Cooperative XML (CoXML) Query Answering at INEX 03

Shaorong Liu and Wesley W. Chu
UCLA Computer Science Department, Los Angeles, CA 90095
{sliu, wwc}@cs.ucla.edu

ABSTRACT

The Extensible Markup Language (XML) is becoming the most popular format for information representation and data exchange. Much research has been investigated on providing flexible query facilities while aiming at efficient techniques to extract data from XML documents. However, most of them are focused on only the exact matching of query conditions. In this paper, we describe a cooperative XML query answering system, CoXML, which cooperates with the users by extending query relaxation techniques and provides approximate matching of query conditions. We also present our participation effort in the Initiative for the Evaluation of XML Retrieval (INEX) with CoXML.

1. INTRODUCTION

With the growing popularity of the Extensible Markup Language (XML) [12], more and more information is stored and exchanged in the XML format [1]. XML is essentially a textual representation of hierarchical (tree-like) data where a meaningful piece of data is bounded by matching starting and ending tags, such as <name> and </name>.

To cope with the tree-like structures in the XML model, several XML-specific query languages have recently been proposed (e.g. Xpath [15], Quilt [3], XML-QL [13] and XQuery [16] etc.). All these XML query languages aim at only the exact matching of query conditions. Answers are found when those XML documents match the given query conditions *exactly*. However, this may not always be the case in the XML model. To remedy this condition, we are developing a query relaxation framework for searching answers that match the given query conditions *approximately*. *Query relaxation* enables systems to relax the user query to a less restricted form to derive approximate answers. Such a technique has been successfully used in the relational databases (e.g. CoBase [5]) and has proven to be a valuable technique for deriving approximate answers.

In the XML domain, the need for query relaxation increases since the flexible nature of the XML model allows varied structure or values, and the non-rigid XML tag syntax enables users to embed a wealth of meta-information in XML documents. Query relaxation is more important for the XML model [14] than for the relational model because:

1. The schema in the XML model [14] is substantially larger and more complex than the schema in the relational model. Therefore, it is often unrealistic for users to understand the full schema and compose very complex queries. Thus, it is critical to be able to relax a user's query when the original query yields null or insufficient answers.

2. As the number of data sources available on the web increases, it is becoming increasingly common to build systems that gather data from the heterogeneous data sources. The structures of these data sources are different although using the same ontology for similar contents. Therefore, the capability to query against differently-structured data sources is becoming increasingly important [8, 9]. Query relaxation allows a query to relax its structure and matches data sources with relaxed structures.

Query relaxation in the XML model introduces new challenges than the relational database. Query relaxation in the relational model is basically focused on the value aspect. For example, for a relational query "*find a person with a salary range 50K – 55K*", if there is no answer or not enough answers available, it can be relaxed to a query "*find person with a salary range 45K - 60K*." In the XML model, in addition to the value relaxation, a new type of relaxation called *structure relaxation* is introduced. Structure relaxation relaxes the nodes and/or edges of a query tree.

Further, we shall develop a methodology to provide automatic structure relaxations and to evaluate the effectiveness of XML structure relaxations.

A knowledge-based relaxation index structure called XML Type Abstraction Hierarchy (X-TAH) is introduced to provide scalable XML query relaxations. X-TAH is a hierarchical tree-like knowledge structure that builds multi-level knowledge representation about the XML data tree. X-TAH can be used to guide the XML query relaxation process.

The paper is organized as follows: Section 2 provides some background information which cover XML data model, query model and XML query relaxation types. Section 3 describes the system architecture that we are using for this year's INEX retrieval task. Query execution and query relaxation processes are presented in Section 4.

The experimental performance is discussed in Section 5. Finally we summarize our participation effort in INEX 03 and discuss future works in Section 6.

2. XML Data Model and Query Relaxation

We first briefly describe the XML data and query model and then introduce query relaxation types in the XML model.

2.1 Data Model and Query Model

An XML document can typically be represented as an ordered, labeled tree where nodes correspond to elements and attributes, and edges represent element inclusion relationships. Each node has a label which is the tag name of its corresponding element or attribute. Elements' text content or attributes' values become the values of their corresponding nodes. Similarly, a query against an XML document can be represented as a tree with two types of edges: a parent-child edge denoted as "/", or an ancestor-descendant edge denoted as "//".

