Volume 4, Issue 7, July 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Algorithmic Approaches to Graph Mining

Parul Saxena
M.Tech, Dept of CSE,
Graphic Era University
Dehradun, India

Bhaskar Pant
Assistant Professor, Dept of CSE,
Graphic Era University
Dehradun, India

Abstract: Graph mining being an important research field of data mining has many interesting algorithms. Every new approach is studied in detail, then implemented and evaluated on various benchmarks. The focus of the study revolves around the Frequent Subgraph Mining (FSM) from a given graph data set. This is directed to following categories: (1) Introducing an effective and efficient technique for complete and redundancy-free candidate subgraph generation. (2) Identifying the desired frequent subgraphs for intelligent analysis of various problems. In this paper various categories of algorithms are discussed and compared on various parameters of research issues in the field of frequent subgraph mining. And a new search algorithm is also proposed.

Keywords: Frequent subgraph mining (FSM), DFS code, lexicographic ordering, canonical representations.

I. INTRODUCTION

There has been a recent rise in large number of graphs with huge sizes and complex networks generated for many new applications. These applications can be for social networks, biological studies, wed searches , chemical structures, data management etc. But in all these applications, the main objective of research is to obtain frequent subgraphs from the given data set. The actual idea behind FSM is to generate candidate subgraph and then to determine the frequency of occurrence of these subgraphs also known as the support count. Redundancy checking is also an important part of subgraph mining. Isomorphic subgraphs increase the complexity and thus reducing the efficiency. Detecting isomorphism in subgraph is the bases of frequent subgraphs. Many significant techniques have been proposed with the goal of reducing the computational complexity reduction associated with the subgraph isomorphism testing.

Frequent subgraph mining techniques are most commonly divided into following two categories: (i) Mining based on Apriori principle[], which is basically Breath First Search (BFS) approach of exploring the subgraph structure in the given data set. Its basic procedure is to generate and then test the subgraphs. Thus for generating *k* subgraph, all (*k*-1) subgraphs have to be considered. (ii) Mining based on the growth of patterns in the subgraphs, which adopts a Depth Firth Search (DFS) approach. The procedure is to extend all discovered subgraphs in a recursive manner to obtain all the frequent subgraphs. In this paper we will be focusing on the pattern growth approach of finding frequent subgraphs.

In this paper, a survey of various algorithmic strategies in the field of frequent subgraph mining is laid down and compared according to various benchmarks. To begin with, section 2 introduces some formal terminologies, used and discussed later in the paper. In Section 3, various categories of frequent subgraph mining algorithms are explained in detail. Then section 4, summarizes all the results drawn from the comparisons of the algorithm and a new searching algorithm is proposed. Section 5, concludes the paper with various future scopes of research available.

II. PRELIMINARY TERMINOLOGIES

This section gives a brief idea about the basic methods and terms used in the algorithms discussed further in the paper. So before going into detailed discussion of the algorithms, various techniques and methods of representing graphs and trees are described. The aim of these techniques is to enumerate a graph efficiently to obtain the desired frequent subgaphs.

2.1 Canonical Representations

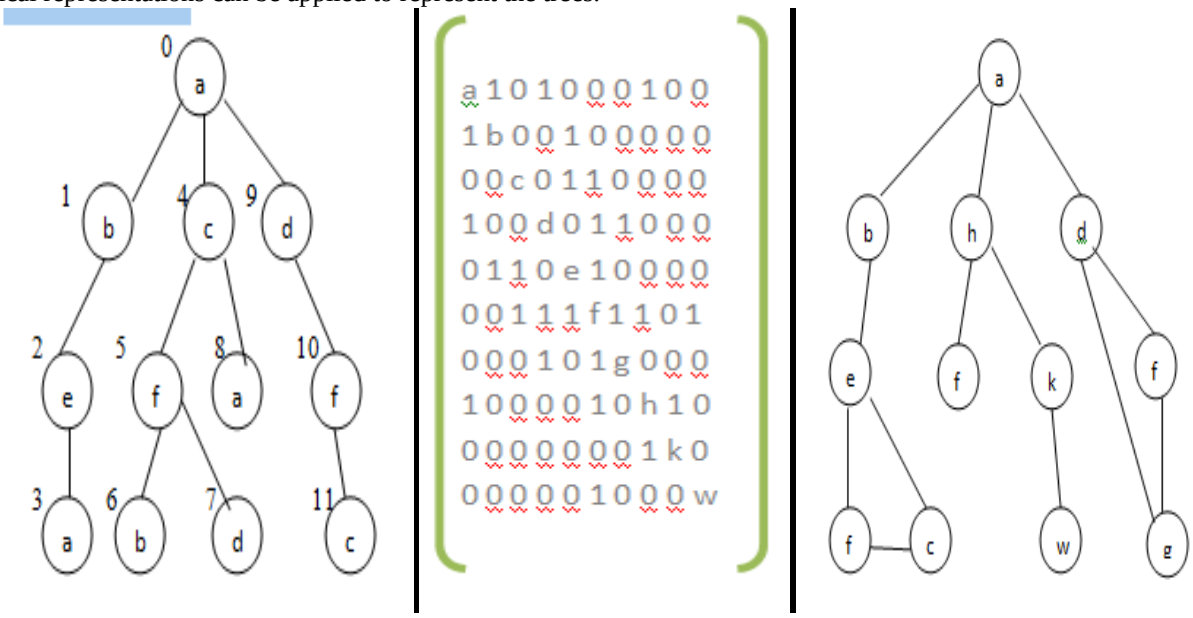
Canonical labelling or representation gives a unique code for a graph. It is very useful for checking the presence of isomorphism in a given pair of graph. Adjacency matrix or adjacency list are a simple way of representing the graph. These associated adjacency matrices of various graphs can be used to produce the minimum lexicographical ordering by combining various rows and columns.

