International Journal of Engineering Science Invention (IJESI)
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org ||Volume 8 Issue 07 Series. II || July 2019 || PP 49-55

Comparative Study of Frequent Graph Structure Pattern Mining Algorithm

Y L Patel[1], Prof. C. K. Bhensdadia[2] and Dr. A. P. Ganatra[3]
[1]*IT Department, Faculty of Technology, D. D. UniversityCollege Road, Nadiad (Gujarat-India)*
[2]*CE Department, Faculty of Technology, D. D. UniversityCollege Road, Nadiad (Gujarat-India)*
[3]*CE Department, C. S. Patel Inst. of Technology CHARUSAT, Changa (Gujarat-India)*
Corresponding Author: Y L Patel

Abstract— *Structure base frequent pattern mining andclassification has remained one of the most interesting and widely use for bioinformatics, chem.-informatics, web content classification like Wikipedia, social networking, molecular structure classification, protein structure identification...etc. The unknown molecular structure, Social Networking, Protein Structure classification…etc are highly non sequential structure data and it depends on many parameters like type of relation between them and property which makes it more interesting at the same time very difficult to classify dataset. Because of complexity and intrications, the problem has received attention of many researchers all over the world. As a lot of application dataset are represented in term of graph structure pattern, software developers experience difficult to classify the frequent graph sub structure in datasets. Many different types of algorithms have been suggested by researchers' over the period of time base on two types one is Apriori base and another is frequent pattern growth base approaches. In the Apriori-based graph mining method Candidate set generation is still costly especially when there exist a large number of long patterns In Frequent pattern growth Time is wasted as the only pruning that can be done is on single items. These approaches have many pitfalls like most of them identify repeated frequent patterns; all identified frequent patterns are not interesting. This researcher work suggests and provides comparative survey of different sub graph mining algorithm like Gspan, Subdue, FSM …etc. This Comparative survey can be useful to identify and select appropriate sub graph mining algorithm for different types of application.*

Keywords*—Frequent sub graph mining, Isomorphism, Pattern growth, Apriori*

Date of Submission: 10-07-2019 Date of acceptance: 28-07-2019

I. Introduction

There are different types of data which can be represented in term of graph structure like web resource, social networking, molecular structure...etc. To classify structure we require identifying most frequent pattern. For identification of frequent pattern there are different graph mining techniques like Apriori base technique and frequent pattern growth base technique. Each technique has some pros and corns regarding performance and working mechanism. This section provides a generic overview of the process of FSM. Any frequent sub graph mining process involves 3 aspects, i) graph representation ii) sub graph Enumeration and iii) frequency counting.

1.1 Graph Representations

The simplest mechanism whereby a graph structure can be represented is by employing an adjacency matrix or adjacency list. Using an adjacency matrix the rows and columns represent vertexes, and the intersection of row i and column j represents a potential edge connecting the vertexes vi and vj . The value held at intersection < i, j > typically indicates the number of links from vi to vj . However, the use of adjacency matrices, although straightforward, does not lend itself to isomorphism detection, because a graph can be represented in many different ways depending on how the vertexes (and edges) are enumerated Washio&Motoda 2003. With respect to isomorphism testing it is therefore desirable to adopt a consistent labelling strategy that ensures that any two identical graphs are labelled in the same way regardless of the order in which vertexes and edges are presented (i.e. a canonical labelling strategy). A canonical labelling strategy defines a unique code for a given graph.

1.2 Canonical labelling:

It facilitates isomorphism checking because it ensures that if a pair of graphs is isomorphic, then their canonical labelling will be identical Kuramochi&Karypis. One simple way of generating a canonical labelling is to flatten the associated adjacency matrix by concatenating rows or columns to produce a code comprising a list

of integers with a minimum (or maximum) lexicographical ordering imposed. To further reduce the computation resulting from the permutations of the matrix, canonical labelling are usually compressed, using what is known as a vertex invariant scheme Read & Corneil, that allows the content of an adjacency matrix to be partitioned according to the vertex labels. Various canonical labelling schemes have been proposed, some of the more significant are described in this subsection.

