Clustering XML Document and Interactive Fuzzy Search

Asvany.T[1], Jamaludeen.A[2], SenthilKumaran.C[3]

[1]Assistant Professor, Department of MCA, CCET, Puducherry.asvanytandabani@gmail.com
[2]Senior Assistant Professor, Department of MCA, CCET, Puducherry.jam.ahamath@gmail.com
[3]Assistant Professor, Department of MCA, CCET, Puducherry.senthilmca81@gmail.com

***Abstract*—In a traditional keyword-search system over XML data, a user composes a keyword query, submits it to the system, and retrieves relevant answers. In the case where the user has limited knowledge about the data, often the user feels "left in the dark" when issuing queries, Cluster the data first and has to use a try-and-see approach for finding information. study interactive fuzzy search in XML documents, a new information-access paradigm in which the system searches XML data on the fly as the user types in query keywords. It allows users to explore data as they type, even in the presence of minor errors of their keywords. Our proposed method has the following features: 1) Search as you type: It extends Auto complete by supporting queries with multiple keywords in XML data. 2) Fuzzy: It can find high-quality answers that have keywords matching query keywords approximately. 3) Efficient: Our effective index structures and searching algorithms can achieve a very high interactive speed. I examine effective ranking functions and early termination techniques to progressively identify the top relevant answers. I have implemented our method on real data sets, and the experimental results show that our method achieves high search efficiency and result quality.**

I. INTRODUCTION

The XML documents as rooted ordered labelled trees, we study the usage of structural distance metrics in hierarchical clustering algorithms to detect groups of structurally similar XML Documents. We suggest the usage of tree structural summaries to improve the performance of the distance calculation and at the same time to maintain or even improve its quality.

1.1 Clustering Approaches

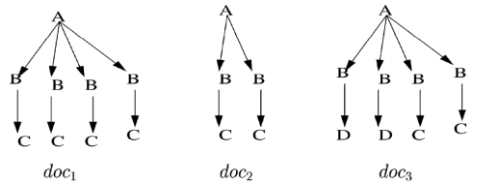
Different algorithms have been proposed for clustering XML documents that are extensions of the classical hierarchical and partitioning clustering approaches. We remind that agglomerative algorithms find the clusters by initially assigning each document to its own cluster and then repeatedly merging pairs of clusters until a certain stopping criterion is met. The end result can be graphically represented as a tree called a *dendrogram*. The dendrogram shows the clusters that have been merged together, and the distance between these merged clusters (the horizontal length of the branches is proportional to the distance between the merged clusters). By contrast, partitioning algorithms find clusters by partitioning the set of documents into either a predetermined or an automatically derived number of clusters. The collection is initially partitioned into clusters whose quality is repeatedly optimized, until a stable solution based on a criterion function is found.

Hierarchical clustering in general produces clusters of better quality but its main drawback is the quadratic time complexity. For large documents, the linear time complexity of partitioning techniques has made them more popular especially in IR systems where the clustering is employed for efficiency reasons.

Quality Measure	Formula	
Recall and precision	$R(i,j)=\frac{n_{ij}}{n_i}$ $P(i,j)=\frac{n_{ij}}{n_j}$	i– a class of the q classes j – a cluster of the k clusters n – number of items n_i– items of class i n_j– items in cluster j n_{ij}– items of class I in cluster j
Entropy	$E(j)=-\frac{1}{logq}\sum_{i-1}^{q}P(i,j)\log P(i,j)$	$Entropy=\sum_{j-1}^{k}\frac{nj}{n}E(j)$
Purity	$Q(j)=max_{i-1}^{q}P(i,j)$	$Purity=\sum_{j-1}^{k}\frac{nj}{n}Q(j)$
F-measure	$F(i,j)=\frac{2R(i,j)P(i,j)}{R(i,j)+P(i,j)}$	$F=\sum_{j-1}^{k}\frac{nj}{n}F(i,j)$

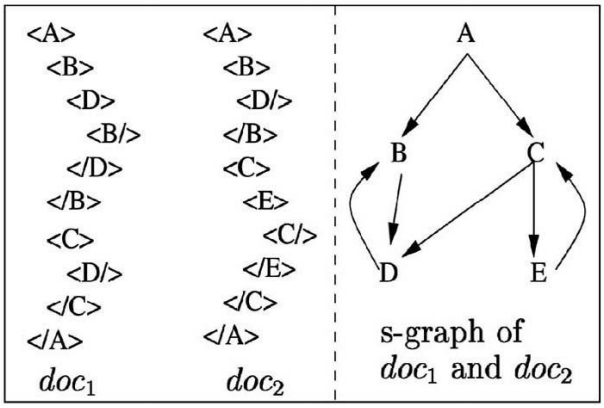
A Framework for Clustering XML Documents Our purpose is to cluster XML files based on their structure we achieve this by summarizing their structure in s- graphs and using the metric. Our approach is implemented in two steps:
Step 1. Extract and encode structural information: This step scans the documents, computes their s-graphs, and encodes them in a data structure.
Step 2. Perform clustering on the structural information: This

step applies a suitable clustering algorithm on the encoded information to generate the clusters.

