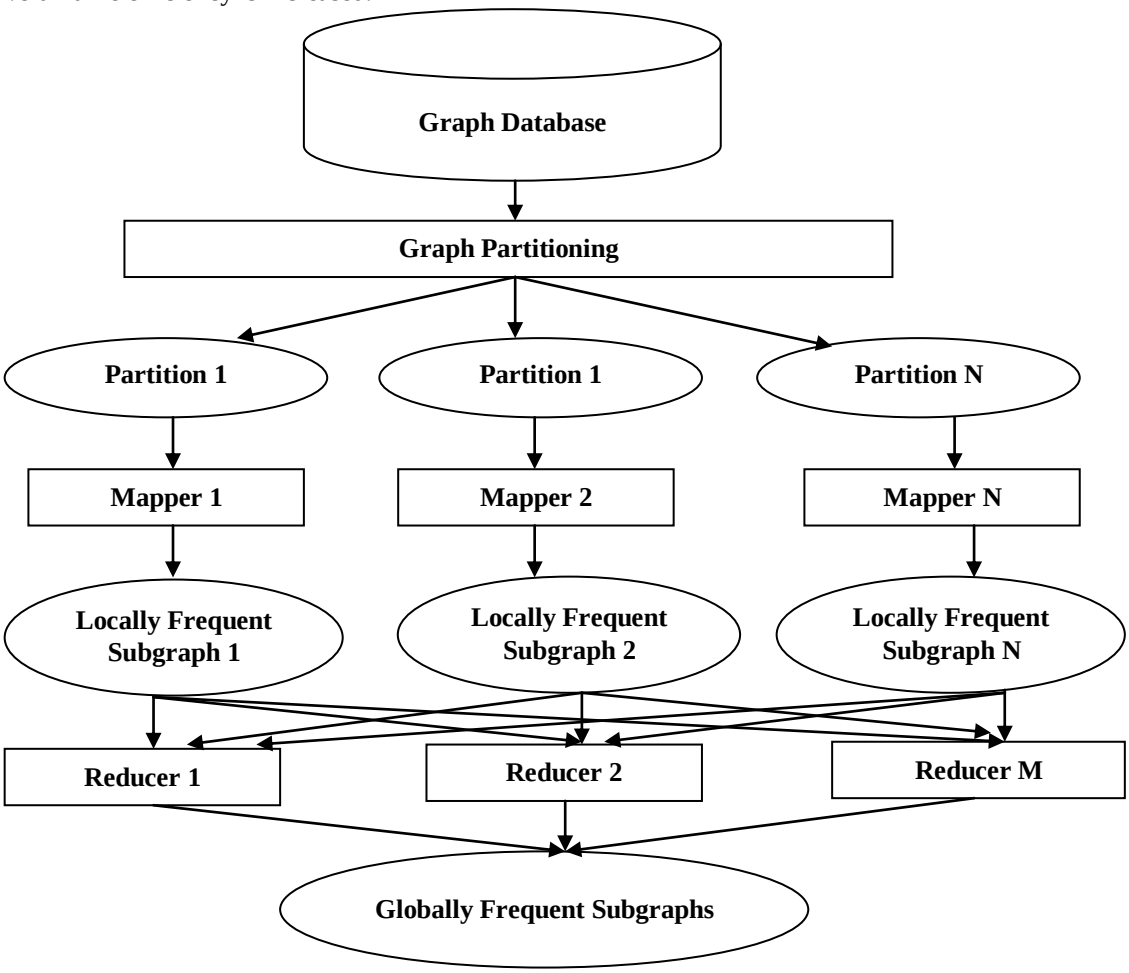


Fig 3.1 FSM Algorithm

V. RESEARCH METHODOLOGY

A. FSM Introduction

Graphs show up in diverse set of disciplines, ranging from computer networks, social networks to bioinformatics, chemo informatics and others. Finding recurrent and frequent substructures may give important insights on the data under consideration. These substructures may correspond to important functional fragments in proteins such as active sites, feature positions, junction sites. Mining these substructures from data in a graph perspective falls in the field of graph mining and more specifically in frequent subgraph mining.

