AN EFFICIENT DATA STORAGE AND RETRIEVAL MODEL FOR SCIENTIFIC RESEARCH APPLICATIONS

Abstract - Analysing scientific datasets can find new correlations to prevent diseases, provide personalized medicines and help decision making in scientific research. These scientific datasets usually have similar jobs that are frequently applied to them by different users. Generally, scientific datasets are large, complicated and are unstructured data which require fast processing. In order to increase the performance of the existing Hadoop and Map Reduce algorithm, it is necessary to develop an algorithm based on the type of dataset and requirements of the job. Genetic, DNA, Amino acid sequence data, Protein-Protein interaction graphs are the examples of the large, unstructured, complicated biological data which have a huge sequence of non-reliable and non-relational letters. The constant development in biological fields like Microscopy and Next Generation Sequencing technology led to the exponential increase in the size of datasets nearly in Terabytes (TB) and Petabytes (PB) resulting in millions of data files. In order to make meaning of this raw data, scientists must be provided with ease of use technology which will efficiently store, retrieve, organize, visualize, annotate and crosslink these data. As the scientific data generally have more interrelated attributes it is intuitive to represent the scientific data as graphs in graph databases. It has become increasingly important to retrieve graphs in the graph databases that matches the query graph. Our proposed graph storage and retrieval model efficiently finds the similar graphs in the graph database which gives helps to give better insights on large data sets.

1 INTRODUCTION

Proteins are the workhorses that facilitate most biological processes in a cell, including gene expression, cell growth, proliferation, nutrient uptake, morphology, motility, intercellular communication and apoptosis. Protein analysis mainly focussed on single proteins, but because of the proteins interact with other proteins for majority of functioning, they should be studied in the context of their interactions partners to fully understand how proteins interact with each other and identify biological networks which is vital 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][3]. The protein interactions are important in studying the interatomic system of the living cell and helps in the analysis of different diseases like Cancer, Creutzfeld-Jacob, Alzheimer's disease [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 hydrophobic bonding, van der Waals forces, 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 biochemical cascades and disease pathogenesis, as well as provide putative new therapeutic targets [19].

Graph databases makes use of graph structures to represent the data [5]. Graph databases have three

basic entities such as node, edge and properties. The edge in the graph databases determines the interrelationship either between two nodes or between the node and the property. The edges may be directed, undirected or weighted, which determines the interactions or relationship between the entities.

Storing data in this format helps us to answer open wide questions and semantic queries. The queries that are analytics of associative or contextual nature are called as semantic queries. These queries helps us to deliver precise results about some open wide questions which can be answered only using pattern matching and digital reasoning. There by meaning patterns emerge when we analyse the interrelationship between the nodes, properties and the edges.

In certain type of data such as molecular data or protein data have characteristic labelled vertices, in such cases relational databases are less effective because of the type of the data used are interrelated and interdependent. To handle such data sets graph databases will be the most appropriate solution [7].

Graph databases scales well as they don't involve complex joins of different normalized table as in the relational database management system [4]. 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.

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 [7]. 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.

In graph database management system querying will be optimal only if the data is stored in the effective way. As in the advancements in the technology effective data storage model for graph store should be proposed for distributed graph storage and other graph storage models to support parallel graph processing engines by some multi-core processors [1].

Distributing the given graph into various replication nodes is one of the major issue faced in the graph database management system. Horizontal scalability is the important aspect in the graph database management system. Horizontal scalability determines increasing the number of slave nodes directly determines the increase in the scalability. Graph partitioning being a NP hard problem, there are different heuristic approaches being proposed for graph distribution.

Streaming queries are used when we are dealing with continuous vales that is the data is streaming, for example finding the top – k tweets from twitter, as the twitter database is constantly updated with tweets, it is tough to find the top – k tweets efficiently. The streaming queries are also called as the long running queries or continuous queries.

There are certain applications where it is not possible to hard code the entire conditional statements as in natural language processing, it is not effective to build a natural language processing engine completely, and on the other hand we could implement a machine learning natural language

processing engine [29]. The graph databases helps in applying the machine learning techniques on the graph in a scalable way. In recent times there has been a lot of interest around machine learning in large graphs.

The visualization of large and dynamic graphs is the major area of research interest because the graphical representation of the protein sequence can help to visualize the complex relationship and also numerically describes the similarity between the protein sequences [12].

