gSpan-H: An Iterative MapReduce Based Frequent Subgraph Mining Algorithm

Mrs.M.H.Sangle[1], Prof.S.A.Bhavsar[2]

[1] Student Computer Department, M.C.O.E.R.C, Maharashtra, India
[2] Assistant Lecturer, Computer Department, M.C.O.E.R.C, Maharashtra, India

ABSTRACT

In data mining applications, mining frequent subgraph from a large number of small graphs is an important operation. For extracting frequent subgraphs many algorithms have been proposed. But now a days, graph data grows both in size and quantity, therefore existing methods cannot extract frequent subgraph on a centralized machine. To overcome this some distributed solution using MapReduce is becoming important paradigm for computation on massive data. In experimented work, we investigate how to efficiently perform extraction of frequent subgraph over a large datasets using MapReduce. We propose a frequent subgraph algorithm called as gSpan-H which is iterative MapReduce based framework. This algorithm uses breadth first search strategy. This algorithm is isomorphism testing free approach for efficiently mine frequent subgraph. Our experiments with real life and large synthetic datasets validate the effectiveness of gSpan-H for mining frequent subgraphs from large distributed datasets.

Keyword: - *Iterative MapReduce, Graph mining, Frequent subgraph mining, gSpan-H*

1. INTRODUCTION

AS of late, graph mining has turn out to be very popular research area and has surround a large number of investigation and application domains like data mining, computational biology, e-commerce, web mining, environmental sciences and social network analysis. FSM is a very significant research area in graph mining [9, 10] for extracting novel insights, because of its extensive range of applications in the above domains. Frequent patterns help to identify different relations. For example, in a protein-protein interaction (PPI) network, a frequent pattern could uncover unknown collaboration of a protein [11]. Also, within social networks, a frequent pattern could show a friend group. For frequent subgraph mining many methods are discovered, however these traditional methods for data investigation and mining are not designed to handle massive dataset. So, in recent years these methods are re-considered and re-implemented to handle big dataset. To analyze the massive dataset, the MapReduce [4] approach of distributed computing has been the most successful. MapReduce approach does distributed computing and it uses a Distributed File System for storing the data. Due to this it improves the input output performance while computing massive dataset. Utilizing MapReduce framework considerable investigation is done on graph data, but mining frequent subgraphs from graph database has gotten the slightest consideration. Because the development of frequent subgraph mining (FSM) in various domains like cheminformatics [12], social networks, bioinformatics [11], and semantic web [13], a versatile technique for FSM using MapReduce is of high demand. In this experimentation gSpan-H i.e. graph based sub-structure pattern mining using Hadoop is proposed, a distributed FSM technique using MapReduce is stated. gSpan-H is a isomorphism testing free approach and it uses breadth-first-approach. gSpan-H discovers frequent subgraph without candidate generation instead that algorithm builds new lexicographic order among graphs, then it maps each graph to unique DFS code as its canonical label. gSpan-H computes frequent subgraph based on this lexicographic order. gSpan-H generates a complete set of frequent subgraphs, by giving a graph database, and a minsup (minimum support) threshold. In a partition, all the patterns which are having non-zero support constructs and keeps, to guarantee completeness of gSpan-H technique within the map phase of the mining. Then in the reduce phase, it combines its support which is calculated from all partitions from different computing nodes and from this support it selects whether a pattern is frequent. FSMH method runs in an iterative manner [1] i.e. output of i-1 reducer's iteration is given as an input to the input of mapper of iteration i. In mapper of iteration i phase computes candidate subgraph of size i. Here size of graph is considering as the number of edges it having. Aggregating the local supports of all reducers of iteration i, it finds true frequent subgraphs. The data processed in subsequent iteration is also writes on HDFS.

2. LITERATURE SURVEY

2.1 In- memory version of frequent subgraph mining method

Traditional methods solves only in-memory version of FSM task and for this many methods are developed. Most important amongst them are AGM [13], FSG [3], gSpan [4], Gaston [5], and DMTL [6]. All these techniques are able to process just small amount of data. And mines this data in reasonable amount of time and assume that mining task fit in main memory of a computer. These methods fail if data grows substantially. Some methods such as, DB-Subdue [7], and DB-FSG [8] and OOFSG [9] consider large database scenario.

