CoXML: A Cooperative XML Query Answering System

Shaorong Liu and Wesley W. Chu

University of California, Los Angeles, Computer Science Department
Los Angeles, California, 90095 USA
{sliu, wwc}@cs.ucla.edu

Abstract

XML has become the standard format for information representation and data exchange. The heterogeneity nature of XML data creates the need for approximate query answering. In this paper, we present a cooperative XML (CoXML) system that provides user-specific approximate query answering. The key features of the system include: 1) a query relaxation language that allows users to specify approximate search conditions and to control the approximate search process; 2) a relaxation index structure for systematic and scalable query relaxation; and 3) both content and structure similarity metrics for evaluating the relevancies of approximate answers. We evaluate our system with the INEX 05 test collections. The results reveal that our language allows users to effectively specify their relaxation specifications and thus enables the system to provide answers with more relevancy. The results also demonstrate the effectiveness of the similarity metrics. Further, compared to other systems in INEX 05, our relaxation features enable our system to retrieve approximate answers with more relevancy.

1 Introduction

The increasing use of XML in scientific data repositories, digital libraries and Web applications has increased the need for flexible and effective methods to search these repositories. For example, the Initiative for the Evaluation of XML retrieval (INEX)[4] was established in 2002 and has prompted researchers worldwide to evaluate the effectiveness of XML retrieval methods.

There are two major paradigms for searching XML data: content-only queries and structure-and-content queries. Queries with both structure and content conditions are more expressive and thus yield more accurate searches than content-only queries. XML structures, however, are often complex and heterogeneous due to the flexible nature of the XML data model. For example, there are about 170 distinct tags and more than 1000 distinct paths in the INEX XML dataset. Thus, it is difficult and unrealistic for users to completely grasp the structural properties of the data and specify the exact structure constraints in queries. To remedy this condition, we propose a cooperative XML (CoXML) query answering system that relaxes query conditions to less restricted forms to derive approximate answers.

Approximate query answering has been well-studied in the relational model(e.g., [12]), which typically enlarges the scopes of value conditions in queries. In the XML model, queries contain both content and tree-like structure conditions. Thus, in addition to relaxing content conditions, we also need to approximately match structure conditions in queries to derive approximate answers.

In addition, approximate query answering is usually user-specific. Different users may have different specifications about which conditions to relax and how to relax such conditions for the same query. The first step towards user-specific approximate query answering is to provide a query language that allows users to include their personalized relaxation specifications. Most existing XML query languages (e.g., XPath, XQuery), however, only allow the specifications of exact conditions. Thus, we need a query language for user-specific approximate searching.

Further, with the explosive increase in the amount of XML data available, it is essential to have a systematic and scalable method for approximate query answering. Most existing work on XML relaxation(e.g., [6]) derive relaxed queries online based on the relaxation types (as introduced in Section 2.3) applicable to a query, which may not be scalable. In addition, existing work do not provide relaxation controls during approximate searching, which may yield undesired approximate answers.

Finally, after query relaxation, a list of approximate answers will be generated. These approximate answers shall be ranked based on their relevancies to both the structure and content conditions in queries. Most existing ranking models (e.g., [21], [18], [14]) only measure the content similarities between queries and answers, and thus are in-

**Proceedings of the 32nd VLDB Conference,
Seoul, Korea, 2006**

adequate for ranking approximate answers that use structure relaxations. Recently, [7] provided seminal studies on XML structure scoring properties and proposed a family of structure scoring functions similar to the *tf***idf* used in IR. The functions, based on the frequencies of query structures in data, guarantee that the closer an answer is to a query, the higher the structure score of the answer will be. XML data provides rich semantics in addition to structure frequencies. Thus, it is essential to fully utilize data semantics in designing scoring functions to improve accuracy.

In this paper, we address the challenges as follows:

First, we propose an XML approximate searching language that enriches the standard tree-structured queries (i.e., *twigs*) with relaxation constructs and controls. Relaxation constructs specify which conditions to approximate during search; and relaxation controls guide the approximate search process. These new features in our language allow users to specify their approximate search more effectively, which in turn enables the system to provide answers with more relevancy.

