Volume: 2 Issue: V Month of publication: May 2014
DOI:

www.ijraset.com Vol. 2 Issue V, May 2014

ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Clustering of Answers of Keyword Search on Graph

Jyoti Devi 1, Vinod Saroha 2

1 M.Tech (Network Security),
School of Engineering & Sciences
Sonepat, Haryana, India.
BPS Mahila Vishwavidyalaya, Khanpur Kalan

2 Lecturer in CSE Dept
School of Engineering & Sciences
Sonepat, Haryana, India.
BPS Mahila Vishwavidyalaya, Khanpur Kalan

Abstract: Keyword search on graph data returns answers that are represented as a set of tree structures. These are referred as answer-trees. It was observed that when following exhaustive breadth-first search strategy many answers of a user search are found to have the common set of keyword-nodes and the root-node, but the paths to root-node from keyword nodes are different. Often thousands of answers get generated with the same set of keyword-nodes and the root-node. However there exist various kinds of similarities in the result trees. The paper present various techniques to cluster the trees which unable efficient analysis of the results and one can drive interesting relationships among the answers.

Keywords :- keyword search, hashing, tree clustering, tree edit distance.

I. INTRODUCTION

The distributed keyword search algorithm generates paths from various keyword-nodes to corresponding root-nodes. The current brute-force approach first groups these paths based on the keyword of the target keyword-node, for a root-node, i.e., all the paths from a root-node to all the keyword-nodes of a keyword are grouped together. It then takes a cross-product of these groups of paths, taking one path from each of the keyword. Each of the result from such a cross-product is assumed to be a valid answer. Objective of this project is to summarize the answers-trees based on various configurations, e.g., common root-node grouping, common root-node and keyword-node grouping, grouping based on root-node class and common semantic form of answers etc. This paper will require choice /design of suitable data-structure as well as design of an efficient algorithm for answer summarization. Answer summarization fulfills two main objectives, which are

1. The search algorithm for finding the answers to the keyword queries takes around 10-15 minutes (for now), however parsing and enumerating the searched results takes hours. In other words, the bottleneck is the Enumeration of results rather than searching the results. It happens because there are thousands of answers to be parsed and enumerated. However, if we can club these thousands of answers into different buckets so that similar

answers are placed in same bucket and enumerate one representative answer for each bucket, we would need to enumerate much less answers. All different answers of same bucket can be enumerated after clicking the representative answer of that buckets. This way we can improve the bottleneck part of the whole procedure of finding and displaying the answers.

2. User, obviously would not like to see hundreds of answers of same kind when he is interested in some other kind of answers to his keyword search query. Through summarized answers he can easily have a glance to all different kind of answers and getting detailed answer by clicking on the answer of his interest. This way we escaped displaying all other answers which is of not his interest. However, summarizing answer based on common semantic form will give most informative summarized answers than all other kind of summarization based on common root-node, common root node and keyword nodes. It is so, because the two answer may be completely different but still they can have common root and common keyword nodes or just common root node. For example if we search "Rekha Bachhan", then two different answers can be as follows These two answers have common root node and common-keyword nodes and still they are totally different from each other. However if answers are summarized based on semantic form then the answers which needed to be clubbed together based on common root-node and common keyword-nodes will automatically be clubbed because they will have same form. Also, summarizing based on only semantic form is also not the best summarization technique. There might me some answers whose semantic form might differ only slightly from other answers' form and they should be clubbed together. Also, there can be some other basis which might turns out to be better than semantic form technique. However, for now, we will go for the semantic based summarization.

II. METHODS

The total number of answer results is very large, say thousands. Firstly, all these answers are converted to their semantic form and let's call these new answers Semantic Results Tree. Then we can have following methods to classify them in already discussed buckets.

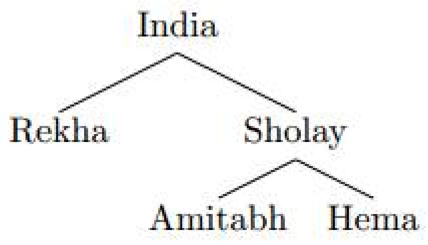
Brute Force Method(Isomorphism)

This is given in Algorithm 1 [3.2.2]. The main drawback of this algorithm is that it is very expensive in term of time. In worst case, it does O N comparisons and each comparison is of order O V, where N is the total number of results and V is the maximum number of nodes in a result tree. Hence total time taken by this algorithm is O N2—————————————
.

