EFFICIENT DATA RETRIEVAL FROM GRAPH DATABASE FOR MEDICAL APPLICATIONS

A PROJECT REPORT

Submitted by

ARUNKUMAR A R (2013503533)

VENKATESH S (2013503556)

JAYASURYA K (2013503510)

in partial fulfilment for the award of the degree
of
BACHELOR OF ENGINEERING
in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER TECHNOLOGY

MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY, CHENNAI 600 044

APRIL 2017

BONAFIDE CERTIFICATE

Certified that this project report "**EFFICIENT DATA RETRIEVAL FROM GRAPH DATABASE FOR MEDICAL APPLICATIONS**" is the bonafide work done by **ARUNKUMAR A R (2013503533), VENKATESH S (2013503556) and JAYASURYA K (2013503510)** under my supervision, in partial fulfilment for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further, that to the best of my knowledge the work reported here in does not form part or full of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion to this or any other candidate.

SIGNATURE
Dr. P. ANANDHAKUMAR
PROFESSOR AND HEAD
DEPT OF COMPUTER TECHNOLOGY
MADRAS INSTITUTE OF
TECHNOLOGY
ANNA UNIVERSITY, CHROMPET
CHENNAI – 600 044.

SIGNATURE
Dr. P. JAYASHREE
PROJECT SUPERVISOR
ASSOCIATE PROFESSOR
DEPT OF COMPUTER TECHNOLOGY
MADRAS INSTITUTE OF
TECHNOLOGY
ANNA UNIVERSITY, CHROMPET

ACKNOWLEDGEMENT

We are highly indebted to our respectable Dean, **Dr. A.RAJADURAI** and to our reputable Head of the Department **Dr. P.ANANDHAKUMAR**, Department of Computer Technology, MIT, Anna University for providing us with sufficient facilities that contributed to success in this endeavour.

We would like to express our sincere thanks and deep sense of gratitude to our Supervisor, **Dr. P.JAYASHREE** for her valuable guidance, suggestions and constant encouragement which paved way for the successful completion of this phase of project work.

We would be failing in our duty, if we forget to thank all the teaching and non-teaching staff of ours department, for their constant support throughout the course of our project work.

ARUNKUMAR A R (2013503533)

VENKATESH S (2013503556)

JAYASURYA K (2013503510)

ABSTRACT

Big data refers to the data sets that are very large, unstructured and complex. Traditional data processing applications are difficult to apply on such huge volume of datasets. Data capturing, storing, querying are some of the challenges faced in processing of big data. Using Graphs, we can easily represent the relationship between large quantities of data also. A graph can also serve as a general-purpose substrate for evaluating decision-making algorithms. Genomic and biological data are complex that can be represented as graphs. Generally, scientific datasets are large, complicated and are unstructured data which require fast processing. So, pre-processing and indexing techniques can help to improve the efficiency. A huge sequence of unreadable and non-relational letters is present in Genomic and biological data which is an example of unstructured data. Datasets in the medical and scientific fields keep increasing in large amounts due to various research and experiments. In order to provide meaning to this large volume of data, efficient methods to store, retrieve and analyse must be provided so that processing can be made easier. In real applications most of the graph data are noisy and incomplete. So it has become increasingly important to retrieve graphs in the graph database which match the query graph approximately, rather than exact graph matching. Hence, based on a given query graph, similar graphs are retrieved efficiently and comparative complexity analysis of path based indexing and structure based indexing method is done.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVATIONS

DAG	Directed Acyclic Graph
GED	Graph Edit Distance
GESC	Graph Edit Similarity Computation
GD	Graph Database
GXL	Graph Exchange Language
PPI	Protein Protein Interaction
Q	Query Graph
QAP	Quadratic Assignment Problem
R	Result Set
SMS	Sub graph Matching with Set Similarity

CHAPTER 1

INTRODUCTION

1.1 BIG DATA

The term big data refers to the data sets that are very large, unstructured and complex. The traditional data processing applications are inadequate since the data set are large and complex in big data. Big data includes the data sets whose storage is beyond the typical database software tools ability to capture, analyze, manage and store. Some of the challenges that the big data analytics currently facing includes data capturing, data curation, storage, querying and sharing. Analysing accurately such large sets of data will give efficient insights in the business process modelling and scientific decision making systems which can result in greater operational efficiency, cost reduction and reduced risk.

1.2 CHARACTERISTICS AND STORAGE OF BIG DATA

Three important big data characteristics are

- Volume
- Velocity
- Variety

There are various big data stores. Some of them are

- HBase
- Hadoop Distributed File System (HDFS)
- Hive
- Data models which includes key value, graph, document, column-family

1.3 ANALYSIS OF BIG DATA

Map reduce is one of the best known method for turning raw data into useful vital information. Map Reduce is a procedure of taking a large data set and doing computations on the large data set across multiple systems or computers in parallel. Map reduce serves as a model for how program or a information is often used to refer to the implementation of the model.

Map Reduce consists of two different parts. The first is the Map function. Filtering and sorting is done by the Map function. It takes data and places it inside of categories so that it can be analyzed later. The Second is the Reduce function which combines all the data and provides a summary. Even while largely credited to a scientific research which is done at Google, Map Reduce is now a generic term and it refers to a general procedure which used by many technologies.

1.4 GRAPH DATABASE

A graph information could be a information which uses linguistics queries using graph structures which has edges, nodes and properties to represent and store information. The system's key idea is that the graph (which can be relationship or edge), that relates directly information things within the store. The relationships permit information within the store to be connected along and is retrieved in one operation in most of the cases.

Graph databases area unit supported graph theory. Graph databases use nodes, edges and properties.

- Nodes represent entities like folks, businesses, accounts, or any other item you may wish to stay track of. They're roughly the equivalent of the record, relation or row in a very electronic information service, or the document in very document information.

- Edges are additionally referred to as graphs or relationships. It measures the lines that connect nodes to different nodes and they represent the connection between them. Meaning patterns emerge once examining the connections and interconnections of nodes, properties, and edges. Edges also measure the key construct in graph databases, representing associate degree abstraction that are circuitously enforced in different systems.
- Properties are the one that measure the pertinent information that relate to nodes. As an example, if we take Wikipedia were one in every of the nodes, one may need it tie to properties like web site, reference material, or word that starts with the letter ‘w’ accounting to that particular aspect of Wikipedia that measure pertinent to the actual information.

1.5 RELATIONAL AND GRAPH DATABASE

The graph databases cannot handle all types of relational queries and there is no natural way to map SQL queries to graph queries. So graph databases cannot considered as a replacement for the relational databases. On the other hand, the relational databases mainly deals with the tables which have right and predetermined schema. So we can write queries up to a particular and fixed depth by joining those normalized tables, but what do we do if the depth of the interrelationship cannot be predetermined for the given application and the input data is arbitrary, changing and ad hoc in nature, in these cases graph databases comes into rescue. It is intuitive to represent the ad hoc data as graph, and we can recursively iterate the formed graph till any depth to analyse, match pattern and for digital learning. In relational database management system the data sets are progressively filtered and grouped, while graphs are usually navigated and recursively defined depth but not pre-determined joins.

Compared to relational database, graph databases are faster in case of associative data sets as the data sets maps more naturally to the graphs as objects. So, the associative data sets that are represented as graphs scale well for large datasets. In most of the object oriented programming scenario it is more intuitive to relate the data sets as graphs. But still we cannot say that graph databases are the general replacement for the relational databases.

Graph databases scales well as they don’t involve complex joins of different normalized table as in the relational database management system. Graph databases work even faster for graph-like queries, for example given a county map, finding the shortest path between the two cities. So, the graph database has its own advantages and specific applications and it is not considered as the general replacement of the relational data management system

1.6 GRAPH EXCHANGE LANGUAGE(GXL)

GXL (Graph eXchange Language) is an XML-based commonplace exchange format for sharing information between tools. Formally, GXL represents written, attributed, directed, ordered graphs that are unit extended to represent hyper graphs and class-conscious graphs. This versatile information model is used for object-relational information and a large sort of graphs. A plus of GXL is that it is accustomed exchange instance graphs at the side of their corresponding schema data in an exceedingly uniform format, i.e. employing a common document sort specification. GXL is employed to produce ability of graph-based tools. GXL has been legal by reengineering and graph transformation analysis communities and is being thought of for adoption by different communities.

1.7 GRAPH VISUALISATION

As the scientific data has more interrelated attributes, it is more intuitive to store it in a graph database. In real applications most of the graph data are noisy and incomplete. So it has become increasingly important to retrieve graphs in the graph database which match the query graph approximately, rather than exact graph matching.

Applications include

- Simulating the gene function.
- Cataloging biological phenomenon.
- DNA and protein structures mapping.

Decision Making is done confidentially by the accuracy of the big data. Greater operational efficiency, reduced risk and cost reductions are meant by the better decisions. The novel work is to propose an efficient data storage and retrieval model which supports effective querying based on storage and to develop an infrastructure to build knowledge base on research applications by a predictive modelling. Disjoint partition based filter, branch based filtering models suffers some disadvantages like finding the optimal disjoint partition. To develop hybrid filtering model, with heuristics for scientific data to effectively find graph similarity search over large graph datasets.