2.2 Frequent subgraph mining designed using Shared memory parallel algorithms

Researchers investigate and consider shared memory to deal with scalability problem of graph datasets. Cook et al. proposed frequent subgraph mining algorithms parallel version Subdue [14]. A parallel toolkit [15] for their Motif-Miner [16] algorithm is developed by Wang et al. The Parmol [17] Software created by Meinl et al includes parallel implementation of Mofa [18], gSpan [4], FFSG [19] and Gaston [5]. ParSeMis[20] tool also provides parallel implementation of gSpan algorithm. There are couples of important works, Part- Miner [21] and PartGraphMining [22], to deal with scalability problem due to the size of input graph. In this methods graph data is partitioning for the processed.

2.2 Frequent subgraph mining using MapReduce framework

To overcome scalability problem of input graph data MapReduce framework is used for mining frequent subgraph. For the mining frequent subgraph from substantial gathering of graph datasets Gabriel Ghinita [23] proposed MapReduce based two-step method i.e. filter and refinement method which is appropriate to very big parallelization inside the scalable MapReduce processing model. Set of graphs partition among worker nodes. Filter step applies on each worker. Then it determines a set of candidate subgraphs that are locally frequent in its partition. The input of the refinement step is the union of all such. Against all partitions, every candidate is checked and only the globally frequent graphs are saved. This method can improve efficiency, diminishes communication cost with low computational overhead. From the large network Suri and Vassilvitskii [24] and Pagh and Tsourakakis [25] finds the substructures using MapReduce. The proposed algorithm is sequential triangle counting algorithm. This method applies black box method like any triangle counting algorithm and share out the computation across various machines. Count the triangles using MapReduce from graph dataset are the main aim of this work. In [26], the authors uses MapReduce framework for frequent subgraph mining; but, their method is unproductive as a result of various shortcomings. In this mechanism it cannot avoid generating duplicate patterns. Due to this size of candidate subgraph space increases exponentially. Also this method contains duplicate copy of graph patterns. Another problem with this method is that, number of iterations required must be specified by user. Method cannot specify how to determine total number of iterations count accordingly that the algorithm is capable to get all frequent patterns for a given support. There exist some works proposed by Lin [27] and Cheng [28] that mine subgraphs from a single large graph and it also consider frequent considering their induced occurrences. However, there objective is different than FSM-H and gSpan-H. The method proposed in [1] for frequent subgraph over a distributed graph data is FSM-H. FSM-H i.e. Frequent Subgraph Mining using Hadoop, a distributed FSM technique using MapReduce is stated. FSM-H generates a complete set of frequent subgraphs, by giving a graph database, and a minsup (minimum support) threshold. In FSM-H method mapper generates candidate subgraph and then perform isomorphism checking. If candidate subgraph passes this test then on reducer it checks its support and if support greater than minsup then pattern is frequent; otherwise it is infrequent. But this algorithm suffers from two additional costs:

1. **Costly Subgraph isomorphism test:** Since subgraph isomorphism is an NP-complete problem, no polynomial algorithm can solve it. Thus testing of false candidate degrades the performance a lot.
2. **Costly candidate generation:** The generation of size (k+1) subgraph candidates from size k frequent subgraphs is more complicated and costly.

The above discussed method had a common problem where all these methods cannot mine the complete frequent patterns from large graph datasets.

3 BACKGROUND

3.1. Frequent subgraph mining

Let, G = {G1, G2… Gn} be a graph database. Every $G_i \in G$, i= {1...n} stands for directed, labeled, connected and simple (i.e. only one edge between pair of vertices) graph. Size of graph g is defined as the total number of edges it contains. Support of graph G is defined as number of times the subgraph g appears in the graph database G. Then the subgraph g said to be frequent if support $\geq \prod_{min}$, where $\prod_{min}$ is user-specified or predefined minimum support (minsup) threshold. FSM i.e. frequent subgraph mining is define as, the method of discovering all subgraph that appear frequently in a database with respect to a given database according and given frequency threshold. F represents the set of frequent patterns and it partitions into a number of disjoint sets according to the size of frequent patterns.Fi represents frequent patterns of size i.