Note that in the paper, we treat an attribute as a sub-element of an element and a reference IDREF as a special type of value.

2.2 Query Relaxation Types

In the XML model, there are two types of query relaxations, value relaxations and structure relaxations:

2.2.1 Value Relaxation

In the XML context, value relaxation involves expanding the value scope of certain nodes to allow the matching of additional answers. A value can be relaxed to a range of numeric values or a set of non-numeric values. Figure 1 illustrates an example of numeric value relaxation and an example of non-numeric value relaxation. The query in Figure 1b is a relaxed query for that in Figure 1a by a numerical value relaxation, and the query in Figure 1d is a relaxed query for that in Figure 1c by a non-numeric value relaxation.

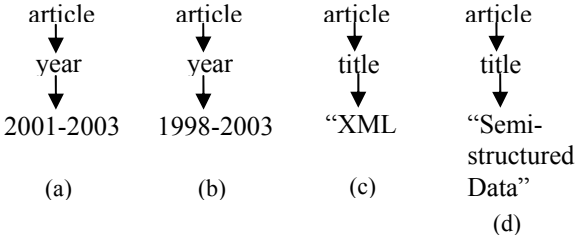


Figure 1: An example of value relaxation

2.2.2 Structure Relaxation

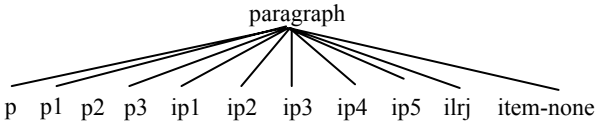
In XML context, structural relaxation is the process of relaxing the nodes and/or edges of a query tree. After the relaxation, a new query tree may have a different structure than the original query tree. There are three types of structural relaxations.

1) Edge Relaxation

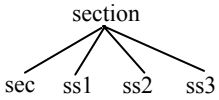
In an edge relaxation, a parent-child edge ('/') in a query tree can be relaxed to an ancestor-descendant edge ('//'). The semantics of edge relaxation is that while the original query finds answers with only a parent-child relationship, the new query will be able to find answers with an ancestor-descendent relationship which is a superset of a parent-child relationship. For example, query topic 69 */article/bdy/sec[about(.//st, "Information Retrieval")]* can be relaxed to */article/bdy//sec[about(.//st, "Informaiton Retrieval")]* by relaxing the structural relationship between node *bdy* and *sec* from "/" to "//".

2) Node Re-label

In this relaxation type, certain nodes can be re-labeled to similar or equivalent tag names according to the domain knowledge. For example, in INEX 03, domain experts have identified sets of equivalent tags as shown in Figure 2. With this domain knowledge, the query */article/bdy//sec [about(., "XML")]* can be relaxed to */article/bdy//section [about(., "XML")]* by generalizing node *sec*'s label to *section*.



(a) Equivalent names for paragraph-like tags



(b) Equivalent names for section-like tags

Figure 2: Domain knowledge for equivalent tags in INEX 03

3) Node Deletion

In this relaxation type, certain nodes can be deleted while preserving the "superset" property. When a node *v* is a leaf node, it can simply be removed. When *v* is an internal node, the children of node *v* will be connected to the parent of *v* with ancestor-descendant edges ("//"). For example, a query */article/bdy/sec[about(., "Information Integration")]* can be relaxed to */article//sec [about(.,"Information Integration")]* by deleting internal node *bdy* so that a section in an article's appendix talking about "Information Integration" can also be returned as an approximate answer.