2 BACKGROUND AND OBJCTIVE

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. To develop graph matching model, with heuristics for scientific data to effectively find graph similarity search over large graph datasets.

Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, and information privacy. The constantly developing high content in microscopy and next generation sequencing technologies routinely produce experiment datasets in the terabyte (TB) or petabyte (PB) range resulting in millions of data files that can vary from simply numbers to signals and images. To draw the conclusions from such increasingly complex and large scale data sources [2], the scientific community must be provided with simple to use methods to retrieve, analyze, visualize, annotate, and crosslink these data sources on a common platform in an efficient manner. Our proposed work is to do efficient data storage and retrieval model for scientific research applications as of suggested as an architecture diagram in figure 3.1

3.1 OBJECTIVES

To develop a heuristic graph matching model to effectively find graph similarity search over large graph datasets.

To develop a data storage model for effectively store graphs for easy querying and retrieval.

There are three modules in the proposed system

1. Graph Comparison Module
2. Graph Storage and Indexing Module
3. Optimal Sub-graph Selection Module

Commented [j1]: Storage and indexing

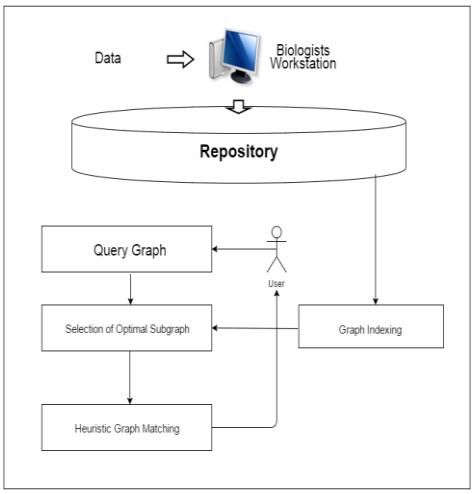


Figure 3.1 Architecture Diagram

3.3 MODULES

3.3.1 GRAPH COMPARISION MODULE

Algorithm:

Input : Query graphs and input graph

Output: Returns the matching of the query graph and the input graph

$Algorithm\ graph\ Match\ (Gq, G0)$

$for\ every\ node\ \in \{Gq, G0\}$

$list1 \leftarrow append\ (\ compute << x, y >, r >) for\ Gq$

$list2 \leftarrow append\ (\ compute << x, y >, r >) for\ G0$

$plot < x, y > in\ 2D\ plane$

$draw\ a\ cirle\ around\ it\ with\ radius\ r$

$a, b = order(list1), order(list2)$

$return\ sequential\_compare(ordered1, ordered2)$

$end\ graph\ Match$

$Algorithm\ order(list)$

$if\ list.size > 1$

$a \leftarrow order(list[1 \ldots \frac{size}{2}])$

$b \leftarrow order(list[\frac{size}{2} + 1 \ldots size])$

$merge(a, b)$

$end\ order$

For Instance, for a sample gxl file <<enzyme_1.gxl>>
The visualized graph is shown in figure 3.2 and the graph representation is shown in the figure 3.3

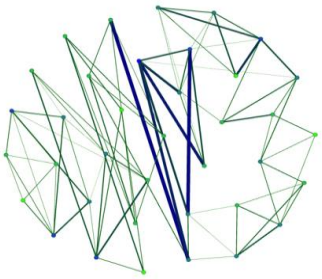


Figure 3.2 Graph visualization

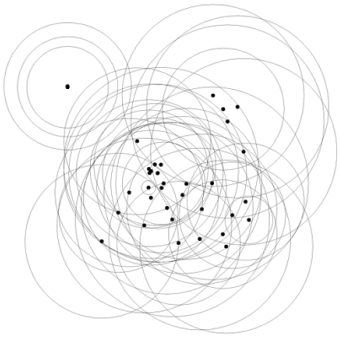


Figure 3.3 Graph in 2D Representation

3.3.2 GRAPH STORAGE AND INDEXING MODULE

Commented [j2]: Storage and ...