1.8 PROTEIN – PROTEIN INTERACTION (PPI)

The most biological processes in a cell, including gene expression, morphology, cell growth, proliferation, motility, nutrient uptake, intercellular communication and apoptosis are facilitated by the proteins which are the workhorses of the cell. Protein analysis mainly focussed on single/one proteins, but because of the proteins interact with other proteins for majority of functioning, they should be monitored in the context of their interactions partners to fully understand how proteins interact with each other and identify biological networks which is important to understand how protein functions in the cell. PPIs refers to the interactions between proteins as a result of biochemical event of electrostatic forces. Proteins rarely act alone.

The protein in the organism hold special status. Every phenomenon of life goes through these structure and function of protein to be reflexed [8]. The protein interactions are important in studying the inter atomic system of the living cell and helps in the analysis of different diseases like Cancer, Alzheimer's disease, Creutzfeld-Jacob [21]. Protein – protein interactions are studied form different perspectives right form the bioinformatics, quantum chemistry and molecular dynamics. Proteins bind to each other through a combination of van der Waals forces, hydrophobic bonding, and salt bridges at specific binding domains on each protein. Constant development in these fields lead to the large scale growth of protein – protein interaction graphs. Making meaningful information out of this large data sets will empower the current knowledge on disease pathogenesis and biochemical cascades, as well as it provides new therapeutic targets.

1.9 FREQUENT PATTERN MINING

Subsequence, itemsets or substructures are some of the frequent patterns that seem during a knowledge set with frequency number which is less compared to user given value of the threshold. . A substructure will talk over with totally different structural forms, like sublattices,subgraphs, subtrees etc which can be combined along with subsequences or set of items. If a substructure occurs often during a graph information, it is referred to as a (frequent) structural pattern. Finding frequent patterns plays an important role in associations of mining, correlations, and plenty of alternative fascinating relationships among knowledge. Moreover, it helps in knowledge compartmentalization, classification, clustering, and alternative data processing tasks further. Frequent pattern mining is a very important data processing task and a targeted theme in data processing analysis.

1.10 GRAPH QUERIES

There are four different types of graph queries: Search of Substructure, Search of Reverse substructure, Search of substructure similarity and search of reverse similarity.

A query of substructure search searches the graph database for all the graphs which contain the query pattern, which can either be a graph where some parts are designated as wildcard or even a small graph.

A reverse substructure search query is also called as “Supergraph query”. It returns all the member graphs of the graph database which are contained in the query graph.

A search of substructure similarity of a query pattern in a database of graph will return all the approximately containing graph query pattern.

1.11 PROBLEM STATEMENT

Given a query graph ‘q’ and a large graph database containing various graphs $g_1,g_2,...g_n$, retrieve all similar graph patterns based on the query graphs using heuristic functions.

1.12 OBJECTIVE

To build an heuristic graph matching model to effectively find graph similarity in a large graph databases.

1.13 ORGANISATION OF THESIS

The following chapter in this thesis are organised as follows: Chapter 2 describes the literature survey undertaken for the project. Chapter 3 describes the proposed work and description of various modules in the project. Chapter 4 discusses about the implementation details with output screenshots. Chapter 5 discusses conclusion and future work.

CHAPTER 2

LITERATURE SURVEY

2.1 GRAPH DATABASE

Emir Septian Sori Dongoran et.al [2] proposed that not all kinds of information are appropriate for graph database, one that matched is that the molecular graph information kind. It has featured labelled vertices and afloat edges. Graph information even have categorisation method. For molecular data kind study case, [2] proposed a brand new algorithm GraphGrep which is the most acceptable technique as a result of it assume every node within the graph information incorporates a distinctive variety(node-id) and label (node-label). Therefore it is appropriate for molecular datatype. [2] uses a hash table (fingerprint) as an index, examining the graph information fingerprint with graph query fingerpint to filter the database and use Ullman algorithm to perform subgraph matching. Using [2] we have a tendency to additionally get the foremost economical length-path supported the deepest depth in a graph query.

Patil, Shefali et.al [7], it suggested an overall view about the graph database .Data has been hold on in a tabular form thereby increasing the classification and readability. The illustration of information in the sort of graph lends itself well to schema which is dynamic and data in a well structured manner. No customary system or source language has been denied for graph information. [13] Discussed current applications and implementations of graph databases, giving an summary of the various sorts out there and their application.

L.Zou et.al [9] suggested the conversion of RDF data into large graph database. The RDF is defined as the Resource Description Framework. It is a data model for modeling Web objects as part of developing the semantic web. It has been used in various application (ex-Wikipedia). Querying the large RDF data is very complex. So the RDF database is changed into graph database (RDF graph). The SPARQL query is changed into query graph. Now the similarity search over the graph database is done to provide the results. This problem now has become the sub graph matching over a large graph database. We already have lot of algorithm for the query graph retrieval over large graph database. In this method the RDF data is converted into Adjacency list table. When the query is converted into adjacency list table it will be easy to search over the RDF graph. Many web applications are based on RDF data. The RDF data are getting updated frequently so it is good to use graph database to represent RDF data.

C.Xiao et.al [11] discussed that Graphs are widely used to model complicated data semantics in many applications like social networks, chemistry, pattern recognition, etc. The graph edit distance is suggested in this method. Using the edit distance method the subgraph matching over a large graph database is done. The threshold value is given by the user. The edit distance of the subgraphs less than or equal to the threshold corresponding to the given query graphs will be the output. This method is also the similarity search not the exact match over the graph database. The major drawback from this method is this method follows path based indexing. But this method handles both scattered and clustered edit operations. Using graph database we can accommodate the future increments or updates will be added to the graph database.

DePiero F.W. et.al [23] stated that the graph structure is the very important means which models the schema less and complicated models such as chemical compounds, protein – protein interaction network, road network, chemical compounds and knowledge query systems [14]. Existing prediction algorithms will not work well in these situations. Introducing use of existing information which was collected for information management in high performance computing systems to build accurate coarse-grained prediction systems. Object abstraction can provide the pros of both files and blocks with better abstraction and flexibility.

2.2 GRAPH INDEXING

Xifeng Yan et.al [3] proposed the problems of categorization graphs and propose a unique answer by applying a graph mining technique. Completely different from the prevailing path-based strategies, [16] then proposed a new technique referred to as graph Index that makes use of frequent substructure as the basic categorization feature. Frequent substructures are important candidates since they explore the internal characteristics of the information and are comparatively stable to information updates. To scale back the dimensions of the index structure, 2 techniques, discriminative fragments and size-increasing support constraint are introduced. [22] showed that gIndex has ten times smaller index size, however achieves ten times higher performance as compared with a typical path-based technique.

G. Wang et.al [5], it detailed mainly about structure of the index for the problem of similarity search on a group of enormous distributed graphs and proposes an Q-gram plan which is an economical compartmentalization mechanism. By moldering the graphs into small fragments i.e little grams (which is organized by k-Adjacent pattern of the tree) and pairing-up on those k-Adjacent pattern trees, we can efficiently calculate the lower bound estimation based on their edit distance and can be used for candidate filtering.

Xiaoli Wang et.al [8], it proposed SEGOS, which is one of the query processing framework and method for efficient indexing for graph similarity search. A good two-level index is made off-line supported by graph decomposition subunits initially. Then, a completely unique search strategy supported by the index is projected as in [17]. Two algorithms custom-made from CA and TA methodology are integrated into the projected strategy to boost graph search. Graph pruning is continuously supported by the projected framework which is straightforward to be pipelined to support graph pruning at regular intervals. Based on the two real datasets which are taken as samples, in depth experiments are conducted and the effectiveness and quantifiability of the approaches are tested.

J. Rocha [21] discussed the importance of indexing. Graph is mainly indexed to facilitate the sub graph isomorphism and similarity queries. The project work consisted of various levels of structures [18]. The primary structure is first composed of a directed acyclic graph (DAG) that contains a vertex for each of the unique, induced sub graphs of the graphs of the database. The hash tale is the secondary structure, in which cross-indexes each sub graph for fast isomorphic lookup. The key challenge in using inputs of object-oriented applications are no longer expressed as stable file identifiers; rather than they become much more dynamic and unstatic, hidden inside application logic.

F. Bai et.al [24] proposed that querying is easy because of using index structures, but constructing the index structures is costly and resource consuming. The problem boils down for finding all fragments in a large set of graph that matches a given user query graph pattern [42]. Here we can propose a multi-step R-join procedure with fetch step and filter step based on cluster-based join-index with graph codes. The new algorithm outperforms the old traditional algorithm in case of the elapsed time, and it reduces dominant data transmission cost over the network. Since each step involves filter and fetch, the overall complexity of using this method is high. The algorithm is optimized using a stack-based algorithms to handle pattern queries in directed acyclic graphs, by building index structures for predecessor list which is used for the purpose of easy querying[26]. It is because of the index structures, the querying becomes easy.