Prime Number Based Summarization

Feature Vectors

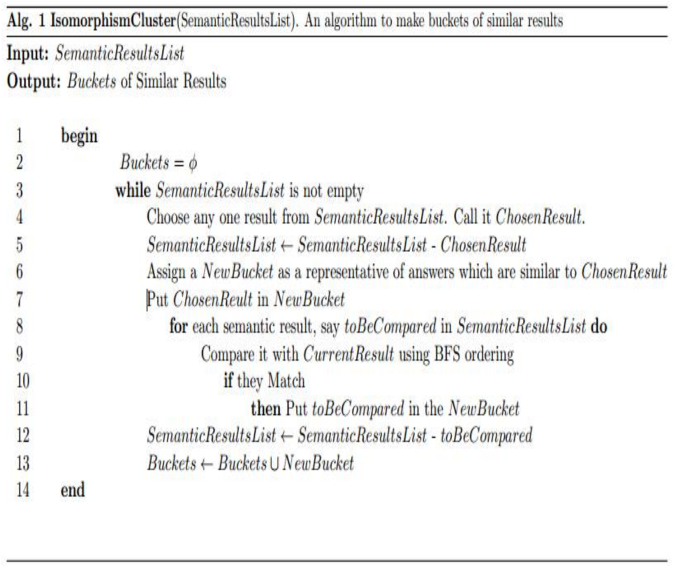
In a result tree, each node has some degree. A vector is generated for each tree which has the degrees of nodes in increasing order. We call this vector to be feature vector of the tree. For example, the Feature Vector for following tree would be [(Rekha,1), (Amitabh,1), (Hema,1), (Sholay,2),(India,2)]



Instead we can also use pre-order, post-order or in-order ordering of trees to generate Feature Vector.

Prime Keys

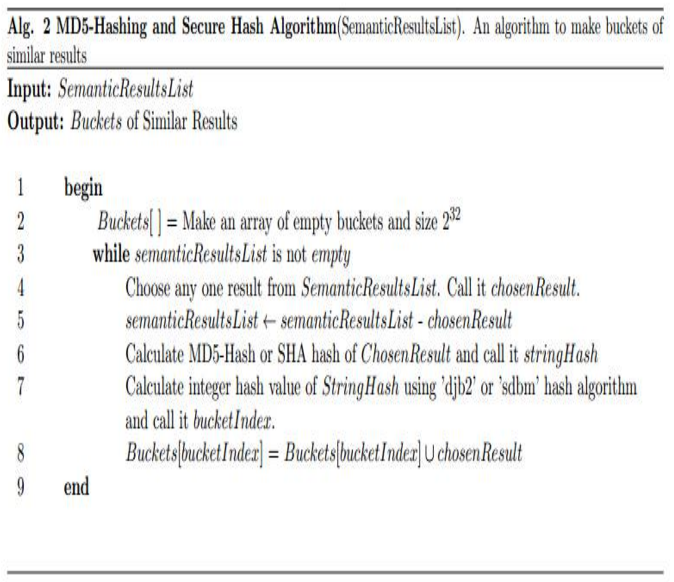
We start from a random result tree and assign its root the first prime number i.e. 2 . Then we do BFS traversal of the tree and assign increasing distinct prime number. When further other trees are traversed we check if the current node-class has already been assigned a prime number, if yes then assign it the same prime number else assign it next distinct prime number. For example, following are two result trees



Alg. 1 IsomorphismCluster(SemanticResultsList). An algorithm to make buckets of similar results

Input: *SemanticResultsList*
Output: *Buckets* of Similar Results

```
begin
        Buckets = $\phi$
      while SemanticResultsList is not empty
           Choose any one result from SemanticResultsList. Call it ChosenResult.
           SemanticResultsList ← SemanticResultsList - ChosenResult
           Assign a NewBucket as a representative of answers which are similar to ChosenResult
           Put ChosenReult in NewBucket
              for each semantic result, say toBeCompared in SemanticResultsList do
                  Compare it with CurrentResult using BFS ordering
                    if they Match
                        then Put toBeCompared in the NewBucket
           SemanticResultsList ← SemanticResultsList - toBeCompared
           Buckets ← Buckets ∪ NewBucket
end
```

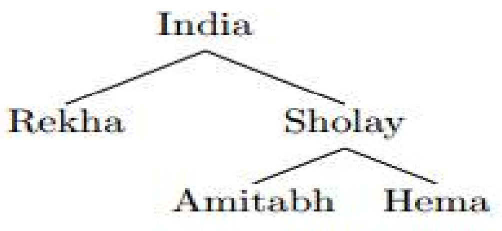


Alg. 2 MD5-Hashing and Secure Hash Algorithm(SemanticResultsList). An algorithm to make buckets of similar results