Add few lines of pseudo code for storage of graph

Algorithm:

***Input*:** N graphs from the graph database

Output: Persistent Quad tree index on top of the graphs

$Algorithm\ Graph\ storing\ and\ Indexing$

$for\ each\ F <- GXL\ file\ in\ G$

$for\ each\ node\ N <- in\ F$

$insertIntoPersistentQuadTree(N\ \{r, < x, y >\})$

$for\ each\ node\ in\ the\ graph\ databse\ G$

$position\ the\ node\ 'n'\ in\ the\ 2D\ plane.$

$recursive\ split\ the\ plane \leftarrow$
$until\ there\ is\ atleast\ one\ node$

$end\ graph\ indexing$

3.3.2.1 POSITIONING THE NODE IN 2D PLANE

The position of each node, in the 2D plane (X, Y) is

defined by the node sequence which is basically an Amino acid sequence,

Table 3.1 Amino acids and its identifiers

Amino Acid	Identifier
Glycine	G
Proline	P
Small and hydrophobic	A, V, L, I, M, F, W
Hydroxyl and amine amino acids	S, T, N, Q
Charged amino – acids	D, E, R, K
Histidine and Thyrosine	H, Y

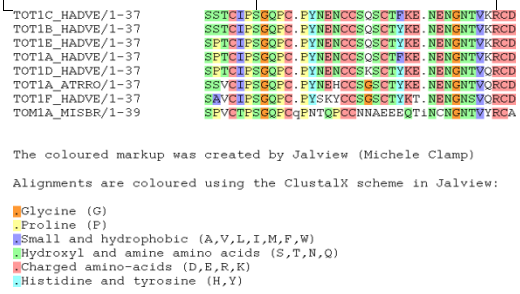


Figure 3.4. Node – Amino Acid sequences

3.3.2.2 GENERATING THE RADIUS OF EACH NODE

The radius of each node, is defined by the attribute distance to all the other nodes, to which the current node is connected. Say the node A is connected to the nodes B, C, D, E, F with the corresponding distances V, W, X, Y, Z. The radius of node A will be determined by the values of V, W, X, Y, Z

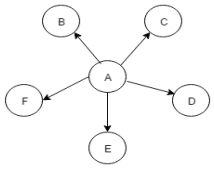


Figure 3.5 Generating radius of each node

3.3.3 OPTIMAL SUB – SET OF GRAPH SELECTION MODULE

The optimal sub – set of graph selection module makes effective use of quad tree. A quad tree is a tree data structure in which each internal node has exactly four children. Quad trees are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions.

There may be some graphs which don't make an exact match. To fix this we'll fix a threshold level. Say there are 40 sequences in a graph and the result from the quad tree (figure 3.6) indexing algorithm gives the graphs (for example enzyme_2.gxl results in 36 sequence match, enzyme_5.gxl results in 31 sequence match, enzyme_10.gxl results in 5 sequence matches and so on till the list of GXL's which matches at least once). We'll fix a threshold level of node mismatch, then pick all the sub set of graphs which lies above the threshold level.

Then the selected sub set of graphs are compared parallel in distributed Hadoop clusters [9], using our heuristic graph matching algorithm which runs in N log N time.

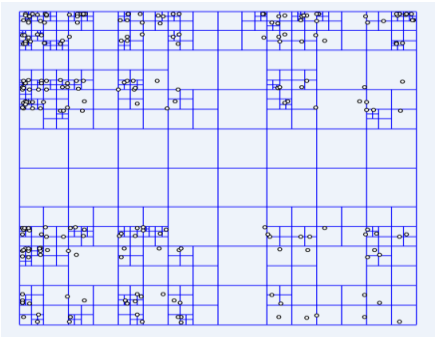


Figure 3.6 Quad tree

Algorithm:

Input: Set of graphs {G1, G2, G3 … Gn}, Graph index I, query graph Q

Output: Optimal sub graphs that may match the given query graph Q

Algorithm optimal subset selection

Let $s \leftarrow \{empty\}$

for each node in query graph Q do

add generated X, Y *co − ordinates to S*

return depthSearch($A \leftarrow$ *root node of quad tree,* S)

end optimal subset selection

Algorithm depthSearch$(node, S)$

if node $\rightarrow TL \notin$ *from S*

prune top left child

if node $\rightarrow TR \notin$ *from S*

prune top right child

if node $\rightarrow BL \notin$ *from S*

prune bottom left child

if node $\rightarrow BR \notin$ *from S*

prune bottom right child

return $dfs(a \rightarrow TL, s)$

$\cap \quad dfs(a \rightarrow TR, s)$

$\cap \quad dfs(a \rightarrow BL, s)$

$\cap \quad dfs(a \rightarrow BR, s)$

end depthSearch

As the graph database, has a large number of graphs to be compared for similarity we are in need of a mechanism to prune those graphs that won't match the given query graph q.

