HEURISTIC APPROACH FOR SUB GRAPH PATTERN SEARCH USING TREE TECHNIQUE

[1]Nilofar Rafik Chaus, [2]Prof.Prashant Jawalkar
Computer Engineering,
BhivarabaiSawant Institute of Technology & Research
Wagholi, Pune-421207

Abstract— Due to drastic increase in digital data searching process takes higher space and time complexity. Most of the searching techniques are existed, each one of them use one or other method to achieve better performance grade. Many methodologies uses tree inclusion technique to improve the searching process where as some other includes graph pattern formation to search the given queries efficiently. This paper concentrates on combining graph approach and tree technique to search the required query more efficiently. Paper put forwards an idea of using the graph database to store the pre created hyper graph entities to create the weighted based entities eventually that eases the process of searching. Proposed technique identifies the weight of each entity to be search and based on the weight it creates M-tree that is sorted according to node weights. These node weights eventually create the clusters as the answers for the searched query.

Index Terms— Tree, hyper graph, neo 4j, clusters, Pattern

I.INTRODUCTION

A query is an alternate term in question. Computer queries are processed by software when queries are forward to a computer system. Each time we look or search for something using a search engine, we perform search queries. The search query is performed by lots of peoples numerous times a day. There are 3 main categories of search queries named as Navigational Search Queries, Transactional Search Queries and Informational Search Queries. Navigational Search Queries are seeking for a particular website or webpage. Informational Search Queries cover a broad topic for which thousands of results display. Transactional Search Queries used for purchasing purposes, to reflect the expectation of user to play out a specific action. A query used to find out information using a web search engine is called as a web search query. Web search queries are plain content or hypertext, and it is different from standard query language. A graph data structure contains two components nodes and edge. Nodes are a finite vertices set and edge is a finite ordered pairs set. The graph edge contain value/weight/cost. Figure 1 shows a simple graph representation.

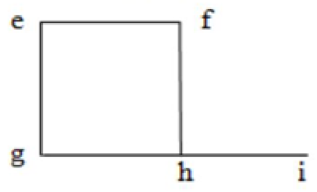


Figure 1: Graph Representation

In the above graph Vertices = {e, f, g, h, i} and Edges= {ef, eg, fh, gh, hi}

The most popular representation of a graph is in the form of Adjacency Matrix, Incidence Matrix and Adjacency List. The selection of graph representation depends upon the type of operations to be performed. The main basic operations of graph are add vertex, add edge and display vertex. Many real life applications are represents by graph like networks, social networks.

A data structures of tree is consist of various collections of nodes connected by edges (directed or undirected). Every Tree data structure node contains a value and it is a nonlinear type of data structures. Tree data structure contains one node called root and zero or more sub-trees. Figure 2 represents tree data structures: In the figure 2 ‘a’ is a parent of ‘b, c, d’ and ‘b’ is a child of ‘a’ and parent of ‘e and f’. In a tree data structure each node of a tree contains arbitrary number of children. Leaves or External nodes are nodes without child and a node with at least one child is called internal nodes. Same parent’s nodes are called siblings. Total number of edges from the root to the node is called as Depth of a node and node height is evaluated as no. of edges from the node to the deepest leaf.

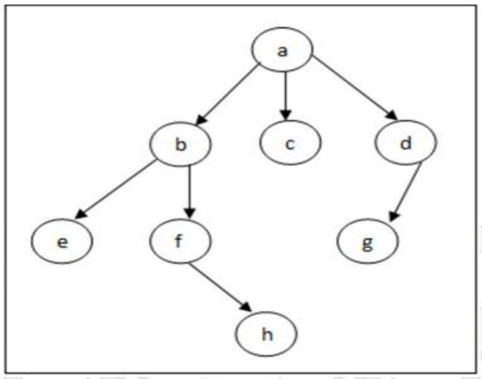


Figure 2: TREE Representation

The rest of the paper is organized as follows. Section 2 discusses some related work and section 3 presents the design of our approach. The details of the results and some discussions we have conducted on this approach are presented in section 4 as Results and Discussions. Sections 5 provide hints of some extension of our approach as future work and conclusion.