3. The CoXML Framework

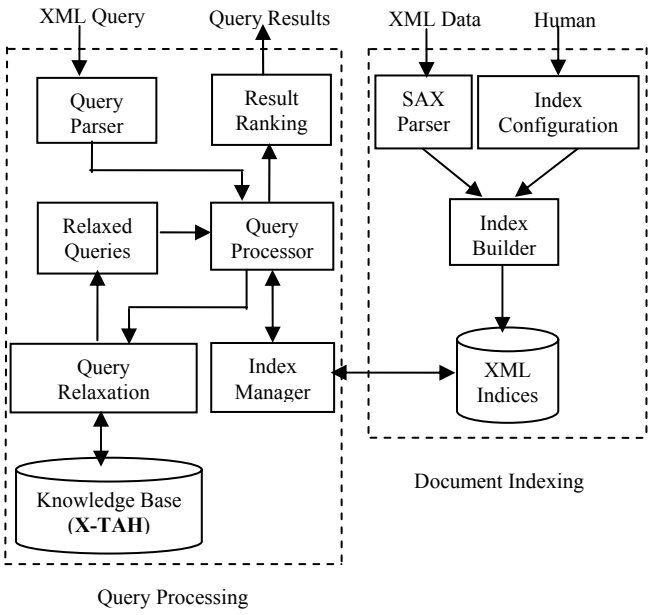


Figure 3: The CoXML System Architecture

Figure 3 shows the cooperative XML query answering system (CoXML) which performs two types of functions: document indexing and query processing as discussed in the following:

Document Indexing

While a *SAX parser* parses XML documents, the *Index Builder* builds indices on these data based on the index configurations provided by the *Index Configurations* module (Section 3.1). The *Index Builder* module builds several types of indices (refer to section 3.2) for query processing.

Query Processing

An XML query is first parsed by the *Query Parser* to check its correctness. If the query is invalid, it will be returned to the user with the error information. Otherwise, the *Query Processor* will consult the *Index Manager* to load the corresponding indices for processing the query. If there are enough XML answers returned, the *Result Ranking* module will rank the results based on their relevancy to the query and return the ranked results to the user. If there is no answer or the available answers are not enough, the X-TAH that resides in the *Knowledge Base* will guide the *Query Relaxation Manager* to relax the query. Then the relaxed queries will be resubmitted to the *Query Processor* for answering. This process will be repeated until there are enough answers available or the query is no longer relaxable.

3.1 Index Configurations

XML documents in the INEX document collections are document-centric. There are two types of tags in these documents: 1) semantic tags, and 2) presentation tags. Semantic tags describe the semantics of the elements. For example, in the XML document fragment example in Figure 4, <article>, <bdy> and <sec> etc. are semantic tags for they encode the semantics of the elements, while <scp> is a presentation tag because it only senses the purpose of displaying: informing the browser to display the characters bounded by <scp> and </scp> in lower cases.

```
1. <article>
2.   <bdy>….
3.     <sec>
4.       <st>K<scp>NOWLEDGE</scp> B<scp>ASED</scp>
5.             S<scp>EMANTIC</scp> T<scp>EMPORAL</scp>
6.              I<scp>MAGE</scp> M<scp>ODEL</scp>
7.       </st> …
8.     </sec> ….
10.  </bdy> ….
11. </article>
```

Figure 4 : An XML document fragment

Presentation tags sometimes are undesirable in query processing. For example, suppose a user wants to find an article that has a section with title containing a keyword "knowledge", which can be expressed in XQuery as *//article [contains(//sec/st, "knowledge")]*. Intuitively, the XML document fragment in Figure 4 is an answer because the title of the article's section (Line 4-7 in Figure 4) is "Knowledge Based…". However, if we do not ignore the markup <scp> and </scp> (Line 4), it will not be returned as an answer since the presentation tag <scp> separates "K" from "NOWLEDGE".

To support keyword and phrase matching in document-centric XML documents, it is necessary to ignore such presentation tags [2]. The set of ignorable tags during indexing is listed in the *Index Configurations* module (Figure 3). For XML documents in the INEX document collections, the list of ignorable tags for index configurations is shown in Table 1.

Category	Ignorable Tags
List-items	item-bold, item-both, item-bullet, item-diamond, item-letpara, item-mdash, item-numpara, item-roman, item-text
Lists	li, l1, l2, l3, l4, l5, l6, l7, l8, l9, la, lb, lc, ld, le, list, numeric-list, numeric-rbrace, bullet-list
Text font, style, size, emphasis etc	ss, tt, b, ub, it, rm, scp, u, sub, super, large, ariel, bi, bu, bui, cen, rom, h, h1, h1a, h2, h2a, h3, h4

Table 1: Index configurations used in INEX

3.2 Indexing XML documents

Each node in an XML data tree is represented by a triple (ID, size, level), where *ID* uniquely identifies the node in the XML document collections, *size* indicates the size of the sub-tree rooted at this node and *level* describes the node's height in the data tree. The advantage of this encoding scheme is that the hierarchical relationship (either parent-child or ancestor-descendant relationships) between any pair of nodes can be checked in constant time.