For example, consider three sub-graphs a, b, c which are plotted by means of the above graph comparison algorithm.

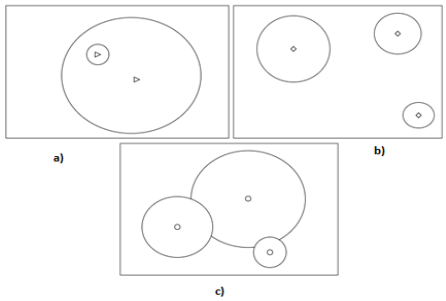


Figure 3.7 Graph Plot

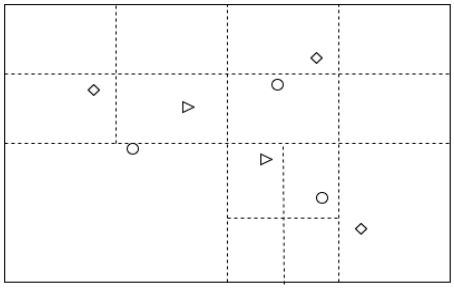


Figure 3.8 Pruned Sub set graphs

IMPLEMENTATION OF PROPOSED WORK

4.1 GRAPH MARKUP LANGUAGE

Graph ML is the graph storage and portable file format that is developed by the graph drawing community to define a common format for exchange graph structure data. Graph ML is based on the XML syntax including directed, undirected, mixed graphs, hyper graphs, and applications-specific attributes. Graph ML contains various XML elements such as graph, node, and edge. Every node element should have an id attribute, edge element should have source and target attributes which represents the id of the nodes. Here is a sample Graph ML with two nodes and one edge between the nodes.

$<?xml\ version = "1.0"\ encoding = "UTF-8"?>$

$<graphml>$

$<graph\ id = "G"\ edgedefault = "undirected">$

$<node\ id = "n0"/>$

$<node\ id = "n1"/>$

$<edge\ id = "e1"\ source = "n0"\ target = "n1"/>$

$</graph>$

$</graphml>$

There are some additional features in graph ML which allows the users to specify whether the edges are directed or undirected, and to specify additional data in the vertices or edges.

4.2 GRAPH STORAGE FORMAT

Graph exchange Language (GXL) is also derived from extensible markup language (XML) and the syntax of GXL is defined by XML document type definition (DTD). GXL is defined as the standard exchange format for graphs and it offers an adaptable and flexible means to support interoperability between graph-based tools.

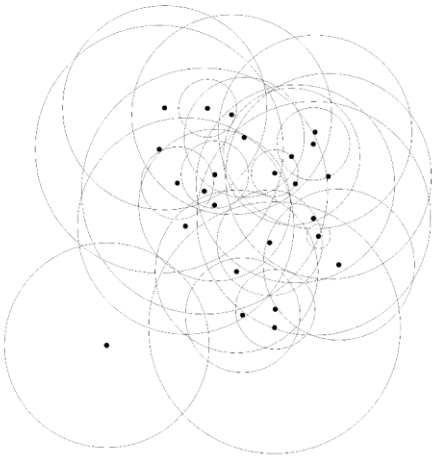
GXL was primarily developed for the interpretability and to enable exchange of graphs between software reengineering tools and components, such as code extractors (parsers), analysers and visualizers.

Our input data set is a protein – protein interaction graph in which every protein node contains an Amino acid sequence and the interaction between the protein nodes is represented as edges in the graph and the weight in the edge denotes the level of interaction between the two nodes. For efficient comparison of graphs, it is intuitive to map the nodes as pair of $< x, y >$ point in the $2D$ plane, and the edges that are directly incident on the point forms the circle with the radius r. The value of r is computed from the distance of every neighboring node from that particular node.