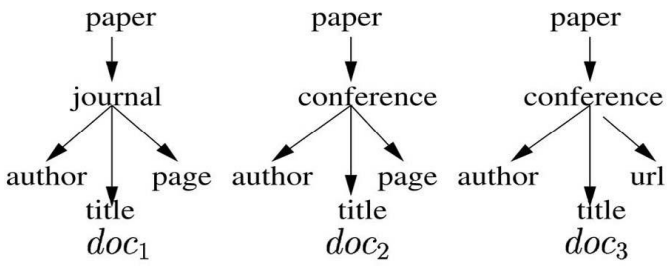


The above figure shows the Tree Distance between Documnets.

This simple example shows that the tree distance based method may not be able to distinguish structural differences in some cases. In the following, we propose a new notion to measure the similarity between XML documents. Given a set of xml documents C, the structure graph (or s-graph) of C, sg(c)=(N,E), is a directed graph such that N is the set of all the elements and attributes in the documents in C and (a,b)€E if and only if a is a parent element of element b or b is an attribute of element a in some document in C.The structure graph defined here is different from the DTD graph. The below figure shows a example of s-graph.



The structure graphs are derived from XML documents, not from their DTD. For example, the s-graph sg(doc1,doc2) of two documents doc1 and doc2 is the set of nodes and edges appearing in either document. The same manner, a path expression q can be viewed as a graph(N,E),where N is the set of elements or attributes in q and E is the set of element – sub element or element – attribute relationships in q. Given a path expression q which has an answer in an XML document X, the directed graph representing q is a sub graph in the s-graph of X. For simplicity we will denote the graph of a path expression q also by q.



An example of s-graph based similarity

Search Introduction

Keyword search is a proven and widely accepted mechanism for querying in textual document systems and the World Wide Web. A keyword search looks for words anywhere in the record. It is emerged as most effective paradigm for discovering information on web. The advantage of keyword search is its simplicity-users do not have to learn complex query language and can issue query without any knowledge about structure of xml document. The most important requirement for the keyword search is to rank the results of query so that the most relevant results appear.

Keyword search provides simple and user friendly query interface to access xml data in web. Traditional methods use query languages such as Xpath and XQuery to query XML data. These methods are powerful but unfriendly to non expert users. First, these query languages are hard to comprehend for non-database users. For example, XQuery is fairly complicated to grasp. Second, these languages require the queries to be posed against the underlying, sometimes complex, database schemas. Fortunately, keyword search is proposed as an alternative means for querying XML data, which is simple and yet familiar to most Internet users as it only requires the input of keywords. One limitation of Auto complete is that the system treats a query with multiple keywords as a single string; thus, it does not allow these keywords to appear at different places.

Keyword search over xml is not always the entire document but deeply nested xml. Xml was designed to transport and store data. Xml document contains text with some tags which is organized in hierarchy with open and close tag. Xml model addresses the limitation of html search engine i.e. Google which returns full text document but the xml captures additional semantics such as in a full text titles, references and subsections are explicitly captured using xml tags. For querying xml data keyword search is proposed as an alternative method. While query semantics and algorithm efficiency have been widely discussed, top-K keyword search in XML databases is an important issue that very little work has concentrated on. As is typical in the keyword search systems, a ranking function can be defined [5], [13] to assign to results ranking scores, and ranked results are returned to users. Top-K processing aims to compute the results with highest scores first so that execution can terminate earlier after the top K results have been generated.

Existing algorithms focusing on efficiency cannot provide effective support for top-K processing. These

algorithms share some common characteristics: inverted lists are sorted by the document order. At least one list is scanned sequentially. This behaviour determines that results are generated in the document order, rather than the order of ranking scores. All the results must be generated in order to return the top K results. Essentially, these algorithms are designed to optimize the semantic pruning, and are incapable of supporting top-K processing.

In traditional approach to query over xml data it requires query languages such as XPath and XQuery which are very hard to comprehend for non database users. It can only understand by professionals. Recently database community has been studying challenges related to keyword search over xml data. However the traditional approaches are not user friendly. To solve this problem many systems introduced various features.