Li Chen et.al [29] suggested a method for handling the path and pattern queries. [28] makes a stack based algorithms to handle the queries. It overcomes the disadvantages faced in the existing index structure approach that stores the pre computed transitive relations. It can be used as a graph model for many scientific applications. It proposes a quadratic time running algorithm for querying the graph. It achieves best running time for navigation graph.

2.3 SIMILARITY OF GRAPHS

W.Zheng et.al [1], proposed the study of similarity search of graph, which retrieves all the graphs which are similar to a given query graph under the properties of edit distance of the graph and a problematic method for edit-distance based similarity search problem. In [41], it additionally gift an even index structure, particularly u – tree, to facilitate effective pruning and economical question process.

K. Gouda et.al [4], proposed a method for calculating graph comparison based on edit distance of the graph. Existing edit distance of a graph calculation methods adopt the best first search or the breadth first search algorithm A Star. These algorithms are space and time bound. At most, these algorithms can compute or calculate the edit distance of the graphs containing twelve vertices in practice. To calculate graph edit similarity calculation on bigger and distant/distinct graphs, [27] presented a method, a novel mapping method which maps the edges. It is used for the computation of graph edit distance. It uses a sub structure isomorphism as a common enumeration solution. In [30], the algorithm uses a backtracking search which is joined with a various number of heuristics/algorithms to reduce space requirements and quickly remove away a big portion of the search space which is being mapped.

X. Zhao et.al [12] proposed a method using Edit Distance for similarity search in graph database. The GED (Graph Edit Distance) is the important function unit in this method. In this method we divide graph data into variable size non overlapping partitions. This is the first method to use variable size partitions of the graph database for graph indexing and filtering. This method also suggest partition based filtering algorithm for reducing time complexity. In GED subgraph search is to retrieve the data graphs that approximately contain the query graph. The similarity is defined as number edges need to be modified in the subgraphs of graph database respect to the given query graph. This method provide a dynamic partitioning technique to reduce the cost. The cost aware graph partitioning method is the advantage of this method. The updates on the graph database can be handled. It is the advantage of representing data in the graph structure.

L. Hong et.al [6] discussed the query of sub graph matching with set similarity (SMS2) over an oversized graph information, that retrieves sub graphs which are isomorphous structurally to the graph query. In order to do the efficiency method for query of sub graph matching with set similarity question, [36] styled a unique lattice-based index for knowledge graph, and light-weight signatures for each vertices of the query and knowledge vertices. [33] additionally proposed associate degree economical two-phase pruning strategy as well as set similarity pruning as well as pruning in an structural manner.

Zheng [35] proposed a method which is like searching RDF (Resource Description Framework). The RDF data retrieval is used in semantic web like Wikipedia. This method suggested data retrieval in a sub graph matching way. The large knowledge data base is already exist. It will built into a graph data base like converting from RDF data base to graph database. The same way the knowledge data base will be converted into graph database. Because it is easy to query from the graph structured data base comparing to normal knowledge database. The query is converted into graph. In this method after converting the knowledge database to graph database all the subgraphs in graph database are extracted. Compare these subgraphs to the query graph given by the user. In the comparison if result is match then output the matching subgraph.

Qiao [40] proposed that graph pattern matching has become the important topic in big data. Many big data applications use the method finding the matching sub graphs corresponding to query graph. Many algorithms are trying to improve the efficiency of the indexing and subgraph matching. In this method we use model checking technology to find the matching subgraphs similar to given query graph. [38] mainly focused on the matching phase not utilizing formulas. This method can be very useful in handling big set of data graphs. But this method doesn’t suggest any additional ways to handle the regularly updated graph database or incremental graph database.

Fred W. DePiero [23] proposed a method for structure wise comparison and finding the common subgraphs. Without any extra information like node or edge properties,colors etc, it helps us to identify the common subgraphs. It can be applied on any arbitrary undirected graph type. [31] suggested that it can be found out in polynomial bound time and worst case compute effort. It also verifies the validity of subgraph obtained and one to one mapping between the nodes. It compares the graph structure dynamically based on neighbourhood since it varies dynamically. This method can also be extended to finding common subgraphs using nodes attributes and colors.

2.4 GRAPH MINING

H. Li and C. Liu [22] proposed that data modelling is widely used procedure in a wide range of learning machines and data analytics problems. The technique of Sample Subset Optimization (SSO), which relies on a validation of a cross procedure used for selecting and identifying the most useful samples as sub groups.

S. Zhang et.al[10] proposed that Subgraph query has become most important task in graph matching. Graphs are large and has tens and thousands of vertices and millions of edges. The subgraph indexing and approximate matching suggest method for indexing subgraph over a very large graph database. Subgraph indexing is to find the query graph in the graph database. It is not necessary to find the exact matching subgraphs from the graph database. This method suggest approximate matching to the query graph. The major difficult task is to index the graph database so that the query matching retrieval from graph database will take less time. For approximate matching to the query graph the Edge Edit Distance is followed. The query graph edge can be modified so that it can match with the subgraphs in the graph database. The Edge Edit should be minimum. So that the user can give the threshold value which is maximum no of edges that can be modified to find the approximate matching subgraphs in the graph database.

2.5 BIOLOGICAL GRAPHS

K. Aoyama et.al [19] discussed about the protein sequence and their enormous development that challenged the bio scientists so that reliable and fully automated methods to quantify huge amount of data sequence in which large amount of work and efforts have been made. Since the practical methods suggested for protein analysis can be doubtful and workload consuming and may be very time consuming, and hence they are not used for identifying efficiently the gene assortment events in a quick span of time. Due to the recent methodologies restrictions, it is a huge task for the people in the scientific crowd to predict and follow the protein analysis and their experimental results. The phylogeny which are obtained are consistently in par with the tree phylogeny which are built by MEGA4 software.

Assayony.M et.al [20] discussed about graph based dividing methods where an vital and static algorithm in computer technology can be used to manipulate the group a large amount of protein sequences or protein sequences that shared a common attribute like similar amino acid sequences. As the size of the graph databases increases largely, a fast and quick graph based protein sequencing grouping method is very much needed to be developed. We have a parallel and distributed algorithm/procedure in graph-based grouping methods by increasing the performance of available method Protclust [32] by using parallel and distributed methods. The speed of the existing algorithm is being improved, involves using multiple processors without the state of idleness. The complexity of calculating the pair-wise similar is quadratic to the number the sequences compared.

Li, Peipei [39] stated that protein prediction function has become the one of the difficult problem in the bio genetics medical applications. Finding the similar structure to the query structure is difficult. This method focus on increasing the accuracy. This method also proposes the new algorithm for increasing the accuracy. The frequent patterns are generated for the database structure. The existing prediction methods are path based approach and protein-protein interaction information. This method represents the protein-protein interaction in undirected unweighted graph data structure. The neighbor finding and pattern finding and function annotation. The proposed method in [37] has increased the prediction accuracy to 0.6. This prediction function has three parts.

Sheng-Lung Peng [34] proposed a protein-protein interaction network based on amino acid. The link is polypeptide chains. In graph matching two graphs are given. We have to find the similarity between them by mapping the one graph to another. The graph spectrum method is suggested in this method. The graph spectrum method is used for graph matching in this method. The isomorphic relation is hard to determine when matching the two graphs. The Graph spectra method is used as a graph matching method for finding the similar protein structure related to given protein structure. So the protein structure is represented in graph structure and similarity matching done by graph matching to find all the similar protein structures.