Values of nodes are processed in the following three steps:

1) A stop words list is used to delete words with weak discriminative powers (such as articles, pronouns, conjunctions and auxiliary words). This step significantly reduces the index size.

2) The Lovins stemmer [7] is used to derive word stems. For example, the stem for "clustering", "clusters" and "clustered" is "cluster". Word stemming reduces the index size and also supports keyword matching.

3) Each stem is represented as a pair of (ID, pos), where *ID* is the unique identifier of a node that contains this stem and *pos* is its relative position in the node's value. We assign a node's ID to its corresponding value to avoid the expensive join operations between nodes and their values and keep each stem's relative position in a node's value to support phrase matching.

To support efficient and scalable query processing, the *Index Builder* builds several types of indices, as listed below:

- Tag Name Index (tag name → name identifier)

  Each tag name *s* is mapped to a unique name identifier (NID) to minimize index size and computation overhead by eliminating string comparisons.

- Node Index (name identifier → (ID, size, level))

  Each name identifier is mapped to a set of nodes (in the form of (ID, size, level)) whose labels are the same as the one represented by the name identifier.

- Inverted Stem Index (stem s → (ID, pos))

  Each stem *s* is mapped to a set of pairs (ID, pos), where *ID* is the unique identifier of the node that contains stem *s* and *pos* is its relative position in the node's value.

- Text Size Index (ID → text size)

  For each node that has a value, its ID is mapped to the number of words it contains. The text size index is useful for result ranking (refer to section 4.4).

The indices for the XML document fragment in Figure 5 are shown in Table 2, which consist of four indices: a tag name index (Table 2.a); a node index (Table 2.b); an inverted stem index (Table 2.c) and a text size index (Table 2.d).

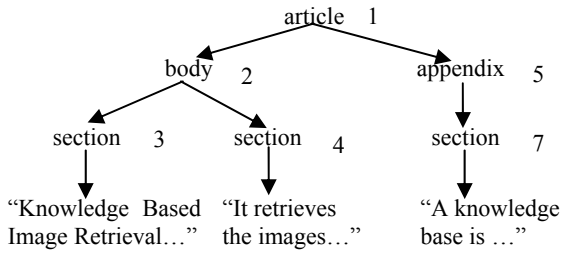


Figure 5 : An XML document fragment

Tag Name	NID
article	0
appendix	1
body	2
section	3

(a) A tag name index

NID	Nodes (ID, size, level)
0	(1, 5, 1)
1	(5, 1, 2)
2	(2, 2, 2)
3	(3, 0, 3) (4, 0, 3) (7, 0, 3)

(b) A node index

Stem	(ID, pos) pairs
bas	(3, 1) (7, 2)
imag	(3,2) (4, 3)
knowledg	(3, 0) (7, 1)
retrief	(3, 3) (4, 1)

(c) An inverted stem index

ID	Text Size
1	1000
2	600
…	…
7	100

(d) A text size index

Table 2: Indices for the XML fragment in Figure 5, a) maps a tag name to a unique name identifier; b) maps a name identifier to a set of nodes in the format of (ID, size, level); c) maps a stem to a set of (ID, pos) pairs d) maps a node ID to its text size

3.3 Knowledge Base

Knowledge Base is an important part in the system architecture, which facilitates XML query relaxation and consists of the following two parts:

1) Domain Ontology

Domain ontology provides the semantic relationships among the tag names in an XML dataset, such as groups of equivalent or similar tag names which can guide the node re-label. For example, Figure 2 lists two sets of equivalent or similar tag names for INEX 03, one for section-like nodes (Figure 2a) and another for paragraph-like nodes (Figure 2b).

2) Knowledge-based XML Relaxation Index (X-TAH)

Query relaxation enlarges the search scope of query conditions which can be accomplished by viewing a query object at a higher conceptual level. To support query relaxation in the XML model, we are generating two types of relaxation index structures, XML Type Abstract Hierarchy - X-TAH: value relaxation index and structure relaxation index for guiding value and structure relaxations.