Frequent subgraph mining is a main task in the area of graph mining and it has attracted much interest. Consequently, several subgraph mining algorithms have been developed. However, existing approaches are mainly used on centralized computing systems and evaluated on relatively small databases nowadays, there is an exponential growth in both the graph size and the number of graphs in databases, which makes the above cited approaches face the scalability issue. In this context, several distributed solutions have been proposed nevertheless, the data distribution techniques adopted by these works does not include data characteristics. Consequently, these techniques may face scalability problems such as load balancing problems. To overcome this obstacle, a data partitioning technique that considers data characteristics should be applied. In this study propose a scalable and distributed approach for large scale frequent subgraph mining based on MapReduce framework.

The proposed approach offers the possibility to apply any of the known subgraph mining algorithms in a distributed way. In addition, it allows many partitioning techniques for the graph database. In research work, consider two instances of data partitioning:

- The default partitioning method proposed by MapReduce framework
- A density-based partitioning technique.
- The second partitioning technique allows a balanced computational load over the distributed collection of machines.

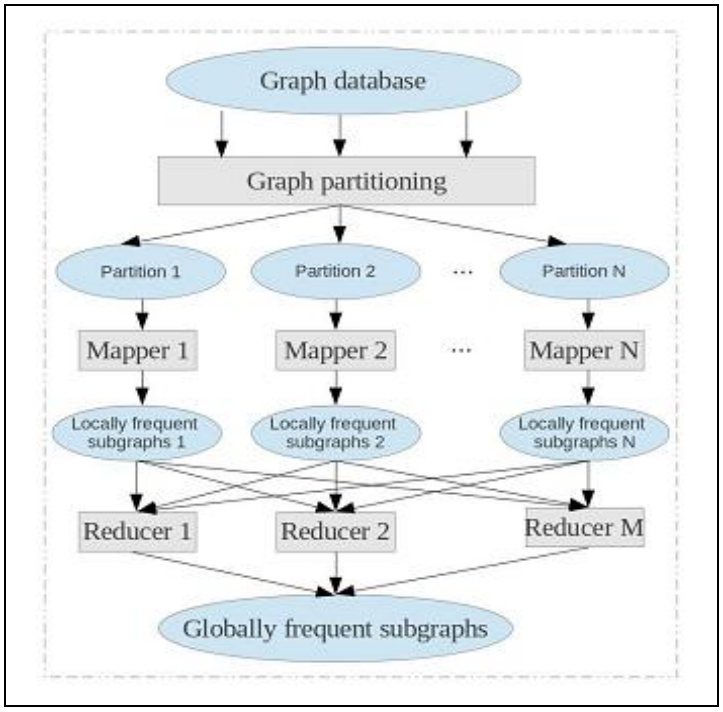


Fig 4.1 Graph Structures

A task of major interest in this setting is frequent subgraph mining (FSM) with respect to a minimum support threshold. There are two separate problem formulations for FSM : graph transaction based FSM and single graph based FSM. In graph transaction based FSM, the input data comprises a collection of medium-size graphs called transactions. In single graph based FSM, the input data, as the name implies, comprises one very large graph. In this work, we are interested in large scale graph transaction based FSM.

B. Mapreduce

MapReduce is a programming model that enables distributed computation over massive data. The model provides two abstract functions: map, and reduce. Map corresponds to the "map" function and reduce corresponds to the "fold" function in functional programming. Based on its role, a worker node in MapReduce is called a mapper or a reducer. A mapper takes a collection if (key, value) pairs and applies the map function on each of the pairs to generate an arbitrary number of (key,value) pairs as intermediate output. The reducer aggregates all the values that have the same key in a sorted list, and applies the reduce function on that list. It also writes the output to the output file

Iterative MapReduce: Iterative MapReduce can be defined as a multi staged execution of map and reduce function pair in a cyclic fashion, i.e. the output of the stage i reducers is used as an input of the stage i + 1 mappers. An external condition decides the termination of the job. Pseudo code for iterative MapReduce algorithm is presented in Figure 4.2.

Algorithm: Iterative MapReduce():
While(Condition)
Execute MapReduce Job
Write result to DFS
Update condition
// G is the database
// k is initialize to 1
Mining Frequent Subgraph(G, minsup):
Populate F1
while Fk 6= ∅
Ck+1 = **Candidate generation**(Fk, G)
forall c ∈ Ck+1
if isomorphism checking(c) = **true**
support counting(c, G)
if c.sup ≥ minsup
Fk+1 = Fk+1 S{c} k = k + 1
return Si=1···k−1 Fi