II.LITERATURE SURVEY

This section of literature survey eventually reveals some facts based on thought analysis of many authors work as follows.

Yair Horesh, Ramit Mehr and Ron Unger [1] Presents the variant of the A* search algorithm for calculating tree edit distance. They prove that there variant of A* algorithm calculated the distance in much less time for the tree with dozen nodes. They evaluate their variant of the A* algorithm with a brute-force exponential search. They compare 5000 random trees and prove that their algorithm is two orders of magnitude faster than brute-force exponential search algorithm. The efficiency gain of variant of A* algorithm is increased with tree size. They also prove that implementation of variant of A*algorithm is also benefited in hierarchical clustering.

Mateusz Pawlik and Nikolaus Augsten[2] present RTED (robust tree edit distance algorithm). They present the class of LRH (Left-Rights-Heavy), which included RTED and quickest tree edit distance algorithms. They prove that performance of RTED algorithm in runtime complexity is much better than all previous LRH algorithms and it is in any tree shape. They prove that the RTED is outperforming the other approaches, especially when the tree shape within dataset varies.

Nikolaus Augsten, Denilson Barbosa , Michael Bohlen and Themis Palpanas [3] proposed algorithm to solve TASM in an individual pass over the document. Their algorithm is based on a prefix ring buffer which allow them to eliminate all subtrees over the range in an individual postorder search of the data and the value of the prefix ring buffer is linear in the range. Thus the space complexity of the TASM postorder relies only on query size and parameter k. Their solution to TASM is portable and depend upon postorder queue data structure which can be implemented by any storage system or XML processing that allows efficient postorder traversal of trees. Their solution allows the possibility of existed and well acknowledged tree edit distance in practical XML systems.

Rui Yang, Panos Kalnis and Anthony K. H. Tung[4] proposed an efficient technique which depends upon binary tree representation, where they transform trees into binary branch numerical vectors which encodes the original information. They proved that the corresponding vectors L1 distance, whose computational complexity is O (|T1|+|T2|), form a lower bound for edit distance between trees. They described a novel algorithm which exploits the positional branch properties to find the best novel lower of edit distance. They proposed to embed the new distance into a filter and refine the architecture to less the calculation of real edit distance between the data and queries and reduce distance calculation cost.

Erik D. Demaine, Shay Mozes, Benjamin Rossman and Oren Weimann [5] presents a new O(n3)-time and O(n2)-space algorithm, for calculating the tree edit distance between two rooted ordered tree. Their algorithm is symmetric in its two inputs and adaptively dependent upon them. The result of the algorithm needs a novel adaptive technique for determining how a changing program separates into sub problems, and thorough knowledge of the previous algorithms for the problem. They proved that their algorithm is optimized and much faster than all previous algorithms in the worst scenario. The implementation and description of the algorithm is very simple to understand.

In [6] author presents a novel algorithm TREEMINER, which find all frequent subtrees in a forest, by using latest data structure called scope list. They compare the performance of their algorithm with a PATTERN MATCHER algorithm and proved that their algorithm outperform than pattern matching by a factor of 4 to 20 and has good scale up qualities. They also propose an application of tree mining algorithm in web usage mining.

Shotarou Tani, Tetsuhiro Miyahara, Yusuke Suzuki and Tomoyuki Uchida [7] proposed a learning method for obtaining multiple tree structured design by calculation using sets of tag tree patterns as individual, from positive and negative tree structured data. Their introductory experimental results proved that their technique of obtaining characteristic sets of tag tree patterns of higher fitness than individual tag tree patterns.

Shohei Nakai , Tetsuhiro Miyahara , Tetsuji Kuboyama , Tomoyuki Uchida and Yusuke Suzuki [8] proposed a technique by using genetic programming and tree edit distance for obtaining characteristic tree patterns with VLDC's from positive and negative tree structured data. They proved that result of their method is much better at finding characteristic tree structured pattern with respect to the glycan data.