An X-TAH is a tree-like multi-level knowledge representation of the structure and value characteristics of an XML data tree. X-TAH can be automatically generated from a set of objects based on their inter-object distance [8]. Objects in an XML value relaxation index are values of XML elements and attributes, while objects in an XML structure relaxation index are structure fragments of XML data trees. X-TAH has two types of nodes: internal nodes and leaf nodes. This differentiates it from a traditional cluster which has no internal nodes. An internal node in an X-TAH is a representative that summarizes the characteristics of all the objects in that cluster, while a leaf node is an object that is either a value (in the XML value relaxation index) or a structure fragment of an XML data tree (in the XML structure relaxation index). For example, Figure 6 is an X-TAH for the values of *//fig//no* in the INEX document collections. Figure 7 is an X-TAH for structure relaxation for *//article/*//section*.

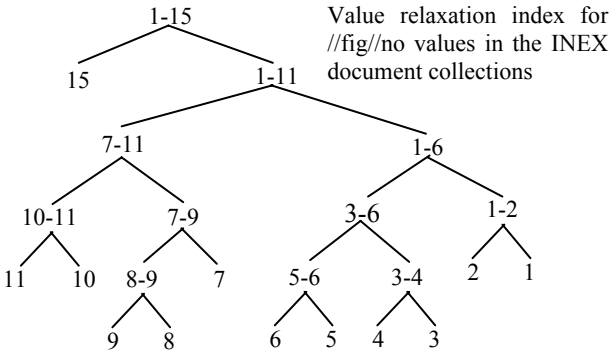
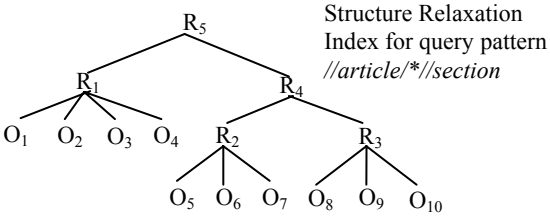


Figure 6: An Example of value relaxation index (X-TAH)

4. Query Processing and Relaxation

The control flow for processing the INEX query topics is illustrated in Figure 8. First, each topic is translated into a tree representation that the *Query Processor* can follow and process. Next, the query is executed to produce a set of results. If there are enough answers produced, the *Result Ranking* ranks each result based on its relevancy to the query. Otherwise, the *Query Relaxation Manager* relaxes the query based on an X-TAH (*Knowledge Base*). The relaxed queries are then submitted to the *Query Processor* for deriving approximate answers. This process will iterate until either there are enough answers or the query is no longer relaxable.



O_1: //article/bdy/sec O_2: //article/bdy/sec/ss1
O_3: //article/bdy/sec/ss1/ss2 O_4: //article/bdy/sec/ss1/ss2/ss3
O_5: //article/bm/sec O_6: //article/bm/sec/ss1
O_7: //article/bm/sec/ss1/ss2 O_8: //article/bm/app/sec
O_9: //article/bm/app/sec/ss1 O_{10} : //article/bm/app/sec/ss1/ss2
R_1 : //article/bdy//section R_2 & R_4: //article/bm//section
R3 : //article/bm/app//section R5 : //article//section

Figure 7: An Example of structure relaxation index (X-TAH)

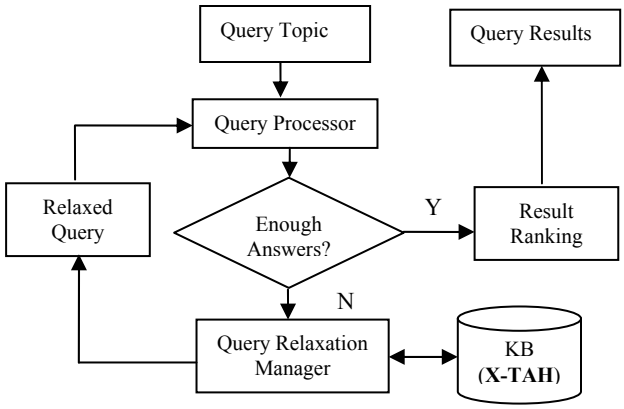


Figure 8: The control flow of CoXML query processing

4.1 Transformation of INEX Query Topics

The topic transformation can be accomplished by the following three steps:

1) Translating each INEX query topic expressed in XPath [15] into a tree representation. This is a straightforward step as most XPath expressions use tree structures.

2) Categorizing each term and phrase in the <title></title> part of a query into one of the three categories as defined below:

- PREFER (P)

  Any term or phrase prefixed by "+" belongs to this category.

- REJECT (R)

Any term or phrase prefixed with "-" or appearing after "!=" operator belongs to this category.