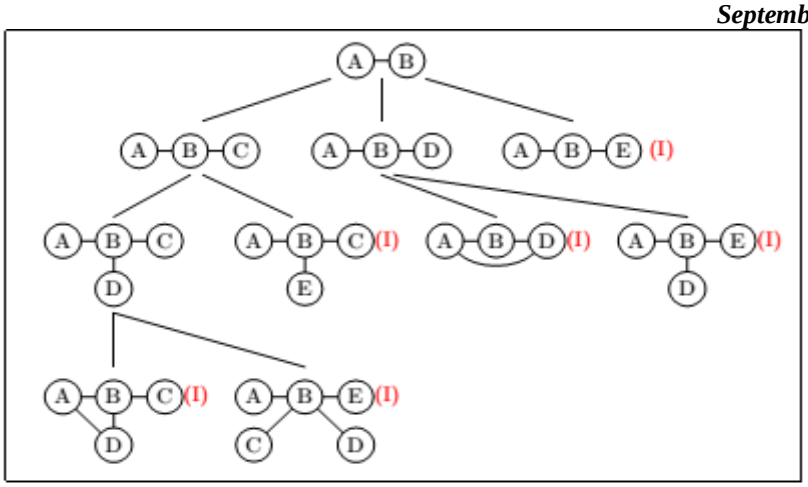


Fig 4.2 Map Reduces Mining

In this paradigm, the mining task starts with frequent patterns of size one (single edge patterns), denoted as F1 (Line 0). Then in each of the iterations of the while loop (Line 1-6), the method progressively finds F2, F3 and so on until the entire frequent pattern set (F) is obtained. If Fk is non-empty at the end of an iteration of the above while loop, from each of the frequent patterns in Fk the mining method creates possible candidate frequent patterns of size k+1 (Line 2). These candidate patterns are represented as the set C. For each of the candidate patterns, the mining method computes the pattern's support against the dataset G (Line 5). If the support is higher than the minimum support threshold (minsup), the given pattern is frequent, and is stored in the set Fk+1 (Line 6). Before support counting, the method also ensures that different isomorphic forms of a unique candidate patterns are unified and only one such copy is processed by the algorithm (Line 4). Once all the frequent patterns of size k + 1 are obtained, the while loop in Line 1 to 7 continues.

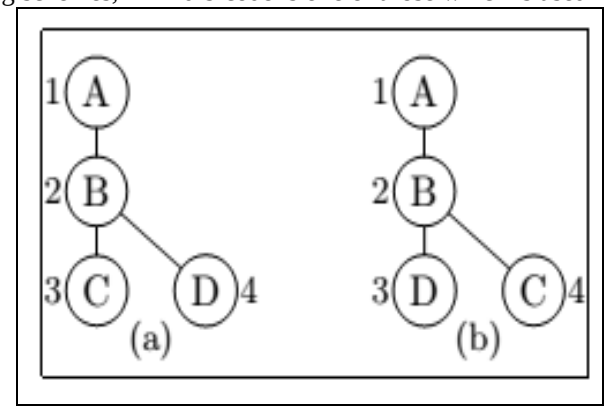
Thus each iteration of the while loop obtains the set of frequent patterns of a fixed size, and the process continues until all the frequent patterns are obtained. In Line 8, the FSM algorithm returns the union of Fi : $1 \leq i \leq k - 1$.

C. Isomorphism Checking

Given a frequent pattern of size k, this step adjoins a frequent edge (which belongs to F1) with c to obtain a candidate pattern d of size k + 1. If d contains an additional vertex then the added edge is called a forward edge, otherwise it is called a back edge; the latter simply connects two of the existing vertices of c. Additional vertex of a forward edge is given an integer id, which is the largest integer id following the ids of the existing vertices of c; thus the vertex-id stands for the order in which the forward edges are adjoined while building a candidate pattern. In graph mining terminology, c is called the parent of d, and d is a child of c, and based on this parent-child relationship we can arrange the set of candidate patterns of a mining task in a candidate generation tree.

FSM algorithms also impose restriction on the extension nodes of the parent pattern so that redundant generation paths can be reduced. One such restriction that is used in the popular gSpan algorithm is called rightmost path generation that allows adjoining edges only with vertices on the rightmost path. Simple put, "right most vertex" (RMV) is the vertex with the largest id in a candidate subgraph and "right most path" (RMP) is the shortest path from the lowest id vertex to the RMV strictly following forward edges.