For every Amino acid sequence that is present in the graph G, a point in the $2D$ plane is generated using the rolling Rabin Karp rolling hash algorithm.

$$H = c1\ a^{k-1} + c2\ a^{k-2} + c3\ a^{k-3} + \cdots + ck\ a^{0} \quad \text{--- (4.1)}$$

Where a is a constant and $c1, c2, \ldots ck$ are the input characters as in the equation 4.1. In order to avoid manipulating with large H value, the computation is done congruent modulo n. This Rabin Karp implementation requires the multiplication of two k bit numbers, integer multiplication is an $O(k \log k 2^{O(\log k)})$. For instance, the node point for the Amino acid sequence *"TEDLPPAR"* may be something like $< 55, 83 >$. Bases on the interaction of the current protein node with the other node in the given interaction graph a value r is computed and is drawn as the circle radius around the node.



4.1 Graph storage format

4.3 QUAD TREE INDEXING

A quad tree is a tree data structure in which every node has exactly four children namely top left (TL), top right (TR), bottom left (BL) and bottom right (BR). Quad tree are most often used in the places where we divide a 2D space by recursively sub dividing it into four quadrants or regions. There is no definite defined shape for a region, the region may be square, rectangular or can take any arbitrary shape. A quad tree with the depth of N may be used to represent an image consisting of $2^n * 2^n$ pixels where each pixel have a value of either 0 or 1. A quad tree is constructed from the 2D representation of all graphs in the graph database. The constructed quad tree indexes the Amino acid sequences that are present in the protein – protein interaction graphs.

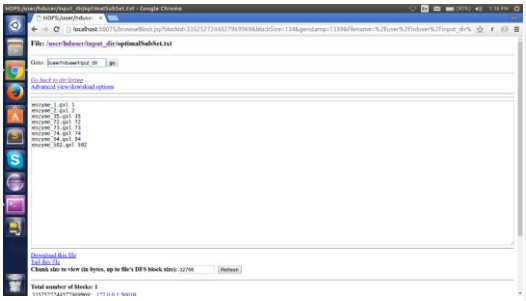


Figure 4.2 Pruned graphs output

4.4 OPTIMAL SUBSET OF GRAPHS

Depth – first traversal of the nodes of the quad tree by maintaining the state variables of the optimal sub set of graphs that matches the given query graph greater than a threshold limit H. Thereby the state variable in the root of the quad tree will contain the optimal sub-set of graphs that mostly matches the given query graph q. The result of the optimal sub-set selection algorithm is a sub-set of graphs say $G1$, G2, G3 … GK from the graph database with N graphs where the node of the k graphs with the given query graph q is greater than the threshold H.

Where, N is the total number of graphs in the graph database, K is the size of the optimal subset, and H is the threshold value.

Commented [j3]: What is this value ? add results for diff values and inference.....

4.5 GRAPH COMPARISION HEURISTIC ALGORITHM

The graph comparison heuristic algorithm takes two graphs $G1$ and $G2$ and returns whether both the graphs is a matching or not matching. The graph comparison heuristic algorithm proceeds as follows, let us assume there are X nodes in the given graph, then for each node $N1, N2, \dots NX$ in the each graph a pair of

$< P_i, R_i >$ Here P_i denotes the $< X - coordinate, y - coordinate >$ in the 2D plane and R_i denotes the radius of the circle generated for the i^{th}node. Thus a mapping for each node is generated as

$[N1, N2 \dots Nx] \Rightarrow [< P1, R1 >, < P2, R2 > \cdots < Px, Rx >]$ --- (4.2)

The generated list of point-radius pair is arranged in an order by sorting by merging algorithm which can be performed in $O(X \log X)$ time. Then the ordered list of each graph is compared in linear time to check if it is a match or a mismatch.

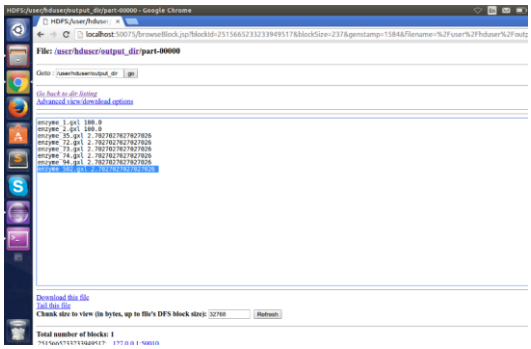


Figure 4.3 Graph comparison result

is no interdependency of data. As in the functional programming paradigm it can be computed parallel in Hadoop Map Reduce distributed framework. The Map Reduce model is one of the attractive models for parallel data processing in distributed frameworks. The Map Reduce framework has a proven scalability because a job in the Map Reduce model is partitioned into numerous small tasks and they are running on multiple heterogeneous computing environments [17]. The directed acyclic graph formation can be visualized in the diagram given below in figure 4.2