1.3 Minimum DFS Code (M-DFSC):

There are a number of variants of DFS encodings, but essentially each vertex is given a unique identifier generated from a DFS traversal of a graph (DFS subscripting). Each constituent edge of the graph in the DFS code is then represented by a 5-tuple: (i, j, li, le, lj), where i and j are the vertex identifiers, li and lj are the labels for the corresponding vertexes, and le is the label for the edge connecting the vertexes. Based on the DFS lexicographic order, the M-DFSC of a graph g can be defined as the canonical labelling of g.

1.4 Canonical Adjacency Matrix (CAM):

Given an adjacency matrix M of a graph g, an encoding of M can be obtained by the sequence obtained from concatenating the lower or upper triangular entries of M, including entries on the diagonal. Since different permutations of the set of vertexes correspond to different adjacency matrices, the canonical (CAM) form of g is defined as the maximal (or minimal) encoding. The adjacency matrix from which the canonical form is generated defines the Canonical Adjacency Matrix or CAM Inokuchi et al.; Kuramochi&Karypis ; Huan et al..

II. Sub graph Enumeration

The current methods for enumerating all the sub graphs might be classified into two categories: one is the join operation adopted by FSG and AGM and another one is the extension operation. The major concerns for the join operation are that a single join might produce multiple candidates and that a candidate might be redundantly proposed by many join operations. The concern for the extension operation is to restrict the nodes that a newly introduced edge may attach to. Equivalence class based extension is founded on a DFS-LS representation for trees. Basically, a (k + 1)-sub tree is generated by joining two frequent k-sub trees. The two k sub trees must be in the same equivalence class. An equivalence class consists of the class prefix encoding, and a list of members. Each member of the class can be represented as a (l, p) pair, where l is the k-th vertex label and p is the depth-first position of the k-th vertex's parent. It is verified, in Zaki , that all potential (k + 1)-sub trees with the prefix [C] of size (k − 1) can be generated by joining each pair of members of the same equivalent class [C].Equivalence classes can be based on either prefix or suffix.

III. Frequency Counting

Two Methods are used for graph counting: Embedding lists (EL) and Recomputed embeddings (RE). For graphs with a single node we store an embedding list of all occurrences of its label in the database. For other graphs a list is stored of embedding tuples that consist of (1) an index of an embedding tuples in the embedding list of the predecessor graph and (2) the identifier of a graph in the database and a node in that graph. The frequency of a structure is determined from the number of different graphs in its embedding list. Embedding lists are quick, but they do not scale very well to large databases. The other approach is based on maintaining a set of active" graphs in which occurrences are repeatedly recomputed.

IV. Issues in Graph Mining

There are different types of issues associated with frequent graph mining Techniques like graph structure representation issue, Isomorphism of sub graph structure, interestingness of identified frequent structure associated with graph, scalability in term of graph structure and numbers of graph structures. So that during design of frequent graph mining algorithm we have to take care about the issues which already we have in previously proposed solutions.

4.1 Graph Representation

There are different representation techniques like

Adjacent Matrix, Adjacent List, sequential representation, tries...etc. If we use Adjacent matrix we have to take care of number of dimension like labelling of graph. Without labelling we can't remove isomorphism. If the size of graph is very large then adjacent list and tries required more time complexity to represent in that graph structure but it is useful to make canonical labelling and DFS code to remove sub graph isomorphic problem.

4.2 Isomorphism of sub graph structure

For large scale graph frequent sub graph structure which we identify are not always different. Same sub graph structure may be identified more than one time so that we have to take care of this type of dummy structure which are identical and remove this type of isomorphic structure using canonical labelling or lexical DFS code representation. So that it is required to reduce isomorphism for this type of redundant sub graph structure.

4.3 Interestingness of frequent sub structure

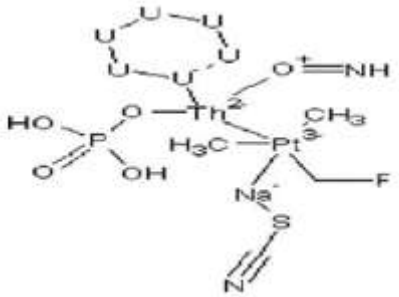

Figure 1 Unknown Molecular

For given example if you want to classify unknown molecular in one of three different class. Same sub structure like CH3 may appear in these three classes. So that it is important to measure interestingness of pattern base on statistic analysis to find interestingness of pattern. Base on information gain and statistic analysis we can identify interestingness of frequent sub graph pattern so that it is easy to classify in appropriate class.

.