CHAPTER 3

PROPOSED WORK

3.1 SYSTEM ARCHITECTURE

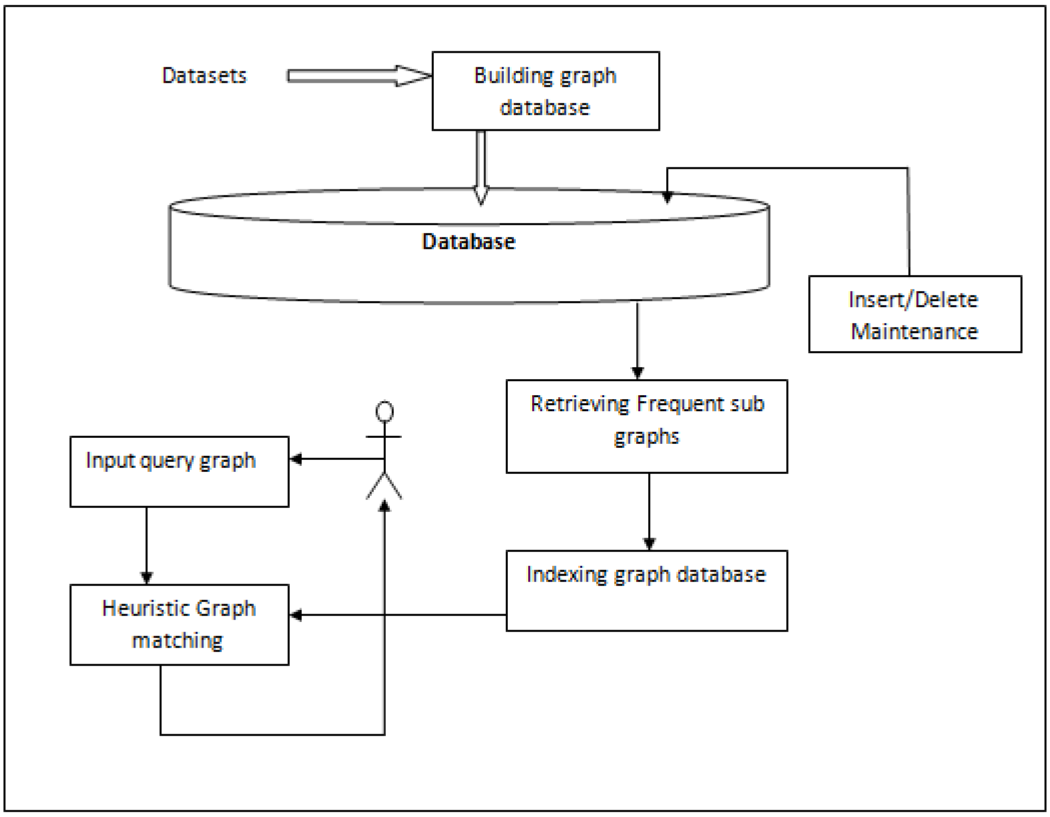


Fig 3.1 Architecture diagram

Datasets are in LINE data format, represented by a list of vertices and edges which is the one of the standard form of representation of graphs. Large graph database is built from the input datasets as shown in Fig 3.1. Then the large graph database is indexed for effective access and retrieval. The graph is further optimized by pruning the dissimilar structures. Using Heuristic Graph Matching algorithm, best similar graph to the input query graph will be retrieved efficiently. Upon addition of new graphs also, effective insert/delete maintenance is ensured by using the indexes built.

3.2 MODULES

This section outlines the various phases of the project along with the methodologies to be applied.

The project is split into various phases and modules.

Module 1 - Generation of DFS code and retrieving frequent subgraphs

Module 2 - Indexing of graph database

Module 3 - Heuristic Graph Matching

Module 4 - Database Maintenance

3.3 GENERATION OF DFS CODE AND RETRIEVING FREQUENT SUBGRAPHS

DFS coding is a set of vertices which is used for identifying the graphs. DFS coding is used as the input for generating the frequent sub graphs. DFS coding $(x, y, L_{x,} L_{y,} L_{(x, y)})$. The above DFS coding has five parts which are the vertex indexes i and j, the label of the vertex i, the label of the edge and the label of the vertex j.

Pre - processing of database has to be done for efficient process before indexing the large graph database. The frequent sub graphs algorithm uses the DFS coding as the input. As shown in Fig 3.2, Frequent sub graph are generated by scanning the Graph database and finding all the edges that can be right most extended to frequent. Then sorting all the frequent set in DFS lexicographical order. This process is repeated for all the tuples identified in the frequent set.

```
Input: c=DFS code, G=graph database, s=min
support
Output: F=frequent sub-graph set;


Algorithm :  frequent_subgraph(G, s, c, F)

Scan G once and find every  edge e so that c can be
right-most extended

If c can't be extended return

Save c U e into c;

Sort c in DFS lexicographical order

For each tuple in c

        frequent_subgraph(G, s, c U e, F);

return F;
```

Fig 3.2 Frequent sub graph algorithm

The output will contain all the fragments which occurs frequently. The fragments are represented in the DFS coding format in order to understand it easily.

3.4 INDEXING OF GRAPH DATABASE

Indexing of a graph database has to be done for efficient retrieval. In this module, the graph database is indexed using the DFS code and the frequent fragments generated. As shown in Fig 3.3, a prefix tree is built using both DFS code and frequent fragments. Prefix tree allows user to check whether a fragment or a sub-graph of a query graph is present in the graph database efficiently.

```
Algorithm : Indexing the database

Input: D-Graph database, F-Frequent Fragment
Output: Prefix Tree P
Algo: Indexing (F)
        Initialize P with dummy node as root
        For each dfscode in set F
            P.Insert (dfscode, 0)

Input: dfscode - Array representing a graph
Algo: Insert (dfscode, position)
        If position == len (dfscode) return
        If dfscode [position] does not exists in P
            Add current dfscode in P
        Insert (dfscode, position +1)
```

Fig 3.3 Indexing algorithm

Each node of a prefix tree contains a fragment which may be either frequent or redundant. The prefix tree is built in a manner such that each level contains the fragment of corresponding size. For example, level 1 in the prefix tree contains all the fragments whose size is 1. Level 2 of the prefix tree contains all the fragment which is of size 2 and the fragment should contain the previous level node as a prefix.

3.5 PRUNING THE INDEXED GRAPH

This phase works by pruning the dissimilar structures and selecting the optimal sub graph from various possible graphs. In this module, prefix tree is constructed by inserting the frequent sub-graphs retrieved. Prefix tree is mainly for identifying a super graph in which two given graphs are sub-graphs. After pruning, optimal sub graph can be obtained which can be used for heuristic graph matching.

3.5.1 PRUNING STRATEGY

In order to perform pruning efficiently, we can eliminate certain fragments i.e. the main strategy is if a fragment is not present in a prefix tree, we can efficiently avoid all the super graph of the fragment.

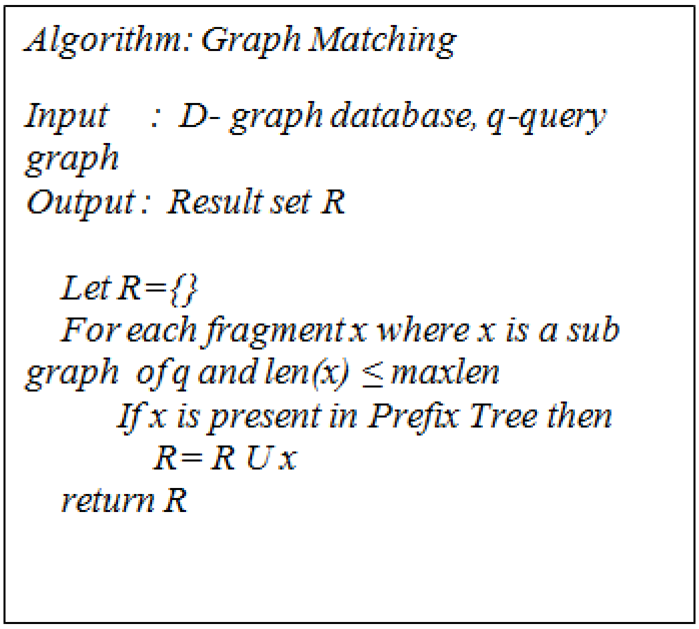
3.6 GRAPH MATCHING HEURISTIC ALGORITHM

The input query graph will be given by the user. This module uses the frequent sub graphs retrieved from the first module and the input query graph as the input. The optimal sub graph obtained after indexing is compared efficiently with the input query graph and the output gives the resultant graphs which are similar to the query graph upto a given threshold. Fragments of all length up to a given maximum length is compared with the frequent sub-graphs retrieved and efficient graph matching is done.

Graph matching can be done efficiently by eliminating (pruning) the redundant fragments i.e. if a fragment is not in the prefix tree, then its super-graph need not be compared. It can be done by using a hash table which contains the hash codes of the nodes in the prefix tree including the intermediate nodes. Whenever we find a fragment in the query whose hash code does not appear in the hash table, we need not check its super-graphs.

We can verify whether the resultant graphs really contain the query graph by performing a sub graph isomorphism test on each graph one by one.

As shown in Fig 3.4, fragments of all length up to a given maximum length is compared with the frequent sub-graphs retrieved and efficient graph matching is done.