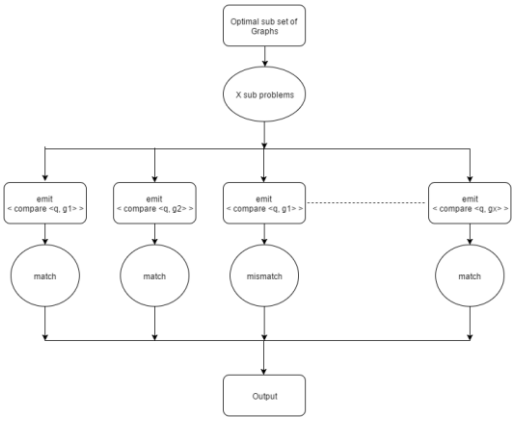


Figure 4.4 Parallel Execution flow diagram

4.6 PARALLEL EXECUTION IN THE DISTRIBUTED SYSTEM

The optimal sub-set of graph selection algorithm selects a sub set of graphs that nearly matches the given query graph Q. For each graph in the optimal sub set of graph selection algorithm, the heuristic graph matching algorithm is to be applied.

Eg: $< q, [G1, G2, G3 \dots Gx] >$ will result in $< q, [0, 1, 1, \dots 0] >$ where 1 denotes the query graph q makes an exact match with graph in the index i and 0 denotes a mismatch. This problem is similar to a list processing problem where a list of independent problems to be processed is given as input and there

5 RESULTS AND ANALYSIS

5.1 PRUNING STRATEGY

The data sets is divided into three disjoint subsets, which can be used for training, validation, and testing algorithms. The protein data set consists of graphs representing proteins. The graphs are constructed from the Protein Data Bank and labeled with their corresponding enzyme class labels from the BRENDA enzyme database. The proteins are converted into graphs by representing the secondary structure elements of a protein with nodes and edges of an attributed graph. Nodes are labeled with their type and their amino acid sequence (e.g.

Commented [j4]: Briefly state about the data set used , its size, structure of each sequence and corresponding grapg structure , its storage how it will be indexed etc ….

TFKEVVRLT). Every node is connected with an edge to its three nearest neighbours in space. Edges are labelled with their type and the distance they represent in angstroms.

The optimal sub set selection algorithm employs an efficient pruning strategy using the quad tree index. Spatial arrangements in the quad tree prunes effectively with cost of updating the index is minimal. In order to improve the pruning efficiency we have adopted the early stop strategy, where pruning stops at once we have found the $(\tau + 1)$ mismatching Amino acid sequences, which are represented as a point in the 2D plane as $< x\ co - ordinate, y\ co - ordinate >$ and the value of τ denotes the threshold limit defined as in the build configuration.

5.1.1 PERSISTENT QUAD TREE CONSTRUCTION

Assume each data set contains N nodes and there are M such data sets available, then the total number of nodes is NM, In persistent quad tree construction the quad tree is constructed only once and it is stored in the secondary storage medium and is loaded into memory whenever required. The persistent quad tree will be reconstructed only when the data set changes. Thereby the time taken for the construction of persistent quad tree will be given in equation 5.1

$$T_{total} = NM\ log_2(NM) + Q * log_4(NM) \quad \text{-- (5.1)}$$

Where, N denotes the average number of nodes in a dataset, M denotes number of such data sets and Q denotes the number of queries

5.1.2 ON – DEMAND QUAD TREE CONSTRUCTION

Let us assume that the entire graph database contains NM nodes, but we are eliminating certain graphs by effectively checking a special node in $O(M)$ time. The special node will be the node with the maximum degree. Thereby reducing the number of graphs to P and $P \leq M$. Thus after this reduction step the number of nodes to be compared will be comparatively low $NP \leq NM$. Then in on – demand quad tree construction, a quad tree is constructed for NP nodes rather than NM nodes. In on – demand quad tree construction, the index is recomputed every time for new query. Then the time complexity for construction of such on – demand quad tree is shown in equation 5.2

$$T_{total} = Q * NP\ log_2 NP + Q\ log_4 NP \quad \text{--- (5.2)}$$

$$and\ NP \leq NM$$

Where, N denotes the average number of nodes in a dataset, M denotes number of such data sets, P denotes the reduce number of data sets and Q denotes the number of queries. The quad tree construction cost $NP\ log_2 NP$ is multiplied by a factor Q as for Q queries, the on – demand quad tree construction will construct Q quad trees.