Second, we introduce a relaxation index structure called XML Type Abstraction Hierarchy (XTAH) to provide systematic and scalable query relaxation. Given a specific twig, XTAH clusters its structurally relaxed twigs into multi-level groups based on relaxation operations (as introduced in Section 2.4) and distances. Each group in an XTAH contains a set of twigs that correspond to a specific relaxation specification. Thus, XTAH can efficiently relax a query based on the query's relaxation specifications by consulting the corresponding group. Further, due to its multi-level organization, XTAH can relax a query at different granularities by traversing up and down the hierarchy.

Third, we evaluate XML structure similarity as inversely proportional to the total cost of the relaxation operations in a way similar to tree editing distances. We propose a semantics-oriented model for measuring a relaxation operation cost. The cost model differentiates twigs that use different relaxation operations, even though the twigs may have the same occurrence frequency in data. We also propose a function that combines structure distance and content similarity to measure the overall relevancy.

We have implemented the CoXML system, which takes a relaxation-enabled query, derives approximate answers guided by relaxation indexes, and ranks the answers based on their structure and content relevancies. We evaluate the quality of approximate query answering provided by CoXML using the INEX test collection, the benchmark collection for evaluating the effectiveness of XML search.

The rest of the paper is organized as follows. In Section 2, we introduce the foundation for XML relaxation. In Section 3, we present an overview of our CoXML system architecture and we introduce the relaxation language in Section 4. In Section 5, we propose a relaxation index structure and describe how to use it to guide relaxation. In Section 6, we present the semantics-oriented structure distance and the overall ranking function. Section 7 present empirical evaluation studies and Section 8 provides related work. We conclude the paper in Section 9.

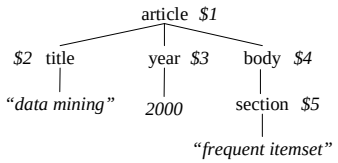


Figure 1: A sample XML twig

2 Foundation of XML Relaxation

2.1 XML Data Model

We model an XML document as an ordered, labeled tree where each element is represented as a node and each element-to-sub-element relationship is represented as an edge between the corresponding nodes. We represent each node u as a triple (*id*, *label*, <*text*>), where *id* uniquely identifies the node, *label* is the name of the corresponding element or attribute, and *text* is the corresponding element's text content or attribute's value. *Text* is optional because not every element has a text content.

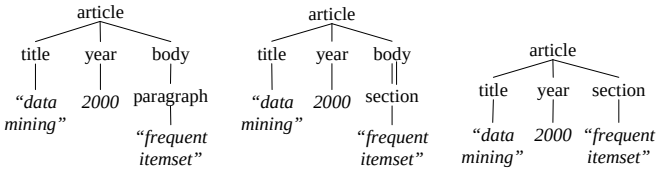
2.2 XML Query Model

A fundamental construct in most existing XML query languages is the tree-pattern query or *twig*, which selects elements and/or attributes with specified tree structures. Thus, we use the twig as our basic query model. Similar to the tree representation of XML data, we model a twig as a rooted tree. For example, Fig. 1 illustrates a sample twig, which searches for articles with a title on "*data mining*", a year in *2000* and a body section about "*frequent itemset*." The twig consists of five nodes, where each node is associated with an unique *id* next to the node. The IDs of the twig nodes can be skipped when the labels of all the nodes are distinct. The text under a twig node, shown in italic, is the content (or value) constraint on the node, which is to be processed in a non-Boolean style.

Now we shall introduce the notations for the twigs. Given a twig T, we use $T.root$, $T.V$ and $T.E$ to represent its root, nodes and edges respectively. Given a twig node v ($v \in T.V$), we use $v.label$ to denote the name of the node's corresponding element or attribute. An edge from node u to v, denoted as $e_{u,v}$, represents either a parent-to-child (i.e., "/") or an ancestor-to-descendant (i.e., "//") relationship between u and v. The term "twig" and "query tree" will be used interchangeably throughout this paper.