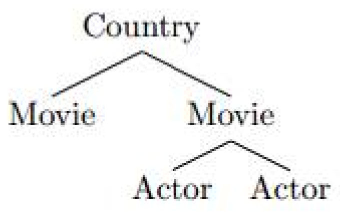
Input: *SemanticResultsList*
Output: *Buckets* of Similar Results

```
begin
    Buckets[ ] = Make an array of empty buckets and size $2^{32}$
    while semanticResultsList is not empty
        Choose any one result from SemanticResultsList. Call it chosenResult.
        semanticResultsList ← semanticResultsList - chosenResult
        Calculate MD5-Hash or SHA hash of ChosenResult and call it stringHash
        Calculate integer hash value of StringHash using 'djb2' or 'sdbm' hash algorithm
        and call it bucketIndex.
        Buckets[bucketIndex] = Buckets[bucketIndex] ∪ chosenResult
end
```

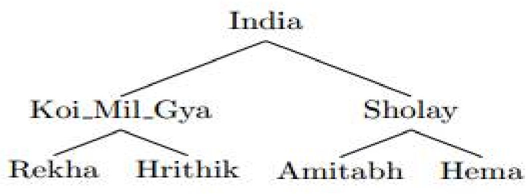
First result is



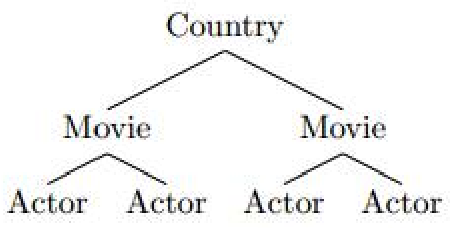
Whose semantic form is given by



Second result is



whose semantic form is given by



Now, when first semantic result tree is traversed, the prime number assigned are as follows

Vol. 2 Issue V, May 2014

ISSN: 2321-9653

Node-Class	Prime Number
Country	2
Movie	3
Actor	5

It is to be noted that when some node class had already been assigned some prime number, it is not assigned again when met in future rather is assigned the already assigned value to that node-call. When second tree is traversed, no node is given new prime number since no new node class is met. Hence each node in all the result trees is assigned some prime number.

Hashing

We can generate a hash function using the Feature vector and Prime Keys as follows.

$$h(FV) = \Pi_{For\_all\_nodes\_in\_FV} FV.node.PrimeKey^{(FV.node.degree)} \pmod{n}$$

Here FV is a Feature Vector and n is some very larger Prime Number.

For example hash value of first result tree would be $5^1.5^1.5^1.3^2.3^2 = 4500$

Hence, this way we can hash each tree to some bucket and finally we will get similar tree in one bucket. It process every tree exactly two times. Hence is linear in number of result trees. However, there is a inherent limitation of this method. If number of total different node-class in our result trees becomes too large say of the order of billion then, we will have to use very large prime numbers. It would force us to use big integer as there will be very large hash value and it might make the procedure slow.

3.3 MD5-Hashing and Secure Hash Algorithm

Algorithm 2 [3.2.2] gives the outline of this algorithm Every result tree is first converted to it's semantic tree form and then the in-order traversal of the tree is written in a document. For each result tree, there is a document. Here document is nothing but a string representing in-order traversal of the tree. This string is then hashed using MD5 Hash function or Secure Hash Algorithm which will give a hash string as an output. This string is further hashed using 'djb2' hashing or 'sdbm' hashing which gives an integer ranging from 1 to 2^{32} .

III. CLUSTERING BASED ON TREE EDIT DISTANCE

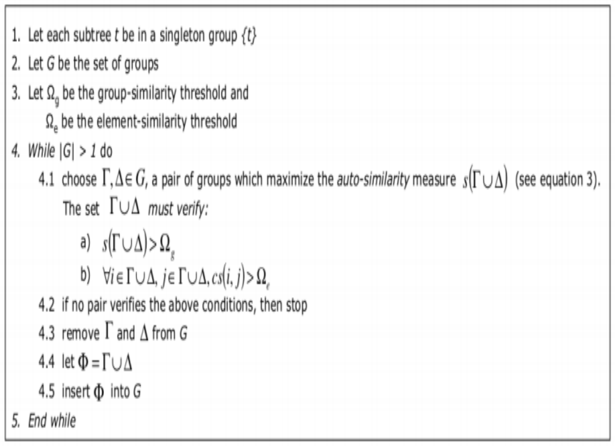
To compute similarity measures we use techniques based in string edit-distance algorithms. Tree edit distance (TED) between two trees Ti and Tj is defined as the number of operations(insertion, deletion and re-labels) required to transform a Ti to Tj. Let ed(Ti and Tj) define the edit distance. We compute is using a well known algorithm called Klein's Algorithm. Then we compute the edit-distance similarity between two trees.

es(Ti and Tj) = 1-(ed(Ti and Tj)/Number of Nodes in both trees)

Clustering the Trees

For grouping the sub trees according to their similarity, we use a clustering-based process we describe in the following lines:

Figure 1: Bottom Up Clustering based on Tree Edit Distance



```
1. Let each subtree t be in a singleton group {t}
2. Let G be the set of groups
3. Let $\Omega_g$ be the group-similarity threshold and
     $\Omega_e$ be the element-similarity threshold
4. While |G| > 1 do
    4.1 choose $\Gamma, \Delta \in G$, a pair of groups which maximize the auto-similarity measure $s(\Gamma \cup \Delta)$ (see equation 3).
        The set $\Gamma \cup \Delta$ must verify:
          a) $s(\Gamma \cup \Delta) > \Omega_g$
          b) $\forall i \in \Gamma \cup \Delta, j \in \Gamma \cup \Delta, cs(i,j) > \Omega_e$
    4.2 if no pair verifies the above conditions, then stop
    4.3 remove $\Gamma$ and $\Delta$ from G
    4.4 let $\Phi = \Gamma \cup \Delta$
    4.5 insert $\Phi$ into G
5. End while
```

- Let us consider the set t1,.....tn of all the result trees.
- Compute the similarity matrix. This is a n*n matrix where the (i,j) position (denoted mij) is obtained as cs(ti and tj), the edit-distance similarity between ti and tj. We define the column similarity between ti and tj, denoted cs(ti,tj), as the inverse of the average absolute error between the columns corresponding to ti and tj in the similarity matrix (2). Therefore, to consider two sub trees as similar, the column similarity measure requires their columns in the similarity matrix to be very similar. This means two sub trees must have roughly the same edit-distance similarity with respect to the rest of sub trees in the set to be considered as similar. We have found column similarity to be more robust for estimating similarity between ti and tj in the clustering process than directly using ed(Ti and Tj).
- Now, we apply bottom-up clustering [3] to group the subtrees. The basic idea behind this kind of clustering is to start with one cluster for each element and successively combine them into groups within which inter-element similarity is high, collapsing down to as many groups as desired. The

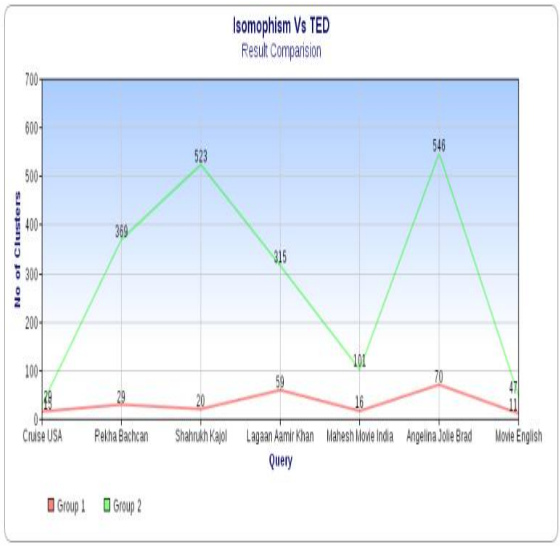
$$cs(t_i,t_j)=1-\sum_{k=1..n}\left|m_{ik}-m_{jk}\right|/n$$

$$s(\Phi)=2/|\Phi|\left(|\Phi|-1\right)\sum_{t_i,t_j\in\Phi}cs(t_i,t_j)$$

algorithm is shown in the Figure 1.

Results

We compared the results of Clustering based on Isomorphism and Clustering based on Tree Edit Distance. The database used was IMDB which has around 10 Millions tuples. For each query, 1000 result trees were generated and then clustered using these two techniques. It can be seen that in some queries no of clustered returned by isomorphism were very high while TED approach give good number of clusters in all the queries. Group 2 shows the result of TED-Clustering and Group-1 shows Isomorphism results.



IV. CONCLUSION

In this paper we provided various techniques to cluster the trees. Clustering based on Tree Edit Distances and clustering based on hashing are two viable techniques to cluster the trees. Isomorphism is both time expensive as well as not that good strategy for clustering as it doesn't give good results.

V. REFERENCES

1. Manuel Manuel lvarez, Alberto Pan, Using Clustering and Edit Distance Techniques for Automatic Web Data Extraction

2. Philip Klein, Srikanta Tirthapura, Daniel Sharvit, and Ben Kimia. A tree- edit-distance algorithm for comparing simple, closed shapes. In Proceed- ings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 696704, 2000.

3. P.N. Klein. Computing the edit-distance between un rooted ordered trees. In Proceedings of the 6th annual European

Symposium on Algorithms (ESA) 1998., pages 91102. Springer-Verlag, 1998.

4. Arvind Gupta and Naomi Nishimura. Finding largest sub trees and smallest super trees. Algorithmica,21:183210, 1998.