2.1.1 Canonical Adjacency Matrix (CAM):

Suppose an adjacency matrix M of a graph g is given, the coding of M can be obtained by the combination of sequence obtained from concatenating the lower or the upper triangle entries of M, and the diagonal entities. We can have different combinations and permutations of the set of vertexes in different adjacency matrices. So the canonical form of representation is considered to be maximal or minimal encoding. Thus if we get a canonical form an adjacency matrix, it is said to be Canonical Adjacency Matrix (CAM)[]. It is applicable to any simple undirected graph.

2.1.2 Canonical Spanning Tree (CST):

It is much simpler and easier to define a canonical labelling for trees than graphs due to tree's inherent structure. A tree can be spanned and the represented by selecting one vertex or a pair of vertexes as root(s). The procedure is to start with the removal of all leaf vertexes and their incident edges in a recursive manner until a single vertex or two adjacent vertexes are left. So a pair of trees is ordered to obtain a tree with smaller root. After getting rooted unordered trees, any canonical representations can be applied to represent the trees.



2.2 Lexicographical ordering

Here in this section we introduce some techniques to represent and extend graphs efficiently. Mapping of a graph to a DFS code or sequence is done. These codes are ordered through lexicographic ordering. All these techniques are described below.

2.2.1 DFS Subscripting

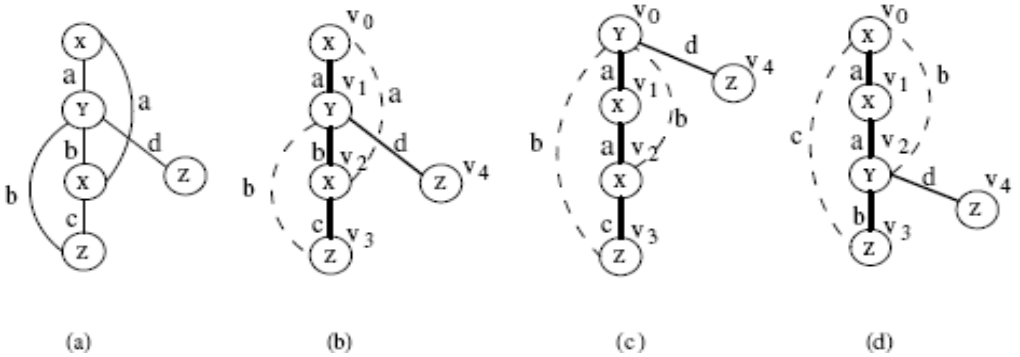
A DFS tree is constructed following a Depth First Search in a graph. Several frequent subgraphs are constructed in the same manner from the dataset. A linear order is formed when the vertices are discovered while a DFS tree is built. According to the discovery of the vertices, their magnitude is decided and hence used in ordering. For example, $i < j$ will mean that *vi* is discovered before *vj*. A graph G is subcripted with a DFS tree T and is written as GT. In a given GT all the edges that are present in the tree are known as the *forward edges*. And once which are in the graph but not in the tree are called as *backward edges*.

2.2.2 DFS Code

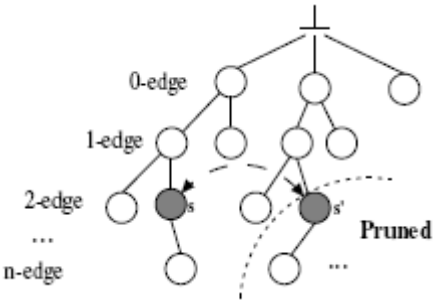
DFS code can be represented by a 5-tuple: (*i*,*j*,*li*,*le*,*lj*). Here *i* and *j* are vertex identifiers and *li* and *lj* are labels for those vertices. And *le* is the label for the edge between the vertices. For every subscripted graph we can have an order in the sequences. The base subscripting is generated and selected in a way that we get minimum sequence generated. We have two kinds of orders for this process. One is edge order, which involves mapping of the edges in a sequence. Another is the sequence order which is according to the sequences in subscript.