2.3 XML Query Relaxation Types

In the XML model, there are two types of query relaxations: value relaxation and structure relaxation. A value relaxation expands a value scope to allow the matching of additional answers. A structure relaxation relaxes the constraint on a node or an edge in a twig to derive approximate answers. Value relaxation, which has been successfully used in relational models, is orthogonal to structure relaxation. In this paper, we focus on structure relaxation.



(a)Node re-label (b)Edge relax (c)Node delete

Figure 2: Examples of structure relaxations for Fig.1

Many structure relaxation types have been proposed ([16], [22], [6]). We use the following three structure relaxation types, similar to the ones proposed in [6], which capture most of the relaxation types used in previous work.

- **Node Relabel** With this relaxation type, a node can be relabeled to similar or equivalent labels according to domain knowledge. We use $Rel(u, l)$ to represent a relaxation operation that renames a node u to label l. For example, the twig in Fig. 1 can be relaxed to that in Fig. 2(a) by relabeling the node *section* to *paragraph*.
- **Edge Generalization** With an edge relaxation, a parent-to-child edge ('/') in a twig can be generalized to an ancestor-to-descendant edge ('//'). We use $Rel(e_{u,v})$ to represent a generalization of the edge between nodes u and v. For example, the twig in Fig. 1 can be relaxed to that in Fig. 2(b) by relaxing the edge between nodes *body* and *section*.
- **Node Deletion** With this relaxation type, a node may be deleted while preserving the "superset" property. We use $Del(v)$ to denote the relaxation of deleting a node v. When 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 instance, the twig in Fig. 1 can be relaxed to that in Fig. 2(c) by deleting the internal node *body*. Since the root node in a twig is a special node representing the search context, we assume that any twig root cannot be deleted.

2.4 Properties of XML Relaxation

Let us now introduce definitions and lemmas related to query relaxation.

Definition 1 ***Valid Relaxation Operation*** *Given a twig T, a relaxation operation r is valid for T if:*

- *r=$Rel(u, l)$ is valid if $u \in T.V$ and $u.label \neq l$;*
- *r=$Del(u)$ is valid if $u \in T.V$ and $u \neq T.root$;*
- *r=$Rel(e_{u,v})$ is valid if $e_{u,v} \in T.E$ and $e_{u,v}$ = '/';*

We use $r(T)$ to represent the twig transformed from T by applying the relaxation operation r.

Definition 2 ***Valid Relaxation Operation Sequence*** *Given a twig T, a sequence of relaxation operations $s = r_1;...;r_n$ is a valid for T if $\forall\, r_i \in s$, r_i is a valid relaxation operation for T_i, where $T_i = r_i(T_{i-1})$ and $T_0 = T$.*

Given a twig T and a relaxation operation sequence s, we use $s(T)$ to represent the twig obtained by applying each relaxation operation in s to T.

Definition 3 ***Relaxed Twig*** *Given two twigs T_1 and T_2, we call T_2 a relaxed twig of T_1 if there exists a relaxation operation sequence s s.t. s is valid w.r.t. T_1 and $s(T_1) = T_2$.*

Definition 4 ***Redundant Relaxation Operation*** *Given a twig T and a relaxation operation sequence s, a relaxation operation r is redundant in s if $s(T) = s'(T)$, where $s' = s - r$, i.e., a sequence obtained by deleting r from s.*

For example, given the twig in Fig. 1 and a relaxation operation sequence $s = Rel(e_{\$1,\$4});Del(\$4)$, the relaxation operation $Rel(e_{\$1,\$4})$ is redundant in s. This is because applying a single relaxation operation $Del(\$4)$ has the same effect as applying the sequence s to the twig.

Definition 5 ***Non-redundant Relaxation Operation Sequence*** *Given a twig T, a relaxation operation sequence s is non-redundant if each relaxation operation r in s is non-redundant.*

By definition 5, we have the following properties about non-redundant relaxation operation sequences.