Let's say you have the following XML document (table.xml)

```
<xml>
  <table>
    <rec id="1">
      <numField>123</numField>
      <stringField>String
Value</stringField>
    </rec>
    <rec id="2">
      <numField>346</numField>
      <stringField>Text
Value</stringField>
    </rec>
    <rec id="3">
      <numField>-23</numField>

<stringField>stringValue</stringField>
    </rec>
  </table>
</xml>
```

The main purpose is to develop Interactive Fuzzy Search in XML Data. Interactive Fuzzy search searches the XML data on the fly as user's type in query keywords, even in the presence of minor errors of their keywords. An interactive search is a user interface interaction method to progressively search for filter through text. As the user types text, one or possible matches for text are found and immediately present to user. The interactive fuzzy search in xml data returns the approximate results.

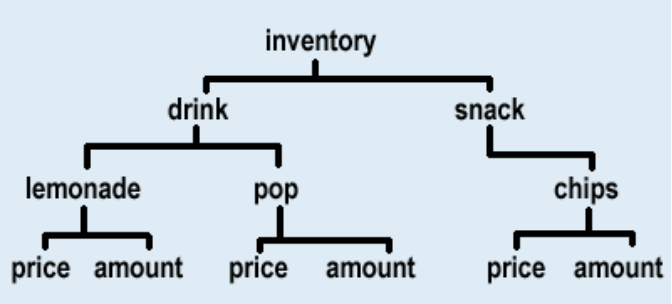


Fig 1.1.1Architecture of an XML Document

One method is Auto complete which predicts the words the user had typed in. One limitation of this approach is it treats multiple key words as single keyword and do not allow them to appear in different places. To address this problem other method is proposed complete search in textual documents which allows multiple keywords to appear in different places but it does not allow minor mistakes in query.

In the above structure, an XML document contains the parent-child relationships. Here bibliography is the root element. Conference and Journal is the parent element for the upcoming child element such as name, year, paper, chair, etc and WWW, 2009, IR DB, Tom Mices are the xml datas for the above mentioned child element.

Overview of Approach

There are two challenges to support fuzzy type-ahead search in XML data. The first one is how to interactively and efficiently identify the predicted words that have prefixes similar to the input partial keyword after each keystroke from the user. The second one is how to progressively and effectively compute the top-k predicted answers of a query with multiple keywords, especially when there are many predicted words.

LCA-Based Interactive FUZZY SEARCH

The **l**owest common ancestor (LCA) is a concept in graph theory and computer science. Let *T* be a rooted tree with *n* nodes. The lowest common ancestor between two nodes *v* and *w* is defined as the lowest node in *T* that has both *v* and *w* as descendants.

The LCA of *v* and *w* in *T* is the shared ancestor of *v* and *w* that is located farthest from the root. There are different ways to answer the query on an xml document; one commonly used method is LCA based method. Many algorithms that use query over xml uses this method. Content nodes are the parent node of the keyword. For example consider keyword db in fig 1.1.1 then content node of db is node 13 and node 16. The server contains index structure of xml document which each node is letter in keyword and leaf node contain all nodes that contain the keyword this leaf node is called inverted list.

Algorithm


```
1.  function LCA(u)
2.  MakeSet(u);
3.  u.ancestor := u;
4.  for each v in u.children do
5.  LCA(v);
6.  Union(u,v);
7.  Find(u).ancestor := u;
8.  u.colour := black;
9.  for each v such that {u,v} in P do
10. if v.colour == black
11. print "Lowest Common Ancestor of " + u +" and " +
    v + " is " + Find(v).ancestor + ".";
12. function MakeSet(x)
13. x.parent := x
14. x.rank   := 0
15. function Union(x, y)
16. xRoot := Find(x)
17. yRoot := Find(y)
18. if xRoot.rank > yRoot.rank
19. yRoot.parent := xRoot
20. else if xRoot.rank < yRoot.rank
21. xRoot.parent := yRoot
22. else if xRoot != yRoot
23. yRoot.parent := xRoot
24. xRoot.rank := xRoot.rank + 1
25. function Find(x)
26. if x.parent == x
27. return x
28. else
29. x.parent := Find(x.parent)
30. return x.parent
```

For keyword query the LCA based method retrieves content nodes in xml that are in inverted lists. Identify the LCAs of content nodes in inverted list. Takes the sub tree rooted at LCAs as answer to the query for example suppose the user typed the query "www db" then the content nodes of db are {13,16} and for www are 3 ,the LCAs of these content nodes are nodes ,12,15,2,1.here the nodes 3,13,12,15 are more relevant answers but nodes 2 and 1 are not relevant answers.