- NORMAL (N)

Any term or phrase not in the PREFER or REJECT category is classified in the NORMAL category.

3) Expanding a query's value predicates in the <title></title> part with terms and phrases in the <keyword></keyword> part that do not appear in the <title></title> part. Such terms and phrases are in the KEYWORD (K) category.

For example, the tree representation for the INEX 03 query topic 89 (Figure 9) with classified terms and phrases and expanded keyword value predicates is shown in Figure 10.

<inex_topic topic_id="89" query_type="CAS" ct_no="123">
<title>
//article[about(./bdy,'clustering "vector quantization" +fuzzy +k-means +c-means -SOFM -SOM')]//bm//bb[about(.,'"vector quantization" +fuzzy clustering +k-means +c-means') AND about(./pdt,'1999') AND ./au/snm != 'kohonen']
</title>
<description>
Find articles about vector quantization or clustering and return bibliography details of cited publications about clustering and vector quantization methods, from recent years, not authored by Kohonen.
</description>
<narrative>
Bibliography elements of publications, preferably from around 2000 (1996 to 2002 is fine, descending relevance thereafter). Preferred documents have reference to k-means or c-means clustering. Not interested in publications where the author is Kohonen, or in his work on self organizing feature maps (SOM SOFM). The citing article and the cited publication should be about clustering or vector quantization methods.
</narrative>
<keywords>
cluster analysis,adaptive clustering,Generalized Lloyd, LBG, GLA
</keywords>
</inex_topic>

Figure 9: INEX 03 Query Topic 89

4.2 Query Processing

After the topic translation, a query tree is sent to the *Query Processor* for execution. Several query processing strategies have been proposed for XML tree pattern queries [e.g.11, 10]. The basic idea of these query processing strategies is to decompose an XML tree pattern query into a set of basic structural relationships (i.e. parent-child relationship and ancestor-descendant relationship) between pairs of nodes. Query answers can be derived by first matching each of these basic structural relationships and then combing these basic matches. Matching each structural relationship is usually based on XML indices and structural join algorithms [10, 4 etc.]. We leverage on these query processing strategies for deriving the exact matched query answers with additional care for processing value constraints in a query tree.

As illustrated in section 4.1, each term and phrase in the <title></title> and <keyword></keyword> part of a query topic is classified into one of the four categories. The semantics for terms and phrases in the PREFER, NORMAL and KEYWORD categories are quite clear. However, the semantics for terms and phrases in the REJECT category is context sensitive. If a value predicate in a query contains only REJECT category terms and phrases, it is interpreted as "strictly MUST NOT". Otherwise it means "fuzzy MUST NOT". For example, for the query tree in Figure 10, the semantics for "R: SOFT, SOM" under node *bdy* is different from that for "R: Kohonen" under node *snm*. The semantics for the first one is that if an article's body (*bdy*) contains either term "SOFT" or "SOM", it is still an answer but with lower relevancy. However, the semantics for the second one is that if an author's surname (*snm)* contains the term "Kohonen", it will not be returned as an answer.

4.3 Query Relaxation

If there is no answer or not enough available answers, the *Query Processor* will call the *Query Relaxation Manager* to relax the query in the following three steps:

1) A set of relaxable conditions as well as their respective relaxation order are generated. For example, for INEX 03 query topic 85, *//article[.fm//yr >= 1998 and .//fig//no >9]//sec[about(.//p, 'VR, "virtual reality", "virtual environment", cyberspace "augmented reality")]*, the set of relaxable conditions and their relaxation order may be assigned as: relaxing the value of figure numbers (*//article//figure/no > 9*) first and then relaxing the value of the article's year (*//article/fm/yr >= 1998*).

2) For each relaxable condition, a relaxation index (X-TAH) will be selected to guide the relaxation process. The *Query Relaxation Manager* will first examine the internal representatives to find the one that contains the exact or closest match against the relaxable condition and relax the query condition accordingly. There are two types of operations in an X-TAH: i) Generalization - moving up the hierarchy to enlarge the search scope; and ii) Specification – moving down the hierarchy to narrow the search scope. The query relaxation process may incur a sequence of Generalization and Specification operations.

3) The relaxed queries will be sent to the *Query Processor* to derive approximate answers. This relaxation process will continue until there are enough answers or the query is no longer relaxable.