2.2.3 Rightmost path extension

It is the most common strategy for candidate generation. If we have *k* frequent subtrees then, $k + 1$ subtrees are generated by adding vertex only to rightmost path of the tree. Every (k +1) subtree is a rightmost expansion of the *k*-subtree. Thus traversing of the subgraphs can be done either in DFS or BFS manner. Algorithms like gSpan use this type of candidate generation.



edge	(Fig 1b) α	(Fig 1c) β	(Fig 1d) γ
0	$(0,1,X,a,Y)$	$(0,1,Y,a,X)$	$(0,1,X,a,X)$
1	$(1,2,Y,b,X)$	$(1,2,X,a,X)$	$(1,2,X,a,Y)$
2	$(2,0,X,a,X)$	$(2,0,X,b,Y)$	$(2,0,Y,b,X)$
3	$(2,3,X,c,Z)$	$(2,3,X,c,Z)$	$(2,3,Y,b,Z)$
4	$(3,1,Z,b,Y)$	$(3,0,Z,b,Y)$	$(3,0,Z,c,X)$
5	$(1,4,Y,d,Z)$	$(0,4,Y,d,Z)$	$(2,4,Y,d,Z)$



III. DISTINCTIVE IDEAS OF SOME FREQUENT SUBGRAPH MINING ALGORITHMS

Subgraph mining is a very challenging area as compared to the frequent itemset mining. Since here arbitrary graph structures are generated and matched in contrast to the matching of bit vectors in frequent itemset mining. In this section, some well known frequent subgraph mining algorithms are discussed and explored. The algorithms to be discussed are "pattern dependent". This means that according to the application, the patterns are to be discovered. This knowledge of the nature of the patterns helps in reduction of search space for any application. Following are some of the interesting algorithms.

gSpan (graph-based Substructure pattern)(Yan and Han, 2002) [6]

The canonical representation used in this algorithm is the dfs-code. The DFS code generated is ordered in lexicographic ordering. And a tree like lattice is generated over all the patterns, which is called DFS code tree. The hierarchical search tree thus generated is traversed in a DFS manner and all the subgraphs with non-minimal DFS codes are pruned to avoid redundant candidate generations. Refined generation of candidate is restricted in gSpan in following two ways: (i) extension of subgraph is done only in rightmost path.(ii) generation is guided by occurrence in the appearance lists only. Explicit subgraph isomorphism testing needs to be done on all the graphs present in the appearance lists. In the formulation of the gSpan algorithm, sparse adjacency list representation is used to store the graph. The subgraph mining stops searching the search space if support of the graph is less than minimum support or its code is not a minimum code. This means that this graph and all its descendants have been generated and discovered before. In a comprehensive performance study conducted in an experiment performed on both synthetic and real world datasets, gSpan outperforms FSG (another algorithm of same category).

FFSM (Fast Frequent Subgraph Mining) (Huan, Wang and prins, 2003) [10]

This algorithm represents graphs as triangle matrices (nodes on diagonal, and edges elsewhere). It adopted the CAM representation which gives a suboptimal tree-like structure known as CAM tree. The code obtained from the matrix is the concatenation of all its entries from left to right and line by line. The isomorphism in a graph can be detected if two graphs have the same canonical code obtained from lexicographic ordering. Joining of two matrices of fragments generates only two new structures. It has a restricted extension method where a new node and edge can only be added to the last node of a canonical adjacency matrix. Generated matrix is pruned if not in an canonical form. FFSM differs from gSpan in another way, as it stores embedding lists for avoiding explicit isomorphism testing on subgraphs. Embedding list stores only the matching nodes and not the edges which results in fast join and extension operation. Various experiments performed on many chemical dataset indicated that FFSM performs better than gSpan. Handling of isomorphic subgraphs problem which is a time consuming step resulted in better performance. And also the algebraic framework reduces the number candidate generated. Thus FFSM outperforms gSpan.

CloseGraph (Yan and Han, 2003) [11]

In this algorithm, intead of mining all the algorithms, only the closed frequent graph sets are mined. A graph g is a closed graph if in all of the database there is no such supergraph of g, that has the same support as g. closeGraph is built on gSpan algorithm and an early termination condition is applied in closed graph mining. Apart from using the Depth First Search and Lexicographic order in generation of frequent graphs, two other important steps involved in the CloseGraph are equivalent occurrence and early termination. These two are for non closed graph mining. A CloseGraph has same basic outline as that of gSpan. But the main improvement is that it uses Early Termination pruning technique before the rightmost path expansion. The basic recursive steps applied in this algorithm are (i) generation of frequent subgraphs (ii) checking for the conditions that satisfies for the graph being a closed graph (iii) applying the pruning method of Early Termination, and checking for its failure cases. Performance study done for this algorithm in several experiments demonstrates that it performs better than gSpan algorithm.