4.4 Scalability

Scalability of graph can be measure in term of size of graph structure and in term of numbers of graph structure. It is also dependent on application like social networking, molecular classification, protein structure...etc for which graph mining algorithm is developed. Base on that we have to design algorithm for scalable all type of graph data structure.

4.5 Data Set format

It is also important to provide mechanism to type cast application source data set in to appropriate data format conversion like social networking or Wikipedia data source is always in html or xml format so that it is required to type cast in appropriate format so that user can type cast it.

V. Apriori Base Techniques

5.1 Apriori Base Graph Mining (AGM)

The mathematical graph theory based approach mines a complete set of sub graphs under mainly support measure. The initial work is AGM (Apriori-based Graph Mining) system. The basic principle of AGM is similar to the Apriori algorithm for basket analysis. Starting from frequent graphs where each graph is a single vertex, the frequent graphs having larger sizes are searched in bottom up manner by generating candidates having an extra vertex. An edge should be added between the extra vertex and some of the vertices in the smaller frequent graph when searching for the connected graphs. One graph constitutes one transaction. The graph structured data is transformed without much computational effort into an adjacency matrix mentioned.

5.1.1 Candidate Generation

The two indices which are identical to the definitions of support" and confidence" in the basket analysis are introduced. Definition 8 (Support and Confidence) given a graph Gs, the support of Gs is defined as Given two induced sub graphs Gb and Gh, the confidence of the association rule Gb => Gh is defined as

$$sup(G_s) = \frac{number\ of\ graph\ transactions\ G\ where\ G_s \subset G \in GD}{total\ number\ of\ graph\ transactions\ G \in GD}.$$

If the value of sup(Gs) is more than a threshold value min sup, Gs is called as a frequent induced sub graph". Similarly to the Apriori algorithm, the candidate generation of the frequent induced sub graph is made by the level wise search in terms of the size of the sub graph.

5.1.2 Frequency Calculation

Frequency of each candidate induced sub graph is counted by scanning the database after generating all the candidates of frequent induced sub graphs and obtaining their canonical forms. Every transaction graph G in the database can be represented by an adjacency matrix Xk, but it may not be a normal form in most cases. Since the candidates of frequent induced sub graphs are normal forms, the normalization must be applied to Xk of each transaction G to check if the candidates are contained in G. As previously described, the procedure of the normalization of Xk can derive the normal form of every induced sub graph of G in the intermediate levels. Thus, the frequency of each candidate is counted based on all normal forms of the induced sub graphs of G. When the value of the count exceeds the threshold min sup, the sub graph is a frequent induced sub graph. Once all frequent induced sub graphs are found, the association rules among them whose confidence values are more than a given confidence threshold are enumerated by using the algorithm similar to the standard basket analysis.

5.2 Frequent Sub Graph Discovery (FSG)

In developing our frequent sub graph discovery algorithm, we decided to follow the level-by-level structure of the Apriori algorithm used for finding frequent item sets in market-basket datasets. The motivation behind this choice is the fact that the level-by-level structure of the Apriori algorithm achieves the highest amount of pruning In the rest of this section we describe how FSG generates the candidates sub graphs during each level of the algorithm and how it computes their frequency.

5.2.1Generation

In the candidate generation phase, we create a set of candidates of size k +1, given frequent k-sub graphs. Candidate sub graphs of size k+1 are generated by joining two frequent k-sub graphs. In order for two such frequent k-sub graphs to be eligible for joining they must containthe same (k¡1)-sub graph. We will refer to this common (k¡1)-sub graph among two k-frequent sub graphs as their core.Unlike the joining of item sets in which two frequent k-size item sets lead to a unique (k + 1)-size item set, the joining of two sub graphs of size k can lead to multiple distinct sub graphs of size k + 1. This can happen for the following three reasons. First, the diff erence between the shared core and the two sub graphs can be a vertex that has the same label in both k-sub graphs. In this case, the joining of such k-sub graphs will generate two distinct sub graphs of size k + 1. The pair of graphs G^4_a and G^4_bgenerates two different candidates $G^5_a$and G^5_b. Second,the core itself may have multiple automorphisms, and each of them can lead to a different (k + 1)-candidate. In the worst case, when the core is an unlabeled clique, the number of automorphisms is k!. An example for this case in which the core—a square of four vertices labelled with a—has four automorphisms resulting in three differentcandidates of size 6. Because every core has one fewer edge, for a pair of two k-sub graphs to be joined, the number of multiple cores is bounded by k ¡ 1.