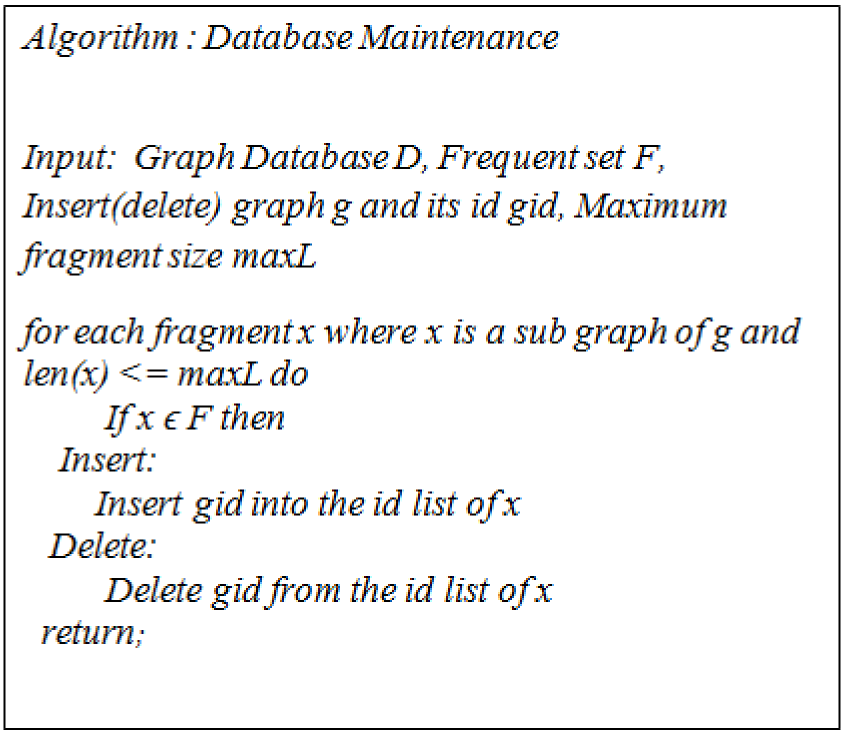
```
Algorithm: Graph Matching

Input   : D- graph database, q-query
graph
Output : Result set R

   Let R={}
   For each fragment x where x is a sub
graph of q and len(x) ≤ maxlen
        If x is present in Prefix Tree then
           R= R U x
   return R
```

Fig 3.4 Heuristic Graph Matching Algorithm

3.7 DATABASE MAINTENANCE

Insertion and Deletion of new incoming graphs has to done for incremental and updating graphs in the graph database. In this module when the new graphs are added or deleted in the existing graph database it should be updated to the Prefix tree for future indexing and accurate results.

As shown in Fig 3.5, inserting of new graphs to the Prefix tree is done by finding the frequent sub graph sets of the inserting graph using the DFS coding which is done by previous module. Once we find the frequent fragment set we compare them to the fragments in the already build Prefix tree and if they match, update the ids of the matching fragment in the id list.



```
Algorithm : Database Maintenance


Input:  Graph Database D, Frequent set F,
Insert(delete) graph g and its id gid, Maximum
fragment size maxL

for each fragment x where x is a sub graph of g and
len(x) <= maxL do
        If x ϵ F then
    Insert:
        Insert gid into the id list of x
   Delete:
        Delete gid from the id list of x
  return;
```

Fig 3.5 Database Maintenance

The same way deletion of graphs in the Prefix tree is implemented. Find the frequent set of corresponding graph which needed to deleted and check them with the frequent sets in the Prefix tree if they match then remove those fragments from the fragment set and remove the id of fragment from the id list.

CHAPTER 4
IMPLEMENTATION AND RESULTS

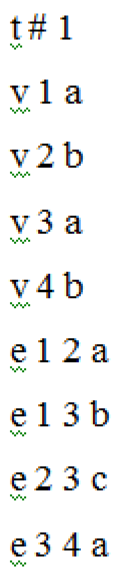
4.1 RUNTIME ENVIRONMENT

PYTHON

Python is chosen as the implementation language because of its readability and clear syntax. It is also an object oriented programming language. Python language is used for implementation of the algorithms mentioned above. Pycharm IDE is used as the coding environment. Numpy package is used for the usage of large arrays. Implementation of basic graphs, queue, prefix tree are done using python which is easy to understand and portable.

4.2 IMPLEMENTATION OF PROPOSED SYSTEM

The database contains a set of graphs which is represented by a set of vertices and edges. A sample graph is of the following form.



```
t # 1
v 1 a
v 2 b
v 3 a
v 4 b
e 1 2 a
e 1 3 b
e 2 3 c
e 3 4 a
```

t #1 is the indicator which denotes the separation of two graphs. It also tells the starting of a new graph which is used for the creation of the graph.

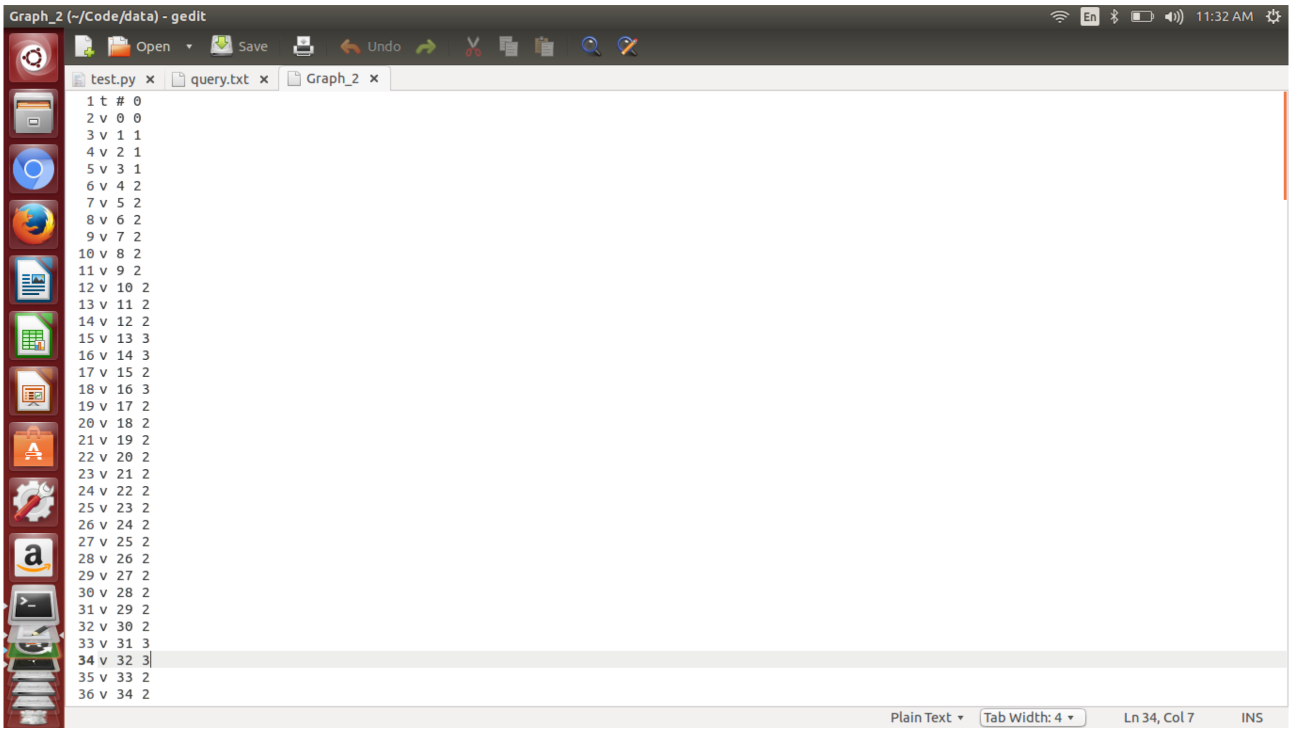
v x l_x - v denotes a new vertex with vertex id as x and the vertex label as l_x.

e u v l_uv - e denotes a new edge which connects the vertex u and v and has the label as l_uv.

Once the graph is built, the frequent sub graph from the database is obtained using the algorithm mentioned above. A prefix tree is built using the obtained frequent set. Each node in the prefix tree contains three values which are the structure of the graph, id list denoting which graphs in the database has this fragment as a subgraph and a child pointer. Once the prefix tree is built, the database is ready for the searching purpose.

The query graph is given by the user. From the query graph, all the subgraph of required size are retrieved. Starting from the smaller size sub graph, a searching on the prefix tree is done. If we find a match in the prefix tree, we proceed with the super graph of the sub graph which is matched. Else all the super graphs of the query graph fragment is pruned. This leads to the efficient pruning strategy. The output contains all the frequent sub graphs from the database and the query graph is pruned giving only the required subgraphs as the output.