Xiaoying Wu , Stefanos Souldatos , Dimitri Theodoratos, Theodore Dalamagas, Yannis Vassiliou and Timos Sellis [9] presents PTPQs (partial tree pattern queries), a queries class which characterize and contain TPQs. They introduce a sound set and inference rules to classify structural relationship derivation. They also demonstrate that PTPQs can be showed as directed acyclic graphs augmented with the same path constrains. They also designed two algorithm IndexTPQGen and PartialPathJoin to calculate a PTPQ by classifying it into simple set of queries. They also developed PartialTreeStack for PTPQs.

J. Travnicek, J. Janousek, and B. Melichar[10] presents an algorithm that create pushdown automation for nonlinear tree indexing. They showed that space and time complexity of nondeterministic nonlinear tree pattern pushdown automaton is depend on the size of the subject tree.

Ding Chen and Chee-Yong Chan [11] presents a new novel storage design for objectify TPQ views and proposed a new TPQ evaluation algorithm using materialized views for handling normal TPQ queries. They proved that their technique performed better than state of the art approaches by a factor up to 5.8.

Tomoya Mori, Atsuhiro Takasu, Jesper Jansson, Jaewook Hwang, Takeyuki Tamura, and Tatsuya Akutsu [12] present an extended version of the concept of unordered tree inclusion by taking the cost of insertions and substitution operation on the pattern tree into account. They also present an MinCostIncl algorithm which has the same time complexity as the original algorithm for unordered tree decision, but have less space complexity. They proved that the proposed algorithm is fast and scalable by experimenting with real and synthetic datasets. The algorithm is also beneficial for bibliographic matching.

III.PROPOSED METHODOLOGY

This section narrates about the techniques that we are included in searching of the query using sub graph pattern matching that includes tree technique. And the whole process is depicted in figure 3

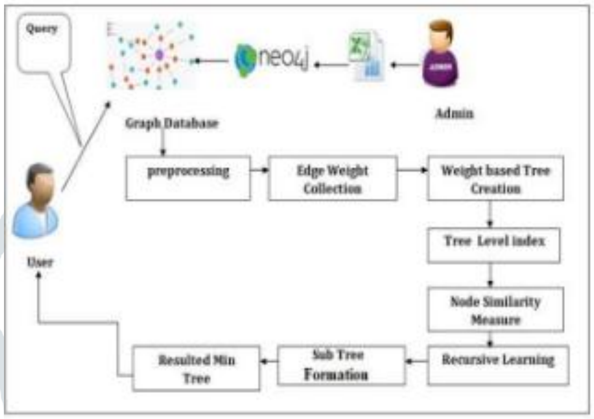


Figure 3: System Overview of sub graph searching using tree

The steps that are depicted in the system over view of figure 3 can be elaborated using the following steps.

Step 1: Graph Creation- Here this is the initial step of our system where system is feeding with the bibliographic dataset of Cora. This Cora Dataset is having many fields like author, article, publication, affiliation and many more. As the system takes the Cora dataset as input it reads all the dataset files in the form of string. And then from this string it searches for the required fields as mentioned above to store them in a list. Once the fields are stored in the list then these lists are used to create the hyper graph to store in the advance graph database like neo4j.

Initially the duplicate elements from this lists are been removed and then the relation from one list elements is measured with the other to for the edges and nodes of the graph G(E,N) to form a set.

Where E represents the edges of the nodes N. Once this set of edges and nodes of the graph are been created then they are subjected to form the graph objects to form intra and inter relationship based on the edge weight. This process eventually generates a hyper graph and the whole process is shown in algorithm 1.