Mention the previous paragraph, a candidate pattern can be generated from multiple generation paths, but only one such path is explored during the candidate generation step and the remaining paths are identified and subsequently ignored. To identify invalid candidate generation paths, a graph mining algorithm needs to solve the graph isomorphism task, as the duplicate copies of a candidate patterns are isomorphic to each other. A well-known method for identifying graph isomorphism is to use canonical coding scheme, which serializes the edges of a graph using a prescribed order and generates a string such that all isomorphic graphs will generate the same string. There are many different canonical coding schemes, min- dfs-code is one of those which is used in .



Pattern	Occerence List (OL)
A–B	1 : [(1, 2)] ; 2 : [(1, 2)]
B–D	1 : [(2, 4)] ; 2 : [(2, 3)] ; 3 : [(1, 2)]
B–E	2 : [(2, 5)] ; 3 : [(1, 3)]
A–B–D	1 : [(1, 2), (2, 4)] ; 2 : [(1, 2), (2, 3)]
A–B–E	2 : [(1, 2), (2, 5)]

Fig 4.3 Isomorphism Graph

Mapper FSG(Fkphx.min-dfs-code, x.obji):
Ck+1 = **Candidate generation**(Fkp)
forall c ∈ Ck+1
if isomorphism checking(c) = **true**
populate occurrence List(c**)**
if length(c.occurrence List) > 0
emit (c.min-dfs-code , c.obj)

In this study present a brief overview of the current status and future directions of frequent pattern mining. There are various inter-disciplinary domains like chemo informatics, bioinformatics etc. where mining of recurrent patterns across large collection of networks is required. Due to increasing size and complexity of patterns in there is a need for efficient graph mining algorithm. With over a decade of extensive research, there have been hundreds of research publications and tremendous research, development and application activities in this domain.

Many algorithms for frequent subgraph mining have been proposed so far. Most of the algorithms, they focus only on a static set of graphs. Very few algorithms are for mining patterns from dynamic set of graphs. Also all the algorithms proposed so far, outperform each other, either in terms of memory requirements or in terms of few orders of magnitude of computation time. None of them completely address the issue of NP-completeness of the subgraph mining problem.

VI. RESULTS AND DISCUSSION

The following Table 5.1 describes experimental result for existing and proposed system analysis. The table contains Graph id, Number of node, Map Reduces count Node and average frequent sub graph mapping sub graph details are shown.

Table 5.1 Frequent Sub-Graph Mining Performances Analysis (Weight of Node)

S.NO	Graph ID	Number of Node	Map Reduces [FSM] [Count]	Map Reduces Parallel-FSM [Count]
1	G1	15	11	9
2	G2	30	24	20
3	G3	45	33	29
4	G4	60	46	35
5	G5	75	58	46
6	G6	90	69	58
7	G7	105	81	72
8	G8	120	92	80
9	G9	135	106	88
10	G10	150	118	97

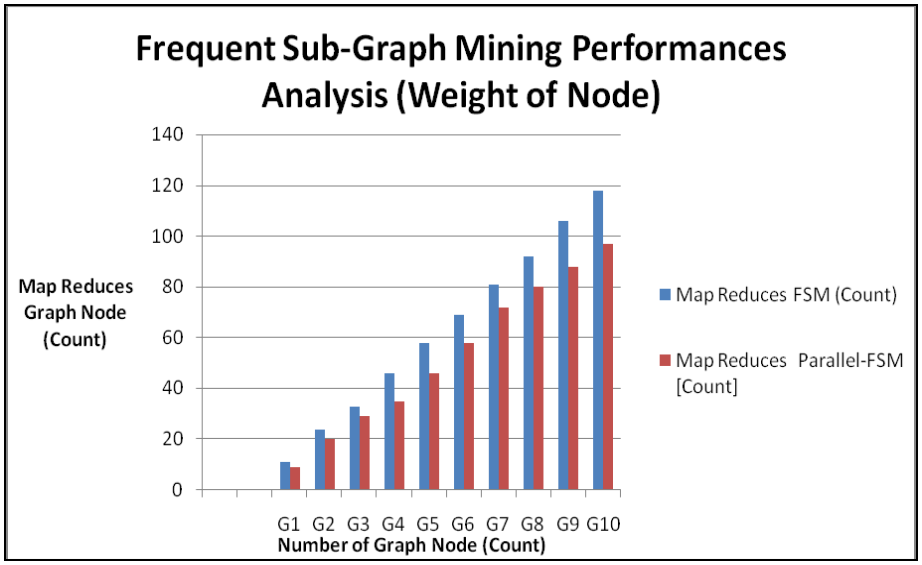


Fig5.1 Frequent Sub-Graph Mining Performances Analysis (Weight of Node)