Example Consider a graph database as shown in fig 1a.with tree vertex labeled graphs (G1, G2 and G3). If $\prod_{min} = 2$, then there are 13 frequent subgraphs are possible as shown in Fig. 1b.

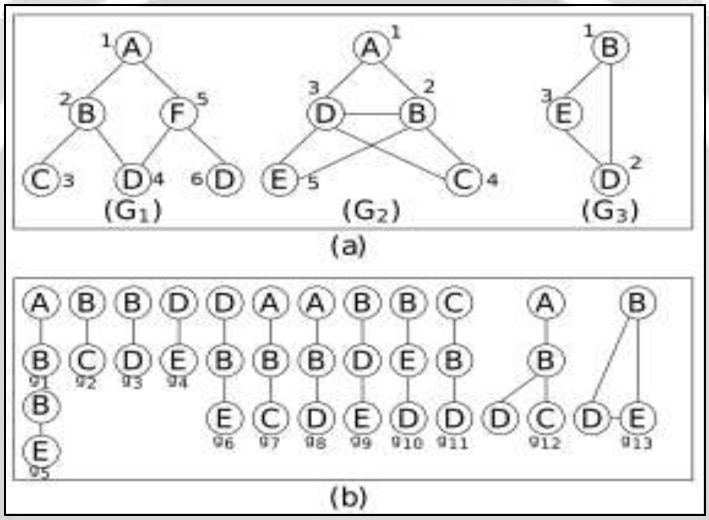


Figure 1 (a) Graph database G having three graphs with labeled vertices (b) 13 Frequent subgraphs of fig (a) having minsup = 2.

3.2. MapReduce

Google proposed native framework called MapReduce for distributive data analysis. It is a programming model. Due to this model it is enables distributed computations on big datasets [29]. It is an execution framework for large-scale data processing. The MapReduce model provides two functions: map, and reduce. According to its job, a worker node in MapReduce is called a mapper or a reducer.

- **Map Function:** For the map function input is given as a collection of (key, value) pairs. On each pairs apply map function. It generates an arbitrary number of (key, value) pairs as a intermediate results.
- **Reduce Function:** The reducer worker node collects the values having same key in a sorted manner, then applies the reduce function on that list. Output is written by reducer in output file. A distributed file system managed all the input and output files of MapReduce. Fig.2 shows the model of MapReduce framework. Hadoop is an open source implementation of MapReduce programming model written in Java language.