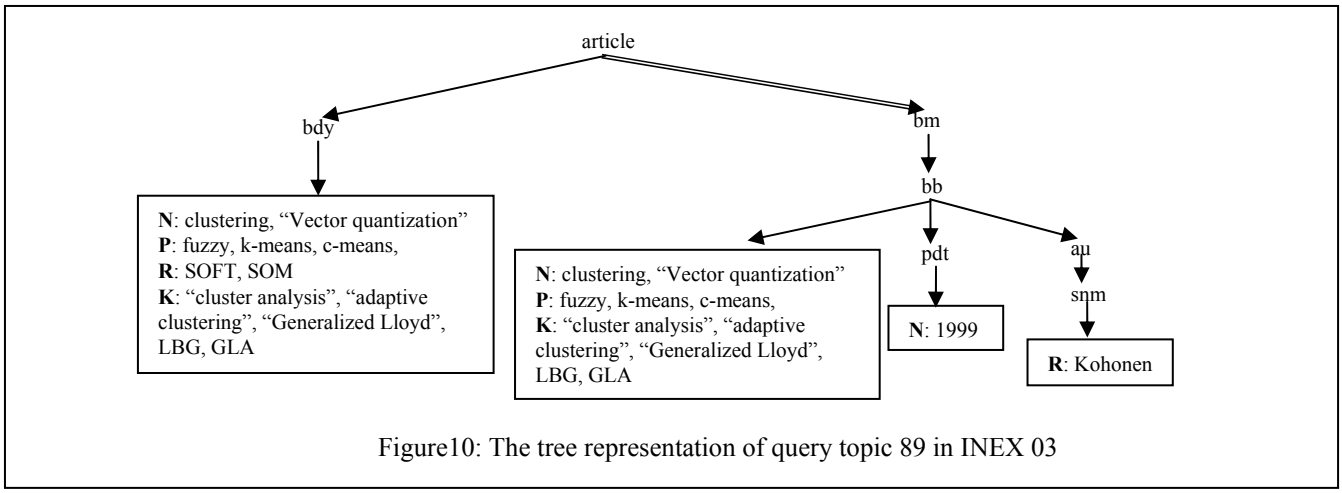


Figure10: The tree representation of query topic 89 in INEX 03

For example, in the query topic 85, to relax the query condition, *//article//figure//no > 9*, the *Query Relaxation Manager* will select the value relaxation index in Figure 6 to guide the relaxation process. The system first locates the closest matched internal representative, which is 8-9, and then relaxes the query condition to *//article/figure//no > 8* to derive approximate answers.

Similarly, to relax the structure constraint *//article/bdy/sec* in the query topic 69 (i.e. */article/bdy/sec[about(.//st, '"information retrieval"')]*), the *Query Relaxation Manager* will first locate the closest matched internal representative, which is *//article/bdy//sec*, and will relax the query topic to *//article/bdy//sec[about (.//st, '"information retrieval"')]*.

4.4 Result Ranking

The query results are ranked by the *Result Ranking* module before returning them to the user. Query results are ranked according to the following priorities: first query results from the original query and then approximate answers from the relaxed queries. The approximate answers are further ranked according to the relaxation order. For example, for the query topic 85, there are two relaxation conditions: 1) *//article//fig//no > 9* and 2) *//article/fm//yr > 1998*. The relaxation order between them is to relax the first condition and then the second one. As a result, the approximate answers for the first relaxation condition are ranked before the approximate answers for the second relaxation condition.

For the query results in the same category, they are ranked according to the following formula:

$$rank_u = \sum_{i = P, N, K, R} \left(\frac{w_i}{|C_i|} \sum_{j=1}^{|C_i|} \frac{frequency\ of\ term_{ij}}{Text\ Size\ of\ node\ u} \right)$$

where w_i is the weight assigned to one of the four categories C_i (i = P, N, K R); $|C_i|$ is the total number of stems (a phrase is counted as a term) in the category; *frequency of* $term_{ij}$ is the number of occurrence of $term_j$ from category C_i in node *u*; and *Text Size of node u* refers to the total number of words in node *u*, which can be accessed from the text size index.

5. Experimental Observations

We shall now discuss the experimental results based on two performance measurements: index size and query execution times.

The indices for all the INEX document collections occupy about 1.2GB, which is roughly about twice the size of the XML document collections. Four types of indices are built by the *Index Builder*: tag name index, node index, text size index, and inverted stem index. The first three are relatively small and the last one is quite large.