ALGORITHM 1: HYPER GRAPH CREATON

```
//Input : Data collection Set S ={ Ai,ARi,Pb}

//Output: Hyper Graph G(Ai,ARi,Pb)
Where

Ai – Author(node)
ARi – Article(node)
Pb- Publication( Edge)

Step 0:Start

Step 1: Get the Set S
Step 2: FOR i=0 to Size of S

Step 3: Separate Ai, ARi into List Ls,Lh
Step 4: END FOR

Step 5:Get unique elements form Ls andLh
Step 6: Ns=Size of Ls( Number of nodes for Author)
Step 7: Nh=Size of Lh( Number of nodes for Article)
Step 8: FOR i=0 to Size of Ns

Step 9: FOR j=0 to Size of Nh
Step 10: Identify the relational Edges E using pb
Step 11: Form Graph G
Step 12: END FOR

Step 13: END FOR
Step 14: return G
Step 15: Stop
```

Step 2: Query Preprocessing - Here in this process user enters the query to search from the graph database. As the system receives the query from the user it preprocesses the query to form the proper list of the words that need to be search in the dataset. For this preprocessing process system uses stop word removing technique and stemming process.
Stopword Removal- This is the process of removing conjunction words in the given query and replacing them with the empty character. Conjunction words are like is, of, the, and, with etc..
Stemming – This is the process of replacing words like ing, ed from the string with the respective words. This process trims the word and brings backs to them to their basic form. For example looking to look, Studied to study.
Step 3: Tree Formation – Once the query is preprocessed then then the graph from the neo4j database is been read in the form of object. This contains the nodes and edge relationships. Now based on the edge weight analysis each nodes are given weights and stored in double dimension list.
Now nodes are used to create the tree based on the their inherited weights and nodes which contains higher weights are shifted to the right child node. And if they are having lesser weights than of the root nodes then they are assigned to the left child node. And here node similarity is measure to form the clusters of the node while forming of the tree itself using recursive learning process.
The whole process can be depicted with the following algorithm 2.

ALGORITHM 2: TREE FORMATION

```
//input : Graph Set List GL
Step 0: Start
Step 1: Create an empty tree as T

Step 2: Create the Root Node for first entity of the graph as Rn

Step 3 :FOR i=0 to size of GL
Step 4: Compare the distance with the root node Rn
Step 5:If (GLi  Weight < Rn)
Step 6:Add node as left child in T
Step 7: Else
Step 8:Add node as Right child in T
Step 9:End FOR
Step 10: return T
Step 11: Stop
```

Step 4: Subtree cluster **–** Here in this step a sub tree is created based on the query fired by the user from the tree. Which is eventually nothing but a cluster formed due to similar node weights and this represents the answer for the query fired by the user.

IV.RESULTS AND DISCUSSIONS

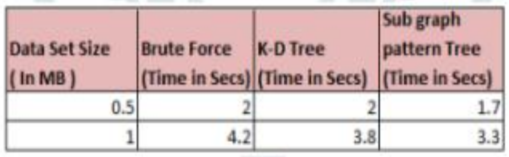
Experimental Setup

The proposed system's implementation is carried out by using java based windows machine having configuration of intel corei3 processor with 4 GB primary memory. Proposed system uses Neo4j advance graph database to store the the hyper graph created on pre seracing process. System uses netbeans as standard development IDE along with the Bibilographic cora Dataset .

Evaluation

To evaluate the process of our sub graph pattern search using tree inclusion some experiments are conducted based on the fact of performance time for the dataset of different size. When the performance is measured for brute force search technique and KD- Tree search technique as mentioned in [13]. System accumulates some facts as tabulated in table 1.