5.2 COMPARISION OF PERSISTENT AND ON – DEMAND QUAD TREE

The below table describes the cost for the construction of persistent quad tree and the on – demand quad tree for varying parameters of Q number of queries, NM number of nodes in persistent quad tree and NP denotes the number of nodes in the on – demand quad tree along with the details of the cost to compute the tree in both cases.

Table 5.1 Query cost analysis in Quad Tree

Q	NM	NP	T_p	T_q
100	50000	50000	170800	17012731
		30000		9655149
		10000		2822492
		100		11841

Commented [j5]: Simulated results for larger graphs

Add another table on indexing time / space and scalability

500	50000	50000 30000 10000 100	173522	85063655 48275748 14112462 59209
1000	50000	50000 30000 10000 100	176925	170127310 96551497 28224924 118418

Where Q the number of queries is, NM is the number of nodes in the persistent quad tree, NP is the number of nodes in the on demand quad tree, T_p is the total time for constructing persistent quad tree, T_d is the total time for constructing on demand quad tree.

From the comparison table 5.1 it is clear that on – demand works better when the number nodes for construction is very low and the number of queries is comparatively less. In every other cases persistent quad tree construction works effectively.

5.3 COMPARISION OF GRAPH SIMILARITY

Given two graphs $G1, G2$ the graph matching heuristic algorithm runs in $O(n \log n)$ time to find whether two graphs is similar or not. Whereas the state of art graph isomorphism verification algorithm like VF2 and QuickSI can be adopted to prune the mismatching structures. The mismatching structures can be effectively found by using the inverted index of the sub structures in the graph G. Thereby the first three steps of finding the mismatching structures take linear time, finally if the graph still remains after eliminating the mismatched structures then it invokes a state expansion based sub graph isomorphism verification algorithm which is an exponential algorithm.

Table 5.2 Comparison of complexity analysis in proposed and state of art dynamic partition [30]

Commented [j6]: Mention the name of existing methods base paper method….with which comparison is done…

N	M	$T_{proposed}$	$T_{existing}$
100	100	332.19	301.02
	95		317.95
	90		337.66
	85		692.89
	80		3265.25
500	500	2241.446	1501.0
	490		1527.66
	480		4565.25
	470		193161.0

The comparison of the state of art technique and our proposed model is discussed in the table 5.2

Where N denotes the number of nodes in the graph, M denotes the number of Matching structures, $T_{proposed}$ and $T_{existing}$ represents the time complexities of proposed and state of art techniques respectively. From the table 5.2 it is clear that our proposed method out performs the existing state of art techniques when the number of mismatching structures is large.

The comparison of run time analysis of Graph storing, indexing and graph matching algorithm in Ubuntu 14.04 LTS with Intel core I3 processor with 2 GB of RAM is shown in the figure 5.1. From the figure it is clear that Graph comparison heuristic algorithm runs good in all the cases.

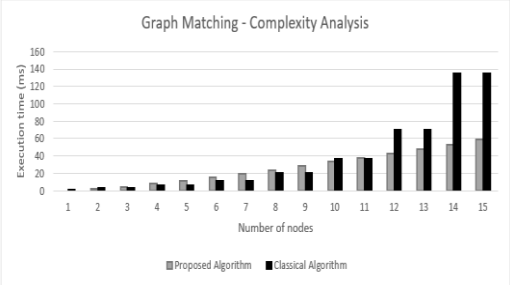


Figure 5.1 Graph Matching – Complexity Analysis

6 CONCLUSION

Graph similarity searching has always been a difficult task to deal with. Classical algorithms for finding the graph similarity searching is NP complete. In Graph similarity search, a query graph comparison a large graph database is studied, by considering the limitations of the existing approach an efficient approach for storing and indexing the graph data is proposed by a systematic method of flattening every graph into a planar structure and quad tree construction for indexing the data. The pruning the speed up by effective quad tree pruning framework. In our approach we use the quad tree for indexing the nodes in the graph database which is constructed only once and updated whenever the dataset changes, in our second step optimal sub set of graph selection algorithm is defined which returns a list of graphs that nearly matches the given query graph Q. Our final step of graph comparison is optimized by executing our graph matching heuristic polynomial algorithm in parallel in the hadoop Map Reduce framework. A learning algorithm can be introduced so that the machine automatically learns the graph matching patterns of patient with particular disease.