Query processing time depends on the following factors:

1) Number of stems and phrases in a query and their corresponding frequency in the XML data.

The query processing time depends on the number of stems and phrases a query contains and their corresponding frequencies in XML documents. More frequent stems and phrases require longer query processing time than less frequent ones.

2) Number of structure constraints in a query and their corresponding frequency in the XML data.

The required query processing time is sensitive to the number of structure constraints a query contains. It is also sensitive to their frequencies in XML documents. For example, a less frequent structure constraint, Q_1 *//article/fm//pdt*, can be processed much faster than a more frequent one Q_2 *//article/bdy//p*. (Q_1 returns the publication date (*pdt*) element of an article in its front

matter part (*fm*) and Q_2 returns the paragraph (*p*) elements of an article in its body part (*bdy*)).

3) The level of query relaxation and the number of relaxable conditions existed in the query.

The more relaxable query conditions a query topic contains, the longer it takes to derive the approximate answers.

Depending on the complexity of its value and structure constraints, a content-and-structure (CAS) query takes from several seconds to over a minute to get exact matched answers. For a relaxable query, it might take several minutes to generated the relaxed queries and derive approximate answers.

6. Summary and Future Works

In this paper, we describe how we index INEX XML documents and extend the query relaxation technique to the XML model to support cooperative XML query answering.

During our INEX 03 investigation, several problems were discovered, which needs future investigations:

1) Index Configurations

Our current index configuration only contains a list of ignorable tags. We plan to support other index configurations, such as ignorable annotations in which both elements and their value can be ignored.

2) Uniform Value Index Scheme

In our current system, we index the elements' text content and attributes' values in XML documents uniformly. Each non-stop word is stemmed and is built an inverted stem index without considering of the value's characteristics. Such an index approach sometimes may derive undesirable results. For example, for a content-only (CO) query "*web, internet*", the document fragment "*<author> <snm>webb </snm></author>*" will be returned as an answer since "webb" and "web" share the same stem: "web". To avoid such undesirable results, we plan to work on a configurable value index framework which supports multiple value treatment options and index types based on the value's characteristics.

3) Ranking Functions

Our current system only supports relative ranking. Ranking functions for query results needed to be investigated to provide more user and context sensitive ranking.

4) Query Relaxation Language

No explicit relaxation constructs is available in a query topic for specifying the relaxable query conditions as well as their relaxation order. We plan to develop a cooperative query language that enables users to specify relaxation constructs in the queries.

ACKNOWLEDGEMENT

This work is supported by NSF Award IIS#: 0219442.

REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. Data on the web: from relations to semistructured data and XML. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1999.

[2] S. Amer-Yahia, M. Fernandez, D. Srivastava, Y. Xu. Phrase Matching in XML, VLDB 2003

[3] D. Chamberlin, J. Robie, and D. Florescu. Quit: An XML query language for heterogeneous data sources. In WebDB, May 2000

[4] S. Chien, Z. Vagena, D. Zhang, V. J. Tsotras, C. Zaniolo. Efficient Structural Joins on Indexed XML Documents, VLDB 02

[5] W. W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson. CoBase: A Scalable and Extensible Cooperative Information System. J. Intelligent Information Systems (JIIS), 6(2/3):223-259, May 1996.

[6] S. Liu and W. W. Chu. A Knowledge-Based Approach for Cooperative XML Query Answering, UCLA CS Dept. Technical Report, 2003

[7] J.B. Lovins. Development of a Stemming Algorithm. In Mechanical Translation and Computational Liguistics, 11(1-2), 11-31, 1968

[8] Y. Kanza, W. Nutt, and Y.Sagiv. Queries with Incomplete Answers over Semi-structured Data. In ACM PODS, 1999

[9] Y. Kanza and Y.Sagiv, Flexible Queries over Semi-structured Data, In ACM PODS, 2001

[10] D. Srivastava, S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. In ICDE 2002

[11] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, G. Lohman. On Supporting Containment Queries in Relational Database Systems, SIGMOD 2001

[12] XML http://www.w3.org/XML/

[13] XML-QL http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/

[14] XML Schema http://www.w3.org/xml/Schema

[15] XPATH http://www.w3.org/TR/xpath

[16] XQuery http://www.w3.org/TR/xquery