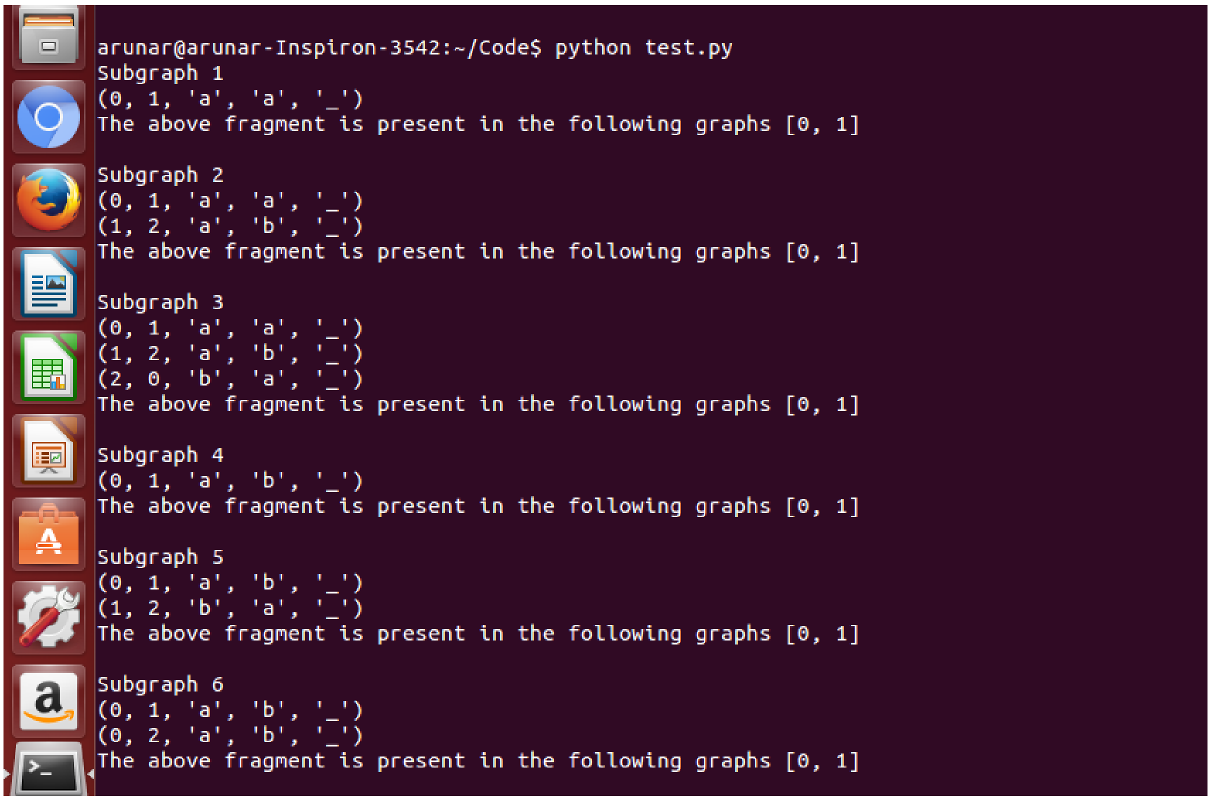
The insertion of a new graph is done by finding all the frequent sub graphs of a new graph and searching it in the query graph and adding the id of the new graph in the id list of the fragment.



```
t # 0
v 0 0
v 1 1
v 2 1
v 3 1
v 4 2
v 5 2
v 6 2
v 7 2
v 8 2
v 9 2
v 10 2
v 11 2
v 12 2
v 13 3
v 14 3
v 15 2
v 16 3
v 17 2
v 18 2
v 19 2
v 20 2
v 21 2
v 22 2
v 23 2
v 24 2
v 25 2
v 26 2
v 27 2
v 28 2
v 29 2
v 30 2
v 31 3
v 32 3
v 33 2
v 34 2
```

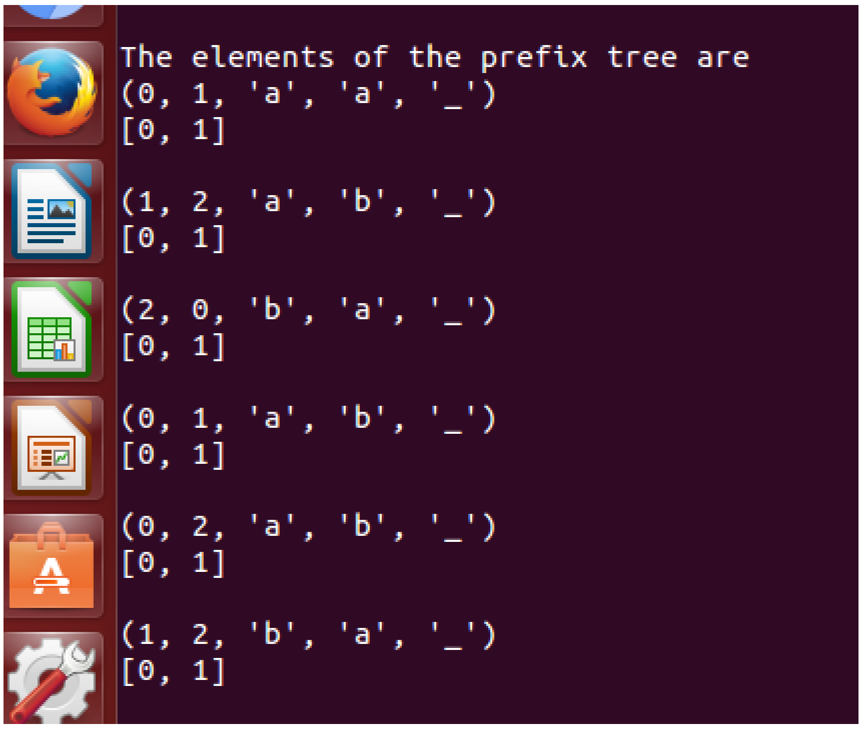
Fig 4.1 Input Database

Fig 4.1 shows the input database from which the graphs are constructed. Once the graphs are built, frequent fragments are generated using the algorithms mentioned above. Fig 4.2 shows the frequent fragments which are retrieved from the input graph database. Using the frequent fragments obtained, a prefix tree is built for efficient indexing. Fig 4.3 shows the elements of the prefix tree which is built using the frequent fragments obtained. A input query graph is given for searching. All the subgraphs are retrieved from the query graph which is shown in Fig 4.4. Then all the subgraphs of the query graph are searched in the prefix tree. Fig 4.5 shows which subgraphs are present in the prefix tree.



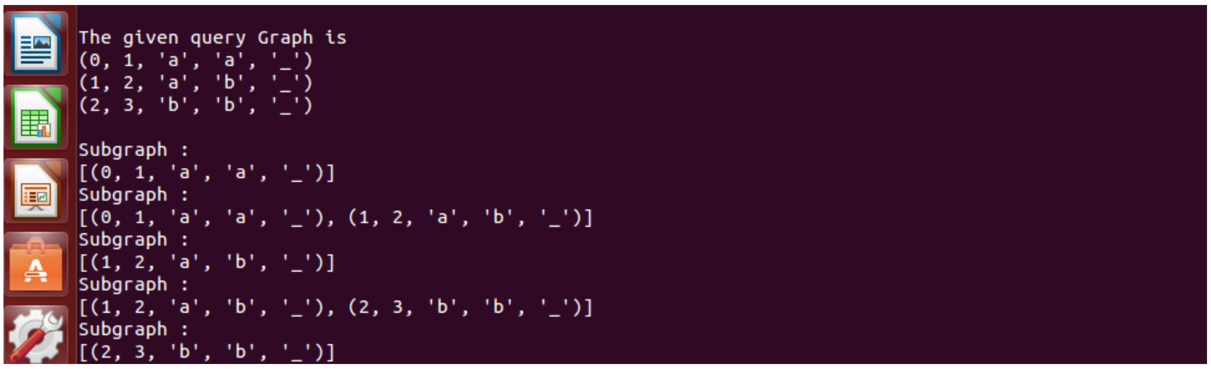
```
arunar@arunar-Inspiron-3542:~/Code$ python test.py
Subgraph 1
(0, 1, 'a', 'a', '_')
The above fragment is present in the following graphs [0, 1]

Subgraph 2
(0, 1, 'a', 'a', '_')
(1, 2, 'a', 'b', '_')
The above fragment is present in the following graphs [0, 1]

Subgraph 3
(0, 1, 'a', 'a', '_')
(1, 2, 'a', 'b', '_')
(2, 0, 'b', 'a', '_')
The above fragment is present in the following graphs [0, 1]

Subgraph 4
(0, 1, 'a', 'b', '_')
The above fragment is present in the following graphs [0, 1]

Subgraph 5
(0, 1, 'a', 'b', '_')
(1, 2, 'b', 'a', '_')
The above fragment is present in the following graphs [0, 1]

Subgraph 6
(0, 1, 'a', 'b', '_')
(0, 2, 'a', 'b', '_')
The above fragment is present in the following graphs [0, 1]
```

Fig 4.2 Output of frequent fragment algorithm



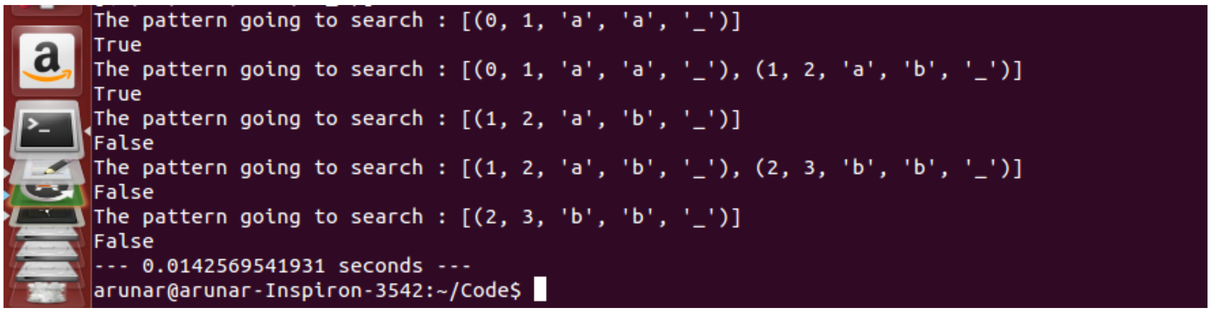
```
The elements of the prefix tree are
(0, 1, 'a', 'a', '_')
[0, 1]

(1, 2, 'a', 'b', '_')
[0, 1]

(2, 0, 'b', 'a', '_')
[0, 1]

(0, 1, 'a', 'b', '_')
[0, 1]

(0, 2, 'a', 'b', '_')
[0, 1]

(1, 2, 'b', 'a', '_')
[0, 1]
```