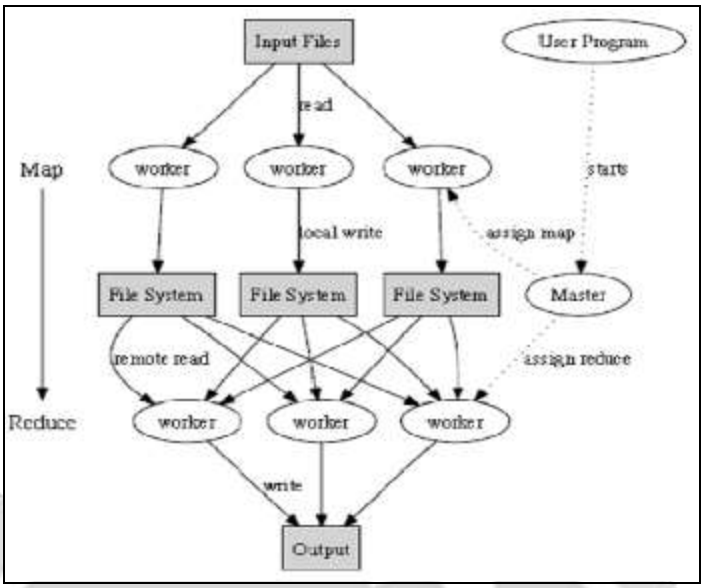


Figure 2 MapReduce Framework

3.3. Iterative MapReduce

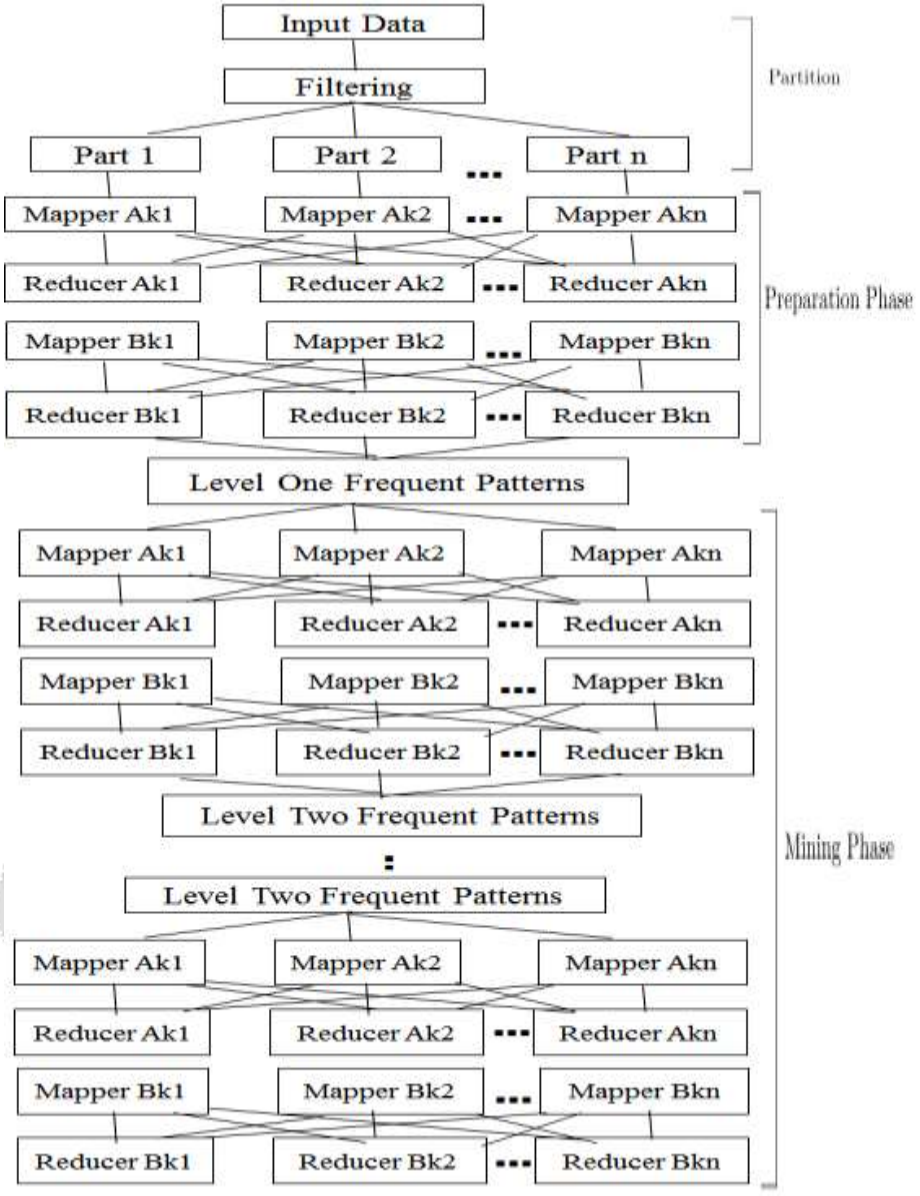
Multi staged implementation of map and reduce function pair in a cyclic manner is called as Iterative MapReduce [1], i.e. output of the previous iteration serves as input to the next iteration. Input for the stage i +1 mappers is the output of the stage i reducer in the iterative MapReduce. Termination of the job decides by an external condition.