REFERENCES

[1] Assayony, M.; Rashid, N.A., "Design of a parallel graph-based protein sequence clustering algorithm", IEEE Transactions on Information Technology, vol.3, no., pp.1-8, 26-28, 2012.

[2] 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.

[3] DePiero, F.W.; Carlin, J.K., "Structural Matching Via Optimal Basis Graphs", IEEE Transactions on Pattern Recognition, vol.3, no., pp.449-452, 2015.

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

[5] 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.

[6] 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, 2012.

[8] Haifeng Fan, Haixing Wang, "Predicting Protein Sub cellular Location by AdaBoost.M1 Algorithm", IEEE Transactions on Biomedical Engineering and Informatics, pp.3168-3171, 2013.

[9] 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.

[10] 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.

[11] Jaroslaw Zola, ”Constructing Similarity Graphs from Large-scale Biological Sequence Collections”, IEEE Transactions International Parallel & Distributed Processing, pp.500-507, 2014.

[12] 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.

[13] 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.

[14] 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.

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

[16] Liang Hong, Lei Zou, Xiang Lian, and Philip S. Yu, ”Subgraph Matching with Set Similarity in a Large Graph Database”, IEEE transactions on knowledge and data engineering, vol.27, no.9, pp.2507-2521, 2015.

[17] 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.

[18] 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, 2015.

[19] Minta Thomas, Anneleen Daemen, and Bart De Moor,”Maximum Likelihood Estimation of GEVD: Applications in Bioinformatics”, IEEE Transactions on Computational Biology and Bioinformatics, vol.11, no.4, pp.673-680, 2014.

[20] 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.

[21] Nisar, M.U.; Fard, A.; Miller, J.A., "Techniques for Graph Analytics on Big Data", IEEE Transactions on Big Data, pp.255-262, 2013.

[22] Peipei Li, Lyong Heo, Meijing Li, Keun Ho Ryu and Gouchol Pok, ”Protein Function Prediction Using Frequent Patterns In Protein-Protein Interaction Networks “, IEEE Transactions on Fuzzy Systems and Knowledge Discovery, pp.1616-1620, 2012.

[23] Pengyi Yang, Paul D. Yoo, Juanita Fernando, Bing B. Zhou, Zili Zhang, and Albert Y. Zomaya, ”Sample Subset Optimization Techniques for Imbalanced and Ensemble Learning Problems in Bioinformatics Applications”, IEEE Transactions on Cybernetics, vol.44, no.3, pp.445-455, 2014.

[24] 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.

[25] R. Bhardwaj, N. Mishra and R. Kumar, "Data analyzing using Map-Join-Reduce in cloud storage," *International Conference on, Parallel, Distributed and Grid Computing,* Solan, pp. 370-373, 2014.

[26] Sagar Patel, Hetalkumar Panchal and Kalpesh Anjaria, "DNA Sequence analysis by ORFFINDER& GENOMATIXTool: Bioinformatics Analysis of some tree species of Leguminosae Family", IEEE International Conference on Bioinformatics and Biomedicine Workshops, pp.922-926, 2012.

[27] 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.

[28] S. G. Manikandan and S. Ravi, "Big Data Analysis Using Apache Hadoop," *International Conference on, IT Convergence and Security,*Beijing, pp. 1-4, 2014.

[29] S. Vaiwsri, A. Prachumwat, S. Ngamsuriyaroj and A. Srisuphab, "Predicting shrimp protein-protein interactions and gene ontology terms using association rule and semantic similarity calculation," *Computer Science and Engineering Conference, International,* Khon Kaen, pp. 278-283, 2014.

[30] Weiguo Zheng; Lei Zou; Xiang Lian; Dong Wang; Dongyan Zhao, "Efficient Graph Similarity Search Over Large Graph Databases", IEEE Transactions on Knowledge and Data Engineering, vol.27, no.4, pp.964-978, 2015