Table 1: Comparision Result with Brute Force and KD tree



Data Set Size (In MB)	Brute Force (Time in Secs)	K-D Tree (Time in Secs)	Sub graph pattern Tree (Time in Secs)
0.5	2	2	1.7
1	4.2	3.8	3.3

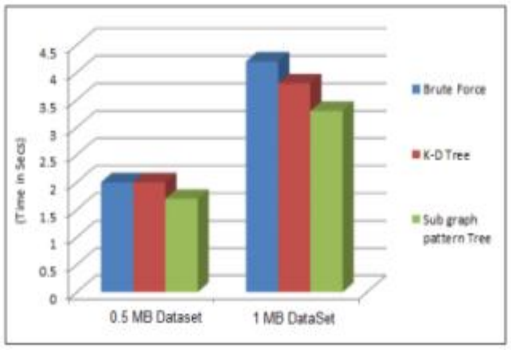


Figure 4: Comparision graph with Brute Force and KD Tree search technique

On observing the above graph in figure 4 our methodology of subgraph pattern using tree technique overperforms than that of the Brute Force and KD tree technique. As the proposed system uses both graph and tree data structure which eventually enhance the process of searching , as a result of this system take considerable less time than that of the methodologies mentioned in [13].

V.CONCLUSION AND FUTURE SCOPE

The proposed methodology of sub graph pattern search using tree technique uses the vast bibliographic data set of Cora. System concentrate on some of the entities for searching the query like author, publication etc. from the considered dataset. For effective searching results system uses the most advance graph database like neo 4j in which our technique stores the hyper graph that eventually stores the enriched node information based on the edge weights.

The edge weights of the graph forces the tree to create in sorted manner in which nodes of the same weights accumulates in the same clusters. These clusters eventually enhance the process of searching to yield best results. This system can be enhancing in the future by considering the vast graph data structure that can be store in independent XML format of files and by inclusion ontology tree process searching technique can make to yield most semantic results

VI. ACKNOWLEDGMENT

We are thankful to the authorities of Savitribai Phule University of Pune and concern members of CPGCON2016 conference, organized by, for their constant guidelines and support. We are also thankful to the reviewer for their valuable suggestions. We also thank the college authorities for providing the required infrastructure and support. Finally, we would like to extend a heartfelt gratitude to friends and family members.

REFRENCES

[1] Yair Horesh, Ramit Mehr and Ron Unger, “Designing an A* Algorithm for Calculating Edit Distance between Rooted Unordered Trees” Journal of Computational Biology, Pp. 1165-1176, Volume 13, Number 6, 2006.
[2] Mateusz Pawlik and Nikolaus Augsten, “RTED: A Robust Algorithm For The Tree Edit Distance” VLDB Endownment, Vol. 5, No. 4, 2150-8097/11/12, 2011.
[3] Nikolaus Augsten, Denilson Barbosa , Michael Bohlen and Themis Palpanas, “TASM: Top-k Approximate Subtree Matching” IEEE, 978-1-4244-5446-4/10, 2010.
[4] Rui Yang, Panos Kalnis and Anthony K. H. Tung, “Similarity Evaluation on Tree Structured Data” SIGMOD 2005, ACM, 1-59593-060-4/05/06, 2005.
[5] Erik D. Demaine, Shay Mozes, Benjamin Rossman and Oren Weimann, “An Optimal Decomposition Algorithm for Tree Edit Distance” MIT, Cambridge, MA 02139, USA 2009.
[6] Mohammed J. Zaki, “Efficient Mining Frequent Trees in a Forest” ACM 1-58113-567-X/02/007, 2002.
[7] Shotarou Tani, Tetsuhiro Miyahara, Yusuke Suzuki and Tomoyuki Uchida, “Acquisition of Multiple Tree Structured Patterns by an Evolutionary Method using Sets of Tag Tree Patterns as Individual”, IEEE, DOI 10.1109/IIAI-AAI.2015.271, 978-1-4799-9958-3/15, 2015.
[8] Shohei Nakai , Tetsuhiro Miyahara , Tetsuji Kuboyama , Tomoyuki Uchida and Yusuke Suzuki,“Acquisition of Characteristic Tree Patterns with VLDC’s by Genetic Programming and Edit Distance”,IEEE, DOI 10.1109/IIAI-AAI.2013.79, 978-0-7695-5071-8/13, 2013.