For each pair of frequent sub graphs, fsg-gen starts by detecting all the cores shared by the two frequent sub graphs. Then, for each pair of sub graphs and a shared core, fsg-join is called at Line 6 to generate all possible candidates of size k + 1. Once a candidate is generated, the algorithm first checks if the candidate is already in $\mathbf{C}^{k+1}$. If it is not, then fsg-gen verifies if all its k-sub graphs are frequent. If they are, fsg-join then inserts the candidate into $\mathbf{C}^{k+1}$, otherwise it discards the candidate. Given a pair ofk-sub graphs, G^k_1 and G^k_2, and a core of size k ¡ 1, first fsg- join determines the two edges, one included only in G^k_1 and the other only in G^k_2 which are not a part of the core. Next, fsg-join generates all the automorphisms of the core. Finally, for each set of the two edges, the core, and the automorphisms, fsg-join integrates the two sub graphs G^k_1 and G^k_2 into one candidate of size k + 1.

5.2.2 Frequency Counting

Once candidate sub graphs have been generated, FSG computes their frequency. The simplest way of achieving this is for each sub graph to scan each one of the graph transactions in the input dataset and determine if it is contained or not using sub graph isomorphism. Nonetheless, having to compute this isomorphism is particularly expensive and this approach is not feasible for large datasets. In the context of frequent item set discovery by Apriori, the frequency counting is performed substantially faster by building a hash-tree of candidate item sets and scanning each transaction to determine which of the item sets in the hash-tree it supports. Developing such an algorithm for frequent sub graphs, however, is challenging as there is no natural way to build the hash-tree for graphs.

However, the computational advantages of the TID lists come at the expense of higher memory requirements. In particular, when FSG is working on finding the frequent patterns of size (k + 1), it needs to store in memory the TID lists for all frequent patterns of size k. FSG can be easily extended to work in cases where the amount of available memory is not sufficient for storing the TID lists of a particular level by adopting a depth-first approach for frequent pattern generation. Starting from a subset of sub graphs of size k without generating all the rest of size k sub graphs, we can proceed to larger size. In this way, we may not be able to get

the same effect of pruning based on the downward closure property. Nevertheless, it is beneficial in terms of memory usage because at each phase we keep smaller number of sub graphs and their associated TID lists.

VI. Frequent Pattern Growth Base Technique

In order to avoid the overhead of apriori algorithms, non-Apriority-based algorithms have been developed, most of which adopt the pattern-growth methodology, as discussed below. Pattern-growth-based graph pattern mining algorithms include gSpan by Yan and Han (2002),MoFa by Borgelt andBerthold (2002) [7], FFSM by Huan et al. (2003), SPIN by Huan et al. (2004), and Gaston by Nijssen andKok (2004). These algorithms are inspired by PrefixSpan(Pei et al. 2001),TreeMinerV(Zaki 2002), and FREQT (Asai et al. 2002) at mining sequences and trees, respectively. The pattern-growth mining algorithm extends a frequent graph by adding a new edge, in every possible position. A potential problem with the edge extension is that the same graph can be discovered many times. The gSpan algorithm solves this problem by introducing a right-most extension technique, where the only extensions take place on the right-most path. A right-most path is the straight path from the starting vertex v0 to the last vertex vn, according to a depth-first search on the graph. Typical pattern growth algorithms are discussed in the following paragraphs.

6.1 Graph-Based Substructure Pattern Mining (Gspam)

Approaches for frequent graph-based pattern mining in graph datasets and propose a novel algorithm called gSpan (graph-based Substructure pattern mining), which discovers frequent substructures without candidate generation. gSpan builds a new lexicographic order among graphs, and maps each graph to a unique mini-mum DFS code as its canonical label [6]. Based on this lexicographic order, gSpan adopts the depth first search strategy to mine frequent connected sub graphs efficiently. Our performance study shows that gSpan substantially outperforms previous algorithms, sometimes by an order of magnitude.