Lemma 1 *Given a twig T, a relaxation operation sequence s=$r_1;...;r_n$ is non-redundant if and only if s satisfies the following properties:*

1. *$\forall\, r_i \in s$, if r_i=Rel($e_{u,v}$), then $\forall$ j ($i < j \leq n$), $r_j \neq$ Del(u) and $r_j \neq$ Del(v);*
2. *$\forall\, r_i, r_j \in s$, if r_i=Rel(u) and r_j=Rel(v), then $u \neq v$;*
3. *$\forall\, r_i, r_j \in s$, if r_i=Rel(u) and r_j=Del(v), then $u \neq v$;*

The first property states that if a non-redundant relaxation operation sequence s contains a relaxation operation on an edge $e_{u,v}$, then s cannot contain a deletion of either nodes (i.e., u or v) on the edge. The second property states that a non-redundant relaxation operation sequence cannot contain two node re-label relaxation operations on the same node. Finally, the third property states that a non-redundant relaxation operation sequence cannot contain a node re-label followed by a deletion of the same node.

Since we are only interested in non-redundant relaxation operation sequences, we assume that every relaxation operation sequence is non-redundant. One interesting property about non-redundant relaxation operation sequences is that they are un-ordered. That is, changing the orders among the relaxation operations in a sequence does not affect the relaxed twig produced by the sequence. We formalize this property as follows:

Lemma 2 *Given a twig T and a relaxation operation sequence s, where s is valid for T, let s' be any permutation of the relaxation operations in s, then $s'(T) = s(T)$.*

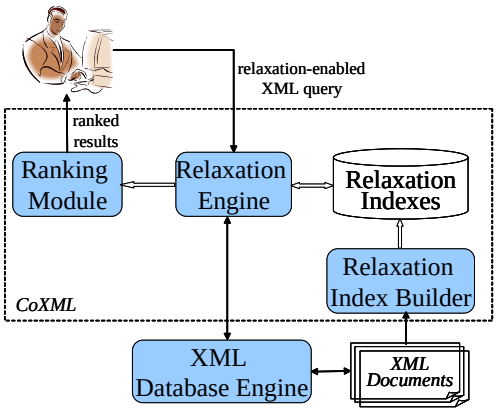


Figure 3: The CoXML system architecture

For example, given the twig in Fig. 1, let r_1 be an operation re-labeling node $5 to *paragraph* and r_2 be an operation relaxing the edge between nodes $4 and $5. The sequence r_1;r_2 has the same relaxation effect as the sequence r_2;r_1 on the twig.

By Lemma 2, given a twig T, we have the following upper bounds for the number of relaxed twigs of T.

Lemma 3 *Given a twig T with m distinct relaxation operations applicable to T, there are at most 2^m different relaxed twigs of T.*

There are at most $\binom{m}{1}$ + ... + $\binom{m}{m}$ = 2^m combinations of relaxation operations. By Lemma 2, each combination generates at most one relaxed twig. Thus, there are at most 2^m relaxed twigs of T.

3 CoXML Architecture

In Fig.3, we present the architecture of our cooperative XML (CoXML) query answering system. The system contains two major parts: off-line components for building relaxation indexes and online components for processing and relaxing queries, and ranking results.

- *Building relaxation indexes*
  The *Relaxation Index Builder* constructs relaxation indexes, XML Type Abstraction Hierarchy (XTAH), for a set of document collections.

- *Processing, relaxing queries and ranking results*
  When a user posts a query, the *Relaxation Engine* first sends the query to an *XML Database Engine* to search for answers that exactly match the structure conditions and approximately satisfy the content conditions in the query. If enough answers are found, the *Ranking Module* ranks the results based on their relevancies to the content conditions[18] and returns the ranked results to the user. If there are no answers or insufficient results, then the *Relaxation Engine*, based on the user-specified relaxation constructs and controls, consults the relaxation indexes for the best relaxed query. The relaxed query is then resubmitted to the *XML Database Engine* to search for approximate answers. The *Ranking Module* ranks the returned approximate answers based on their relevancies to both structure and content conditions in the query. This process will be repeated until either there are enough approximate answers returned or the query is no longer relaxable.

The CoXML system can run on top of any existing XML database engine (e.g., BerkeleyDB[1], Tamino[5], DB2XML[2]) that retrieves exactly matched answers.