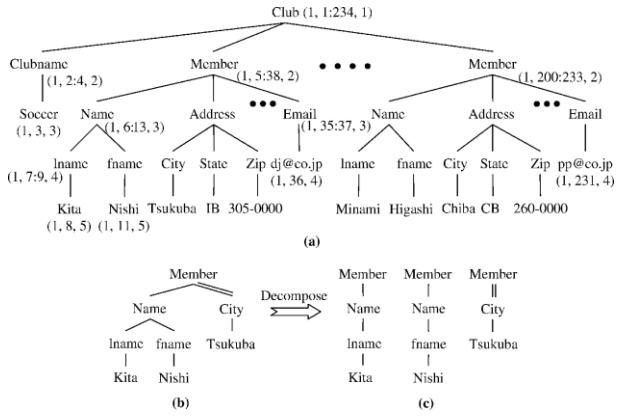


Fig 1.1.2: The extended tire

In the above structure, for each leaf node in the trie, we index not only the content nodes for the keyword of the leaf node, but also those quasi-content nodes whose descendants contain the keyword. For instance, consider the XML document in Fig. 1.1.1. For the keyword "DB," we index nodes 13, 16, 12, 15, 9, 2, 8, 1, and 5 for this keyword as shown in Fig. 1.1.2. For the keyword "IR," we index nodes 6, 16, 24, 5, 15, 23, 2, 20, and 1. For the keyword "Tom," we index nodes 14, 17, 12, 15, 9, 2, 8, 1, and 5. The nodes are sorted by their relevance to the keyword.

MINIMAL- COST TREE

To find relevant answers to a keyword query over an XML document. In the framework, each node on the XML tree is potentially relevant to the query with different scores. For each node, we define its corresponding answer to the query as its subtree with paths to nodes that include the query keywords. This subtree is called the "minimal-cost tree" for this node. Different nodes correspond to different answers to the query, and we will study how to quantify the relevance of each answer to the query for ranking.

Algorithm

1. scan index lists in parallel;
2. consider dj at position posi in Li;
3. E(dj) := E(dj) Є {i};
4. highi := si(q,dj);
5. bestscore(dj) := aggr{x1, ..., xm)
    a. with xi := si(q,dj) for i Є E(dj),
    b. highi for i Є E(dj);
6. worstscore(dj) := aggr{x1, ..., xm)
    a. with xi := si(q,dj) for i Є E(dj), 0 for i Є E(dj);
7. top-k := k docs with largest worstscore;
8. threshold := bestscore{d | d not in top-k};
9. if min worstscore top-k ≥ threshold then exit;

Sort the scores in the inverted lists. If the inverted list is long the partial virtual inverted list. Construct max tree, such that each node contain <node, score>. The top element of max tree is highest score node and is deleted, max tree is adjusted. Deleted node with score<=T (threshold) are taken into result set and return the result set if the top – k answers are retrieved. Consider the XML document and given a keyword query Q = {DB; Tom; WWW}. Nodes 3, 13, 14, 16, and 17 are content nodes of the three keywords; nodes 1, 2, 5, 8, 9, 12, and 15 are their quasi-content nodes. Node 3 is the pivotal node for node 2 and "WWW". Node 16 is the pivotal node for node 2 and "DB". Node 17 is the pivotal node for node 2 and "Tom". The MCT of node 2 is the subtree rooted at node 2, which contains the paths: n2 → n3, n2 → n15 → n16, and n2 → n15 → n17.

CONCLUSION

The keyword search over the xml data which is user-friendly and there is no need for the user to study about the xml data .This paradigm gives the relevant results the user wants. Fuzzy search over xml data is studied which gives approximate results. In my project, various methods for querying on xml data are used. I proposed effective index structures, efficient algorithms, and novel optimization techniques to progressively and efficiently identify the top answers. I examined the LCA-based method to interactively identify the predicted answers. I have developed a minimal-cost-tree-based search method to efficiently and progressively identify the most relevant answers. I have implemented our method, and the experimental results show that our method achieves high search efficiency and result quality.

References

[1] Jianhua Feng, and Guoliang Li," Efficient Fuzzy Type-Ahead Search in XML Data", VOL. 24, NO. 5, MAY 2012.

[2] K. L. A. Nivedita, R. Naveen, K. Sravani, "Fuzzy Type-Ahead Search in XML Data", Vol. 3, Issue 4, Jul-Aug 2013, pp.1579-1583

[3] G. Li, J. Feng, and L. Zhou, "Interactive Search in Xml Data," Proc. Int'l Conf. World Wide Web (WWW), pp. 1063-1064, 2009.

[4] S. Ji, G. Li, C. Li, and J. Feng, "Efficient Interactive Fuzzy Keyword Search," Proc. Int'l Conf. World Wide Web (WWW), pp. 371-380, 2009.

[5] G. Li, S. Ji, C. Li, and J. Feng, "Efficient Type-Ahead Search on Relational Data: A Tastier Approach," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 695-706, 2009.

[6] Z. Bao, T.W. Ling, B. Chen, and J. Lu, "Effective XML Keyword Search with Relevance Oriented Ranking," Proc. Int'l Conf. Data Eng. (ICDE), 2009.

[7] G. Li, B.C. Ooi, J. Feng, J. Wang, and L. Zhou, "Ease: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-Structured and Structured Data," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 903-914, 2008.