GSpan, which targets to reduce or avoid the significant costs mentioned above. If the entire graph dataset can t in main memory, gSpan can be applied directly; otherwise, one can first perform graph-based data projection, and then apply gSpan. To the best of our knowledge, gSpan is the algorithm that explores depth first search (DFS) in frequent sub-graph mining. Two techniques, DFS lexicographic order and minimum DFS code, are introduced here, which form a novel canonical labelling system to support DFS search. GSpan discovers all the frequent sub graphs without candidate generation and false positives pruning. It combines the growing and checking of frequent sub graphs into one procedure, thus accelerates the mining process.

6.1.1 DFS Lexicographic Order

This section introduces several techniques developed in gSpan, including mapping each graph to a DFS code (a sequence), building a novel lexicographicordering among these codes, and constructing a search tree based on this lexicographic order.

6.1.2 DFS Code

When performing a depth- first search in a graph, we construct a DFS tree. One graph can have several different DFS trees. For example, graphs in Fig. 2(b)-(d) are isomorphic to that in Fig. 2(a). The thickened edges in Fig. 2(b)-(d) represent three different DFS trees for the graph in Fig. 2 a. The depth- first discovery of the vertices forms a linear order. We use subscripts to label this order according to their discovery time. i < j means vi is discovered before vj. We call vo the root and vn the right most vertex. The straight path from vo to vn is named the right most path. In fig. 2 b-d three different subscripting are generated for the graph in fig. 2 a.

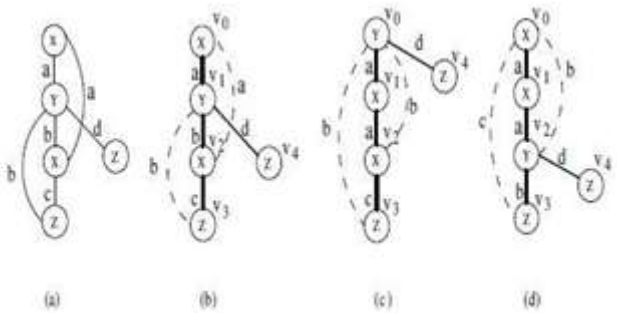

Fig. 2 Depth First Search Tree

Let the canonical DFS code to be the smallest code that can be constructed from G (denoted min(G)) Theorem: Given two graphs G and H, they are isomorphic if and only if min(G)=min(H) mining Sub graph

frequent is equivalent mining their corresponding minimum DFS codes Can be done sequentially by pattern mining algorithms

Each node represents DFS code. Relations between parents and children. With label set L, DFS Code Tree contains all possible graphs for this label set Each graph on the n-th level in the DFS Code Tree constrains n-1 edges. DFS code tree contains minimum DFS codes for all graphs (DFS Code Tree Covering).

If a graph G is frequent, then any subgraph of G is frequent. If G is not frequent, then any graph which contains G is not frequent. If a DFS code α is frequent, then every ancestor of α is frequent. If α is not frequent then every descendant of α is not frequent.

Some graphs can have more DFS nodes corresponding to it in DFS Code Tree. The first occurrence is the minimum DFS code. Theorem: If DFS code is not the minimum one, we can Prune the entire sub tree below this node, and still preserve DFS Code Tree Covering. Pre-order searching of DFS Code Tree guarantees that we can enumerate all potential frequent sub graphs.

6.2 Graph Sequence Tree extractiON (Gaston)

In this technique for sub structure searching first path are considered, then paths are transformed to trees and finally trees are transformed to graphs. It starts with the possibilities of quick starting the search for frequent structures by integrating a frequent path, tree and graph miner into one algorithm. The main challenge is how to split up the discovery process into several phases efficiently. The frequency and support of a graph G in D is defined as per given below.

Feq(G,D) = #{G' D | G $\subseteq$ G'}
Support (G,D) = freq(G,D) / |D|
G1 G2 => freq(G1, D) >= freq(G2, D)

First compare the labels at both ends of the path by comparing the tuple (l(v1), l(e1)) with the tuple (l(Vn), l(En-1)) lexicographically; if one end is higher than the other, the path without the highest tuple is considered to be the unique predecessor. For one specific orientation of a path three symmetry variables are maintained. One for the oriented path called total symmetry, one for front and one for back symmetry. Each of these variables has one of three values: 0, if the corresponding string is symmetric; -1 if the reverse string of the current orientation is the lowest; +1, if the string of the current orientation is the lowest.