Fig 4.3 Elements of the prefix Tree



```
The given query Graph is
(0, 1, 'a', 'a', '_')
(1, 2, 'a', 'b', '_')
(2, 3, 'b', 'b', '_')

Subgraph :
[(0, 1, 'a', 'a', '_')]
Subgraph :
[(0, 1, 'a', 'a', '_'), (1, 2, 'a', 'b', '_')]
Subgraph :
[(1, 2, 'a', 'b', '_')]
Subgraph :
[(1, 2, 'a', 'b', '_'), (2, 3, 'b', 'b', '_')]
Subgraph :
[(2, 3, 'b', 'b', '_')]
```

Fig 4.4 Sub graphs of the given query graph



```
The pattern going to search : [(0, 1, 'a', 'a', '_')]
True
The pattern going to search : [(0, 1, 'a', 'a', '_'), (1, 2, 'a', 'b', '_')]
True
The pattern going to search : [(1, 2, 'a', 'b', '_')]
False
The pattern going to search : [(1, 2, 'a', 'b', '_'), (2, 3, 'b', 'b', '_')]
False
The pattern going to search : [(2, 3, 'b', 'b', '_')]
False
--- 0.0142569541931 seconds ---
arunar@arunar-Inspiron-3542:~/Code$
```

Fig 4.5 Searching the query graph

4.3 COMPLEXITY ANALYSIS

The time complexity for various modules is analysed.

Building the prefix tree:

The entire frequent fragment set has to be inserted into the prefix tree. So inserting one graph into the prefix tree takes O (E) where E is the number of edges in the graph. Since we have N graphs in the frequent set it requires O (N * E) time to insert all the frequent sub graphs. E is the number of edges in a graph and N is the number of graphs present in the frequent set.

Searching:

Generating all the sub graphs of a graph: The worst case time complexity occurs for the graph which is complete. The algorithm does a depth first search for the forward edge alone from all the vertices and it leads to the worst case O (2^N) when the graph is complete. For Non complete graphs it has the best time complexity of O (V * (V + E)) which is O (V^2) where V is the number of vertices in the graph and E is the number of edges in the graph.

Querying/Searching: A graph will be represented in the form of DFS code which is given for searching. The algorithm needs to traverse all the edges and check whether the traversed prefix is present in the prefix Tree which leads to O (E) where E is the number of edges in the graph.

Database Maintenance:

For inserting a new graph into the database, we have to add the id of the new graph into the prefix tree. So searching has to be done to check whether the graph is present in the prefix tree. It can be done in O (E) where E is the number of edges in the graph.

Retrieving the frequent sub graphs:

For retrieving the frequent sub graphs, initially we have to generate all the one edge fragments. For generating it we have to compare all the edges of the graph and sort it. Sorting the edges in the graph takes O (E log E). Since we have N graphs in the database it leads to O (N * E log E). Once the single edge fragments are generated it has to be extended. For extending it we need to do a BFS and find the right most edge. It takes O (V + E). Once the vertex is found we have to do a comparison whether this edge can be extended. It leads to O (V^2). It has to be done for all the edges for which we do BFS. If X is the number of one edge fragment generated then it requires O (X* (V+E) * V^2) which can be written as O (X * V^3) where V is the number of vertices in the graph.

Number of graphs	Path Based(Seconds)	Structure Based (Sec)
5	0.6947	0.5021
10	1.2350	1.0874
15	1.8821	1.5326
20	2.4672	2.0981
25	5.0209	3.4563
30	8.0543	5.3913
35	11.0013	8.9073
40	13.0163	10.2561
45	15.0821	13.4396
50	17.0984	15.2378
55	20.0372	17.4468
60	22.0834	19.5090
70	28.0382	24.4355
80	33.3742	29.9313
90	40.2645	34.0438
100	49.6834	42.3602

Table 4.1 Complexity Analysis – Runtime Comparison

Table 4.1 shows the runtime comparison of path based and structure based indexing methods. All the values are compared in seconds.

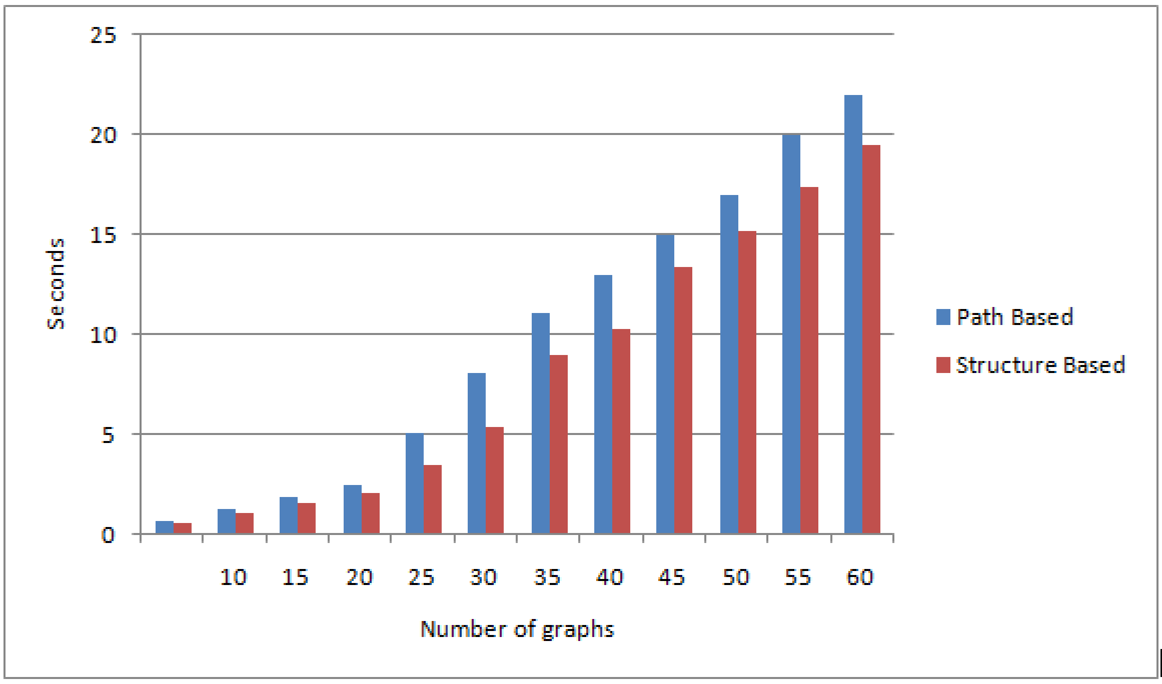


Fig 4.6 Graph Matching – Complexity Analysis

Fig 4.6 shows the graphical representation of path based and structure based indexing methods. All the values are compared in seconds.

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

Graph similarity search is one of the important problems in recent times. It can be used for various applications for finding the similar patterns. Existing classical algorithms for graph similarity are NP complete. Graph similarity search based on a given query graph is studied. Path based indexing is the existing and widely used method for indexing graph database. It traverses all possible paths and then indexes the graph using hashing. In this, we have proposed a structure based indexing approach. It indexes based on the graph structure. There will be large number of possible path available in path based indexing whereas in structure based using efficient pruning and retrieving the frequent fragments, it can be greatly reduced. In our approach, frequent fragments are retrieved from the graph database. Prefix tree is built for efficient indexing purpose. Efficient pruning is done by eliminating all the super graphs of the fragments which are not present in the prefix tree. Then given query graph is efficiently searched in the database.

5.2 FUTURE WORK

Machine learning can be applied for graph similarity finding. A machine learning algorithm can be used so that we can automate the process of identifying the similar patterns.

REFERENCES

[1] W. Zheng, L. Zou, X. Lian, D. Wang and D. Zhao, "Efficient Graph Similarity Search Over Large Graph Databases," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 4, pp. 964-978, April 1 2015.

[2] E. S. S. Dongoran, W. K. Rahmat Saleh and A. A. Gozali, "Analysis and implementation of graph indexing for graph database using GraphGrep algorithm," *Information and Communication Technology (ICoICT), 2015 3rd International Conference on*, Nusa Dua, pp. 59-64, 2015.

[3] Xifeng Yan Philip S. Yu, Jiawei Han " Graph Indexing: A Frequent Structure-based Approach," IEEE International Conference on Big Data, vol., no., pp.271,280, 2014.

[4] K. Gouda and M. Hassaan, "CSI_GED: An efficient approach for graph edit similarity computation," *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, Helsinki, pp. 265-276, 2016.