SPIN (Spanning tree based frequent subgraph mining algorithm) (Huan et al. 2004) [12]
This algorithm is uses the concept of canonical spanning tree (CST), to obtain equivalence classes. Maximal frequent subgraphs are to be obtained from the graph dataset. The concept of local and Global maxima is used in this algorithm. A subgraph is locally maximal if does not have have any any frequent supergraphs in its equivalence class with the same canonical spanning tree a globally maximal subgraph is the one which is the maximal frequent in all of thr graph dataset. The overall computational cost is reduced by the pruning steps involved in this algorithm. The basic plot of the algorithm is two phased: First, use of al algorithm to find the frequent subtrees within input graph dataset. Second, isomorphism testing for all frequent subtree. Canonical spanning tree structure is used for this testing. Maximal frequent subgraphs are obtained after the pruning techniques are applied to it. The three pruning techniques applied to the frequent subtree are (i) Bottom-up pruning (ii) Tail shrinking (iii) External-Edge pruning. Experiments performed on chemical dataset and synthetic dataset displays that SPIN performs better than gSpan and FFSM on many levels.

IV. CONCLUSION

With reference to the literature many different mining strategies have been proposed with respect to many different types of graph to produce many different kinds of patterns. Various algorithms have been studied and various techniques related to these algorithms have also been studied. Each algorithm add a missing feature in the graph mining technique to enhance the output. Graph database such as neo4j can be used to display the output of the techniques used. These approaches can be applied to any application. From our study we establish a fact that graph mining is a better way of depicting and using data that is connected in nature. These connected databases can be mined to retrieve various useful patterns. These algorithms are the various ways of achieving such patterns. Future work can involve application of metahueristic approaches in graph mining to achieve better output.

REFERENCES

[1] K. Abedin Tarafder, Shah M. Khaled, M. Ashraful Islam," Reverse Apriori algorithm for frequent pattern mining", *Medwell journals*, 2008, vol 7, issue 12, pg 524-530.
[2] U. Fayyad, G. Piatetsky Shapiro, and P. Smyth," From Data Mining to Knowledge Discovery in Databases*", AI Magazine*, 1996, Vol no 17, Issue no 3.
[3] Justin J. Miller,"Graph Database Applications and Concepts with Neo4j", *Proceedings of the Southern Association for Information Systems Conference*, Atlanta, GA, USA March 23rd-24th, 2013, vol 3, issue 2, pg 141-146.
[4] A. Dries and S. Nijssen," Analyzing graph databases by aggregate queries", *MLG*, 2010 Washington DC, U.S.A,vol 3, issue 2.
[5] I. Fischer and T. Meinl," Graph Based Molecular Data Mining - An Overview", IEEE, 2008, vol 90, issue 10.
[6] X. Yan and J. Han," gSpan: Graph-Based Substructure Pattern Mining" , 1995, vol 75, issue 1, pg 63-92
[7] D. Chakrabarti, Y. Zhan and C. Faloutsos," R-MAT: A Recursive Model for Graph Mining", *School of Computer Science*, CMU, vol 19, issue 2, pg 160-169.
[8] F. Eichinger, K. Bohm and M. Huber, " Improved Software Fault Detection with Graph Mining", *International Workshop on Mining and Learning with Graphs*, Helsinki, Finland, 2008, vol 90, issue 10.
[9] J. Wang, W. Hsu Mong and L. Lee Chang Sheng," "A Partition-Based Approach to GraphMining", *International Conference on Data Engineering IEEE* , 2006 , vol 22.
[10] J. Huan, W. Wang, J. Prins, "Efficient Mining of Frequent Subgraph in the Presence of Isomorphism*",* vol 63, issue 19.
[11] X. Yan, J. Han, "CloseGraph: Mining Closed Frequent Graph Patterns", SIGKDD '03 Washington, DC, USA.
[12] J. Huan, W. Wang, J. Prins," SPIN: Mining Maximal Frequent Subgraphs from Graph Databases", *KDD'04, August 22–25, 2004, Seattle, Washington, USA.*
[13] C. Vicknair, "A Comparison of a Graph Database and a Relational Database", *ACMSE*, 2010, Oxford, MS, USA ,ACM 978-1-4503-0064
[14] G. Jaiswal and A. Prakash Agrawal, " Comparative analysis of Relational and Graph databases", *IOSR Journal of Engineering (IOSRJEN),* vol 3, issue 8.
[15] C. Cattuto, A. Panisson, M. Quaggiotto and A. Averbuch, "Time-varying Social Networks in a Graph Database", *IJCA*, 2013, vol 20, issue 1.