4 SYSTEM ARCHITECTURE

4.1. Framework of gSpan-H

Figure 3 shows flow diagram of phases of gSpan-H algorithm. gSpan-H has following three important phases:

1) **Data partition**
   In this phase gSpan-H generates the partition of input dataset and also omits the infrequent edges from input graph dataset. gSpan-H uses one straightforward method for partition in which each partition processes equal number of graphs.
2) **Preparation Phase**
   In this phase mapper generates some partition specific data structures. Each partition generates distinct copy of data structure. They are static for a partition in the sense that they are same for all patterns generated from a partition. Preparation phase finds all possible frequent subgraph of size one and writes on HDFS.
3) **Mining Phase**
   In this phase, mining process finds all possible frequent subgraphs through iteration. Each of the mappers of the ongoing iteration reads the chunk written in HDFS and perform mining task.



4.2. Algorithm For gSpan-H

In this section describes process of subgraph construction and finding frequent subgraph from them using two heterogeneous MapReduce jobs per iteration. Below algorithm 1 and 2 shows the pseudo code of mapper and reducer respectively. The first job (denoted as Ak) constructs size-k subgraphs from size-(k - 1) subgraphs, while the second job (denoted as Bk) will check whether or not a subgraph meets the user defined support. The algorithm starts with single edges, and runs until there are no longer any frequent subgraphs constructed. Algorithms 1 and 2 highlight the tasks of Ak Mapper and Reducer respectively and Algorithms 3 and 4 highlight the tasks of Bk Mapper and Reducer respectively.

Algorithm 1 Algorithm for Mapper A_k of distributed gSpan algorithm

Input: The input key is a graph ID, and the input value is F_P^k represents frequent size i subgraph. The mapper reads it from Hadoop Distributed File Systems(HDFS)

Output: key-value pair, such as (c.min-dfs-code, c.obj)

1: Sort labels of the vertices and edges in Fp^k by their frequency.

2: Remove infrequent vertices and edges.

3: Relabel the remaining vertices and edges in descending frequency.
4: S1 all frequent 1 edge graphs in $F_p k$
5: sort S1 in DFS lexicographic order

Algorithm 2 Algorithm for Reduce A_k of distributed gSpan algorithm

Input: F_p^k represents the set of size-k frequent subgraphs having non-zero support in a specific partition p.
Output: key-value pair, such as (c.min-dfs-code, c.obj)
1: emit data for mapper of second MapReduce job in same iteration

Algorithm 3 Algorithm for Mapper B_k of distributed gSpan algorithm

Input: F_p^k represents the set of size-k frequent subgraphs having non-zero support in a specific partition p.
Output: key-value pair, such as (c.min-dfs-code, c.obj)
1: for all edge e ϵ S^1 do
2: For each 1-edge frequent subgraph Mining grows all nodes in the subtree rooted at this 1-edge graph.
3: end for
4: Shrink each graph in the graph set F_p^k by removing the edge after all descendants of this 1-edge graph have been searched.

Algorithm 4 Algorithm for Reducer B_k of distributed gSpan algorithm

Input: F_p^k represents the set of size-k frequent subgraphs having non-zero support in a specific partition p.
Output: key-value pair, such as (c.min-dfs-code, c.obj)
1: if support ≥ minsup then
2: emit subgraph as frequent
3: end if

5 RESULTS AND DISCUSSION

In this section, we demonstrate the performance of gSpan-H for solving frequent subgraph mining task on graph datasets by giving experimental results. For input, we use DBLP dataset. In this experimentation, we compare execution runtime for gSpan-H with FSM-H [1] algorithm considering three parameters support, different dataset sizes and number of Reducer.

5.1. Runtime of gSpan-H for Different Minimum Support

In this experiment, we analyze the runtime of FSM-H and gSpan-H for varying minimum support threshold. We conduct this experiment for the real world dataset DBLP. Here use 2K input file and x the number of mapper and reducer to 4 and keep track of the running time of FSM-H and gSpan-H for minimum support thresholds that vary between 10 to 40 percent. In Figs. 4 show the result. As expected, the runtime decreases as minimum support threshold increases.