VII. CONCLUSION

The thesis achieves solving the task of frequent subgraph mining on a distributed platform like MapReduce is challenging for various reasons. An FSM method is proposed which computes the support of a candidate subgraph pattern over the entire set of input graphs from a set of graphs (Graph Database). 'N' number of nodes is given with their

capabilities. Then the graph details with vertex count are also given. Then graph with minimum vertex count and maximum vertex are found. Then the difference between maximum and minimum is also found out. Then all the groups are grouped such that a) minimum vertex to minimum vertex + 1/3rd of difference '$G^{a'}$, b) minimum vertex + 1/3rd of difference to minimum vertex + 2/3rd of difference '$G^{b'}$ and c) the remaining '$G^{c'}$.

Then the nodes are classified as 1) low capability are assigned with 'G^{a}' graphs, 2) medium apability with 'G^{b}' graphs and high capability with 'G^{c}' graphs. Thus the project presented a novel iterative MapReduce based frequent subgraph mining algorithm, called FSM-H. The proposed system shows the performance of FSM-H over numerous graph records. This project shows that FSM-H is significantly better than the existing method.

- In future, more number of systems, nodes can be allocated for frequent sub graph mining and the time reduction can be checked with current proposed system.
- In addition, graph of different sizes may be given to single system and can be checked. This may eliminate the transfer of vertex and edge details found in isomorphic graphs between two systems since both the sub graphs are mined in single system.

REFERENCES

[1] G. Liu,M. Zhang, and F. Yan, "Large-scale social network analysis based on Mapreduce," in Proc. Int. Conf. Comput. Aspects Soc. Netw., 2010, pp. 487–490.

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, pp. 107–113, 2008.

[3] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A petascale graph mining system implementation and observations," in Proc. 9th IEEE Int. Conf. Data Mining, 2009, pp. 229–238.

[4] U. Kang, B. Meeder, and C. Faloutsos, "Spectral analysis for billion-scale graphs: Discoveries and implementation," in Proc.15th Pacific-Asia Conf. Adv. Knowl. Discov. Data Mining,2011, pp. 13–25.

[5] S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," in Proc. 20th Int. Conf. World Wide Web, 2011, pp. 607–614.

[6] R. Pagh and C. E. Tsourakakis, "Colorful triangle counting and a mapreduce implementation," Inf. Process. Lett., vol. 112, no. 7, pp. 277–281, 2012.

[7] F. Afrati, D. Fotakis, and J. Ullman, "Enumerating subgraph instances using map-reduce," in Proc. IEEE 29th Int. Conf. Data Eng., Apr. 2013, pp. 62–73.

[8] B. ahmani, R. Kumar, and S. Vassilvitskii, "Densest subgraph in streaming and mapreduce," Proc. Very Large Data Bases Endow., vol. 5, no. 5, pp. 454–465, Jan. 2012.

[9] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, and J. Prins, "Mining protein family specific residue packing patterns from protein structure graphs," in Proc. Int. Conf. Res. Comput. Mol. Biol., 2004, pp. 308–315

[10] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in Proc. 20th Int. Conf. Very Large Data Bases, 1994, pp. 487–499.

[11] S. Nijssen, and J. Kok, "A quickstart in frequent structure mining can make a difference," in Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discov. DataMining, 2004, pp. 647–652.

[12] V. Chaoji, M. Hasan, S. Salem, and M. Zaki, "An integrated, generic approach to pattern mining: Data mining template library," Data Min. Knowl. Discov. J., vol. 17, no. 3, pp. 457–495,2008.

[13] B. Srichandan and R. Sunderraman, "Oo-FSG: An object-oriented approach to mine frequent subgraphs," in Proc. Australasian Data Mining Conf., 2011, pp. 221–228

[14] D. J. Cook, L. B. Holder, G. Galal, and R. Maglothin, "Approaches to parallel graph-based knowledge discovery," J. Parallel Distrib. Comput., vol. 61, pp. 427–446, 2001.

[15] C.Wang and S. Parthasarathy, "Parallel algorithms for mining frequent structural motifs in scientific data," in Proc. 18th Annu. Int. Conf. Supercomput., 2004, pp. 31–40.