[5] G. Wang, B. Wang, X. Yang and G. Yu, "Efficiently Indexing Large Sparse Graphs for Similarity Search," in *IEEE Transactions on Knowledge and Data Engineering, vol.24, no.3, pp.440-451, March 2012.*

[6] L. Hong, L. Zou, X. Lian and P. S. Yu, "Subgraph Matching with Set Similarity in a Large Graph Database," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2507-2521, Sept1 2015.

[7] Patil, Shefali, Vaswani, Gaurav; Bhatia, Anuradha , “Graph Databases- An Overview” in International Journal of Computer Science & Information Technolo ,Vol. 5 Issue 1, p657, 2014.

[8] Xiaoli Wang, Xiaofeng Ding, Anthony K.H. Tung, ”An Efficient Graph Indexing Method”, School of Computer Science, Huazhong University of Science and Technology, P. R. China,pp. 456-901, 2010.

[9] L.Zou, J.Mo, L.Chen, M.T. Ozsu and D. Zhao, “gStore: Answering SPARQL queries via subgraph matching,” Proc. VLDB Endowment, vol. 4, no. 8, pp. 482–493, 2011.

[10] S. Zhang, J.Yang, and W. Jin, “Sapper: Subgraph indexing and approximate matching in large graphs,” Proc. VLDB Endowment, vol. 3, no. 1, pp. 1185–1194, 2010.

[11] X. Zhao, C.Xiao, X. Lin, and W. Wang, “Efficient graph similarity joins with edit distance constraints,” in Proc.IEEE 28th Int. Conf. Data Eng, pp. 834–845, 2012.

[12] X. Zhao, C.Xiao, X. Lin, Q. Liu, and W. Zhang, “A partition-based approach to structure similarity search,” Proc. VLDB Endowment, vol. 7, no. 3, pp. 169–180, 2013.

[13] X. Wang, X. Ding, A. K. H. Tung, S. Ying, and H. Jin, “An efficient graph indexing method,” in Proc. IEEE 28th Int. Conf. Data Eng, pp. 210–221, 2012.

[14] Dong Dai, Yong Chen, Kimpe, D. Ross, "Provenance-based object storage prediction scheme for scientific big data applications," IEEE International Conference on Big Data, vol no. 4, pp.271,280, 2014.

[15] Williams, D.W. Jun Huan Wei Wang, "Graph Database Indexing Using Structured Graph Decomposition", IEEE Transactions on Data Engineering, pp.976-985, 2012.

[16] K. McGarry and U. Daniel, "Computational techniques for identifyingnetworks of interrelated diseases," 14th UK Workshop on Computational Intelligence, Bradford, pp. 1-8, 2014.

[17] Hung, Benjamin WK, Anura P. Jayasumana, and Vidarshana W. Bandara. "Pattern Matching Trajectories for Investigative Graph Searches." In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*, pp. 71-79, IEEE, 2016.

[18] G. Wang, B. Wang, X. Yang and G. Yu, "Efficiently Indexing Large Sparse Graphs for Similarity Search," in IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 3, pp. 440-451, 2013.

[19] K. Aoyama, S. Watanabe, H. Sawada, Y. Minami, N. Ueda and K. Saito, "Fast similarity search on a large speech data set with neighbourhood graphindexing," IEEE International Conference on Acoustics, Speech and Signal Processing, Dallas, TX, pp. 5358-5361, 2010.

[20] Assayony, M.; Rashid, N.A., "Design of a parallel graph-based protein sequence clustering algorithm", IEEE Transactions on Information Technology,vol.3, pp.1-8, 26-28, 2012.

[21] J. Rocha, "Graph Comparison by Log-Odds Score Matrices with Application to Protein Topology Analysis," in IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 8, no. 2, pp. 564-569, 2011.

[22] H. Li and C. Liu, "Prediction of protein structures using a map-reduce Hadoop framework based simulated annealing algorithm," IEEE International Conference on Bioinformatics and Biomedicine, Shanghai, pp. 6-10, 2013.

[23] DePiero, F.W.; Carlin, J.K., "Structural Matching Via Optimal Basis Graphs”, IEEE Transactions on Pattern Recognition, vol.3, pp.449-452, 2015.

[24] F. Bai, Y. Li and H. Gao, "The Comparison of Protein Secondary Structure Based on the Graphical Representation," International Conference on Computational and Information Sciences, Chengdu, China, pp. 11-14, 2011.

[25] Changjun Wu; Kalyanaraman, A.; Cannon, W.R., "pGraph: Efficient Parallel Construction of Large-Scale Protein Sequence Homology Graphs", IEEE Transactions on Parallel and Distributed Systems, vol.23, no.10, pp.1923-1933, 2012.

[26] H. Hu; Z. Li; H. Dong; T. Zhou, "Graphical Representation and Similarity Analysis of Protein Sequences Based on Fractal Interpolation," in IEEE/ACM Transactions on Computational Biology and Bioinformatics, pp. 99-111, 2012.

[27] Jiefeng Cheng; Yu, J.X.; Bolin Ding; Yu, P.S.; Haixun Wang, "Fast Graph Pattern Matching”, IEEE Transactions on Data Engineering, pp.913-922, 2013.

[28] J. Wu and DaiChuan Ma, "Comparisons of the protein-protein interaction networks constructed from the DIP database with different version," International Conference on, Electric Information and Control Engineerin, Wuhan, pp. 236-239, 2011.

[29] Li Chen; Gupta, A.; Kurul, M.E., "Efficient algorithms for pattern matching on directed acyclic graphs" IEEE Transactions on Data Engineering, pp.384-385, 2015.

[30] M. Mernberger, G. Klebe and E. Hullermeier, "SEGA: Semiglobal Graph Alignment for Structure-Based Protein Comparison," in IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 8, no. 5, pp. 1330-1343, 2011.

[31] P. Li, L. Heo, M. Li, K. H. Ryu and G. Pok, "Protein function prediction using frequent patterns in protein-protein interaction networks," Eighth International Conference on, Electric Information and Control Engineerin, Shanghai, pp. 1616-1620, 2011.

[32] Yoo, Young Joon, TusharSandhan, Jinyoung Choi, and Sun Kim. "Towards simultaneous clustering and motif-modeling for a large number of protein family." In *Bioinformatics and Biomedicine (BIBM), 2013 IEEE International Conference on*, pp. 22-28. IEEE, 2013.

[33] Hu, Hailong, Zhong Li, Hongwei Dong, and Tianhe Zhou. "Graphical Representation and Similarity Analysis of Protein Sequences Based on Fractal Interpolation." *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 14, no. 1 (2017): 182-192, 2017.

[34] Sheng-Lung Peng, Yu-Wei Tsay, ”Using Bipartite Matching in Graph Spectra for Protein Structural Similarity”, IEEE Transactions on Biomedical Engineering and Informatics, pp.495-500, 16-18, 2013.

[35] Zheng, Weiguo, Xiang Lian, Lei Zou, Liang Hong, and Dongyan Zhao. "Online Subgraph Skyline Analysis Over Knowledge Graphs." *IEEE Transactions on Knowledge and Data Engineering* 28, vol no. 7, pp 1805-1819, 2016.

[36] Thomas, Minta, AnneleenDaemen, and Bart De Moor. "Maximum likelihood estimation of GEVD: Applications in Bioinformatics." *IEEE/ACM*

Transactions on Computational Biology and Bioinformatics 11, vol no. 4, pp. 673-680, 2014.

[37] Mernberger, Marco, Gerhard Klebe, and EykeHullermeier. "SEGA: Semiglobal Graph Alignment for Structure-Based Protein Comparison." *IEEE/ACM transactions on computational biology and bioinformatics* 8, vol no.5 , pp. 1330-1343, 2011.

[38] Wu, Wei, Anuj Srivastava, Jose Laborde, and Jinfeng Zhang. "An efficient multiple protein structure comparison method and its application to structure clustering and outlier detection." *IEEE International Conference on Bioinformatics and Biomedicine (BIBM),*, pp. 69-73. IEEE, 2013.

[39] Li, Peipei, LyongHeo, Meijing Li, Keun Ho Ryu, and GoucholPok. "Protein function prediction using frequent patterns in protein-protein interaction networks." In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, vol. 3, pp. 1616-1620, IEEE, 2011.

[40] Qiao, Rui, XiaoleiZhong, Ling Zhang, and Heng He. "Graph Pattern Matching through Model Checking.", *8th International Conference on Database Theory and Application (DTA)*, pp. 1-5. IEEE, 2015.

[41] Fairey, Jason, and Lawrence Holder. "StarIso: Graph Isomorphism Through Lossy Compression." *Data Compression Conference (DCC),* pp. 589-589. IEEE, 2016.

[42] Zhong, Haojian, Lida Xu, Cheng Xie, Boyi Xu, Fenglin Bu, and HongmingCai. "A Similarity Graph Matching Approach for Instance Disambiguation." In *Enterprise Sysstems (ES), 2016 4th International Conference on*, pp. 21-28. IEEE, 2016.