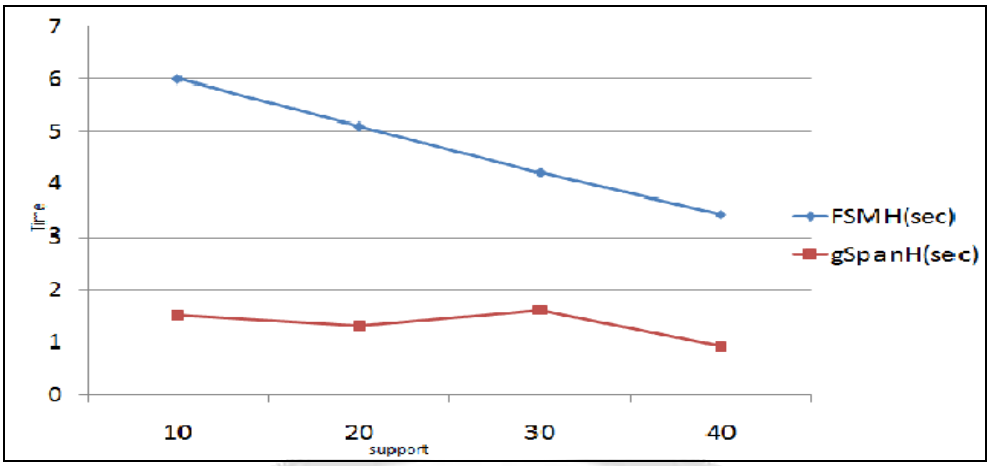


Figure 3 Line plot showing the relationship between the minsup threshold (percent) and the running time (in Sec)

5.2. Runtime of gSpan-H for Different sizes of graph dataset

Fig 7.3 shows relationship between runtime for different sizes of datasets. In this experiment we use four synthetic datasets each having 1K to 4K input graphs. Fig 7.3 shows the runtime of FSM-H and gSpan-H for these datasets for 20 percent minimum support threshold. As we can see, the overall running time increases at a sub-linear rate as the number of graphs in the database increases. This shows the scalability of FSM-H and gSpan-H.

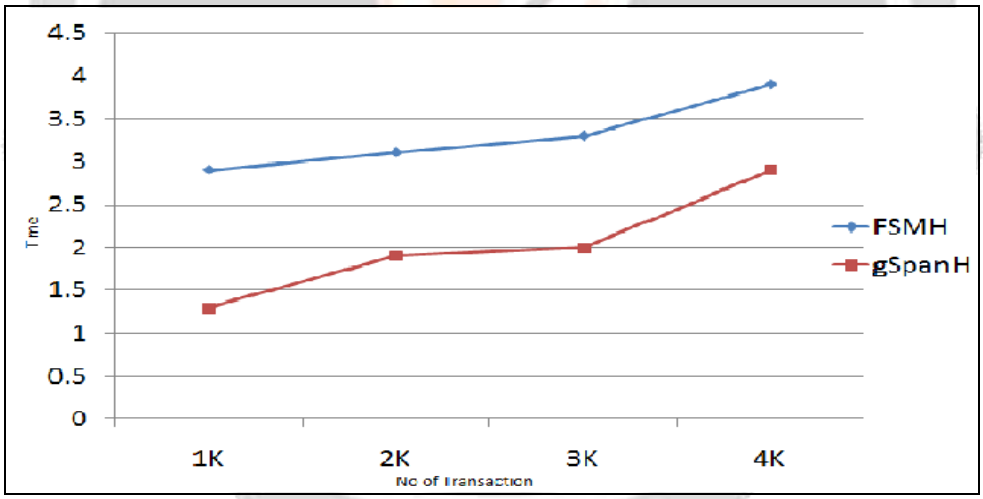


Figure 4 Line Plot showing the relationship between the different sizes(in K) and running time(in sec)

5.3. Runtime of gSpan-H for Varying Number of Reducer

In this experiment, measure the runtime of algorithms for various configurations such as, 1, 2, 3 and 4 reducers. For experiment use, we use DBLP dataset with 20 percent minimum support threshold. Figs. 7.2 show the relationship between execution time and the number of reducers using line plot.