6.3 Subdue Greedy Approach

Subdue is a graph-based relational learning system. The work on Subdue is one of the pioneering works in the field of graph-based data mining. Inputs to the Subdue system can be a single graph or a set of graphs. The graphs can be labelled or unlabeled. Subdue outputs substructures that best compress the input dataset according to the Minimum Description Length (MDL) [17] principle. Subdue performs a computationally-constrained beam search which begins from substructures consisting of all vertices with unique labels. The substructures are extended by one vertex and one edge or one edge in all possible ways, as guided by the example graphs, to generate candidate substructures. Subdue main-trains the instances of substructures in the examples and uses graph isomorphism to determine the instances of the candidate substructure in the examples. Substructures are then evaluated according to how well they compress the Description Length (DL) of the dataset. The DL of the input dataset G using substructure S can be calculated using the following formula,

I(S) + I (G/S)

A common characteristic of a majority of the graph-based data mining methodologies described above is that they focus on complete, frequent sub-graph discovery. Complete, frequent sub-graph discovery algorithms such as FSG and gSpan are guaranteed to find all sub graphs that satisfy the user specified constraints. Although completeness is a fundamental and desirable property, one cannot ignore the fact that these systems typically generate a large number of sub-structures, which by themselves provide relatively less in-sight about the domain. Typically, interesting substructures have to be identified from the large set of substructures either by domain experts or by other automated methods so as to achieve insights into this domain. Subdue typically produces a smaller number of substructures which best compress the graph dataset. These few substructures which compress the input dataset can provide important insights about the domain.

VII. Comparison Of Current Systems

Here we provide the comparative study of current electronic payment system. After study of four payment option we present the comparative study table.

Algorithm	Graph Representation	Candidate Generation	Limitations
FARMER	Tries Structure	Level-wise search ILP	In Efficient
FSG	Adjacency List	One edge extension	NP-Complete
SUBDUE	Adjacency Matrix	Level wise Search	Extremely small of patterns
GSpan	Adjacency List	Right Most Extension	Not scalable
FFSM	Adjacency Matrix	Merging and Extension	Np-Complete
Gaston	Hash Table	Extension	Interesting Patterns may be lost.
MOFA	Adjacency List	Right Most Extension	Frequent graphs Generated may not be exactly frequent

VIII. Conclusion

Comparative study of various frequent sub graph pattern mining algorithms namely FSM, Gaston, GSpan ..etc is carried out and result is presented which can be used as a guide-line to develop more efficient frequent sub graph mining algorithm. It provides necessary inputs to the user for selecting appropriate frequent sub graph mining algorithm for different types of application for graph pattern classification problems.

References:

[1]. DISCOVERING Interesting Molecular Substructures for Molecular Classification Winnie W. M. Lam and Keith C. C. Chan*, Member,IEEE ,2010
[2]. FREQUENT SUBGRAPH MINING ALGORITHMS A SURVEY AND FRAMEWORK FOR CLASSIFICATION K.Lakshmi1 and Dr. T. Meyyappan2, 2012
[3]. An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data Akihiro Inokuchi, Takashi Washio and Hiroshi Motoda, 2003
[4]. A Quickstart in Frequent Structure Mining can make a Difference Siegfried Nijssen LIACS, Leiden University, Joost N. Kok LIACS, Leiden University, 2004
[5]. gSpan: Graph-Based Substructure Pattern Mining Xifeng Yan Jiawei HanDepartment of Computer Science University of Illinois at Urbana-hampaign xyan, hanj_@uiuc.edu, 2003
[6]. Subdue: CompressionBased Frequent Pattern Discovery in Graph Data Nikhil S. Ketkar University of Texas at Arlington ketkar@cse.uta.eduLawrence B. Holder University of Texas at Arlingtonholder@cse.uta.eduDiane J. Cook University of Texas at Arlingtoncook@cse.uta.edu, 2003
[7]. Substructure Discovery in the SUBDUE System by Lawrence B. Holder, Diane J. Cook and Surnjani Djoko, 2004
[8]. CloseGraph: Mining Closed Frequent Graph Patterns Xifeng Yan xyan@uiuc.edu, Jiawei Hanhanj@uiuc.edu, 2003

Y L Patel" Comparative Study of Frequent Graph Structure Pattern Mining Algorithm" International Journal of Engineering Science Invention (IJESI), Vol. 08, No. 07, 2019, PP 49-55