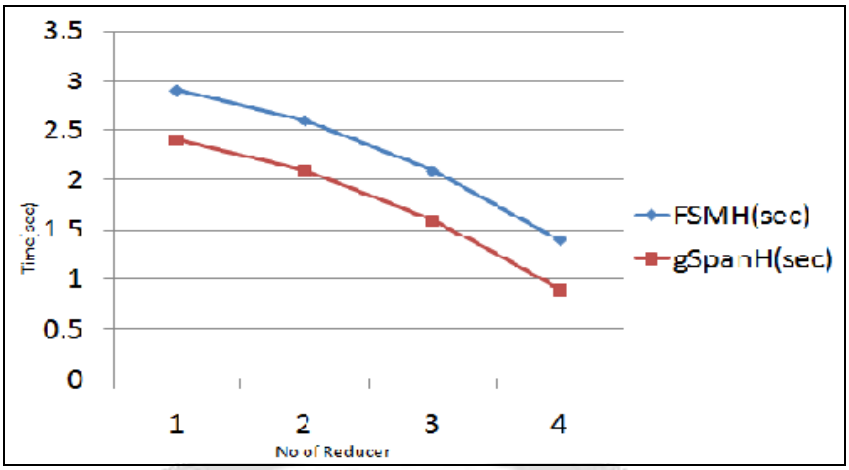


Figure 5 Line plot showing the relationship between the number of Reducer and the running time (in Sec)

6. REFERENCES

[1] Mansurul A. Bhuiyan and Mohammad Al Hasan "An Iterative MapReduce Based Frequent Subgraph Mining Algorithm" Ieee transactions on knowledge and data engineer-ing, vol. 27, no. 3, march 2015

[2] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in Proc. Int. Conf. Data Mining, 2001, pp. 313-320.

[3] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in Proc. Int. Conf. Data Min., 2002, pp. 721-724

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

[5] 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.

[6] S. Chakravarthy and S. Pradhan, "Db-FSG: An SQL-based approach for frequent subgraph mining," in Proc. 19th Int. Conf. Database Expert Syst. Appl., 2008, pp. 684-692.

[7] 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.

[8] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 30-36. AAAI Press, August 1998.

[9] L. B. Holder, D. J. Cook, and S. Djoko. Substucture discovery in the subdue system. InWorkshop on Knowledge Discovery in Databases (KDD) Proceedings, pages 169-180. AAAI Workshop, July 1994

[10] 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.

[11] S. Kramer, L. Raedt, and C. Helma,"Molecular feature mining in HIV data," in Proc. Int. Conf. Knowl. Discov. Data Mining, 2004, pp. 136-143.

[12] B. Berendt, "Using and learning semantics in frequent subgraph mining," in Proc. Adv. Web Min. Web Usage Anal., 2006, pp. 18-38.

[13] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in Proc. 4th Eur. Conf. Principles Data Mining Knowl. Discov., 2000, pp. 13-23.

[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.

[16] S. Parthasarathy and M. Coatney, "Efficient discovery of common substructures in macromolecules," in Proc. IEEE Int. Conf. Data Mining, 2002, pp. 362-369.

[17] T. Meinl, M.Worlein, O. Urzova, I. Fischer, and M. Philippsen,"The parmol package for frequent subgraph mining." Electron. Commun. EASST, vol. 1, pp. 1-12, 2006.
[18] C. Borgelt and M. Berthold, "Mining molecular fragments: finding relevant substructures of molecules," in Proc. IEEE Int. Conf. Data Mining, 2002, pp. 51-58.
[19] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in Proc. 3rd IEEE Int. Conf. Data Mining, 2003, pp. 549-552.
[20] M. Philippsen, M.Worlein, A. Dreweke, and T. Werth. (2011). Parsemis- The parallel and sequential mining suite. [Online]. Available:https://www2.cs.fau.de/EN/research/ParSeMiS/index. html
[21] J. Wang, W. Hsu, M. L. Lee, and C. Sheng, "A partition-based approach to graph mining," in Proc. 22nd Int. Conf. Data Eng., 2006, p. 74.
[22] S. N. Nguyen, M. E. Orlowska, and X. Li, "Graph mining based on a data partitioning approach," in Proc. 19th Australasian Database Conf., 2008, pp. 31-37.
[23] X. Xiao,W. Lin, and G. Ghinita, ?Large-scale frequent subgraph mining in Mapreduce,? in Proc. Int. Conf. Data Eng., 2014, pp. 844-855.
[24] 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