4 XML Query Relaxation Language

Query relaxation is often user-specific. Different users may have different preferences for the conditions to be approximated in the same twig. For example, for the twig in Fig. 1, one user may prefer relabeling the leaf node *section* to deleting the internal node *body*, because the user considers an article with a component similar to a *section* node on "frequent itemset" in the body part as being relevant. Another user, however, may consider that only a *section* node may contain deep discussions on "frequent itemset," such as "algorithms for mining frequent itemset." The second user may then reject the relaxation operation of relabeling the node *section*. Therefore, it is essential for an XML approximate query answering system to provide a query language that allows users to include their personalized relaxation specifications (e.g., which conditions to relax and how to relax these conditions).

A number of XML approximate search languages have been proposed. Most extend standard query languages with constructs for specifying approximate content conditions (e.g., XIRQL[14] and NEXI[25]). XXL[23] is a flexible XML search language that includes constructs for users to specify approximate structure and content conditions. It, however, does not allow users to control the relaxation process. It may be often the case that users may want to specify their preferences over multiple relaxable query conditions. In addition, most existing XML query languages do not allow users to specify non-relaxable query conditions. For instance, the second user in the example above may want to specify that the node *section* cannot be relaxed.

Therefore, we propose an XML relaxation language that allows users to specify approximate search conditions and to control the relaxation process. We represent a relaxation-enabled query as a tuple $\mathcal{Q}$ =($\mathcal{T}$, $\mathcal{R}$, $\mathcal{C}$, $\mathcal{S}$), where:

- $\mathcal{T}$ is a twig as described as Section 2.2;
- $\mathcal{R}$ is a set of relaxation constructs specifying which conditions in $\mathcal{T}$ may be approximated when needed;
- $\mathcal{C}$ is a boolean combination of relaxation controls stating how the query shall be relaxed;
- $\mathcal{S}$ is a stop condition indicating when to terminate the relaxation process.

The execution semantics for a relaxation-enabled query $\mathcal{Q}$ is as follows: first, we search for answers that exactly

match the query; and then we test the stop condition $\mathcal{S}$ to check whether there is any need to relax the query. If the stop condition holds, the answers are returned to the user. Otherwise, we repeatedly relax the twig $\mathcal{T}$ based on the relaxation constructs $\mathcal{R}$ and controls $\mathcal{C}$ until either the stop condition $\mathcal{S}$ is met or the twig $\mathcal{T}$ is no longer relaxable.

Given a relaxation-enabled query $\mathcal{Q}$, we use $\mathcal{Q}.\mathcal{T}$, $\mathcal{Q}.\mathcal{R}$, $\mathcal{Q}.\mathcal{C}$ and $\mathcal{Q}.\mathcal{S}$ to represent its twig, relaxation constructs, controls and stop condition respectively. Note that a twig is required to specify a relaxation-enabled query, while relaxation constructs, controls and stop condition are optional. When only a twig is presented in a query, we repeatedly relax the twig to the (next) best relaxation candidate (i.e., a relaxed twig that is (next) closest to the original twig based on a distance function) until the twig is no longer relaxable.

A relaxation construct for a query $\mathcal{Q}$ can be in any of the following forms:

- $Rel(u)$, where $u \in \mathcal{Q}.\mathcal{T}.V$, specifies that the node u may be relabeled when needed;
- $Del(u)$, where $u \in \mathcal{Q}.\mathcal{T}.V$, specifies that the node u may be deleted if necessary;
- $Rel(e_{u,v})$, where $e_{u,v} \in \mathcal{Q}.\mathcal{T}.E$, specifies that the edge $e_{u,v}$ may be generalized when needed.

The relaxation controls for a query $\mathcal{Q}$ is a conjunction of any of the following forms:

- $!r$, where $r \in \{Rel(u),\ Del(u),\ Rel(e_{u,v}) \mid u,\ v \in \mathcal{Q}.\mathcal{T}.V,\ e_{u,v} \in \mathcal{Q}.\mathcal{T}.E\}$, specifies that the node u cannot be relabeled or deleted, or the edge between nodes u and v cannot be generalized;
- $Prefer(u, l_1, ..., l_n)$, where $u \in \mathcal{Q}.\mathcal{T}.V$, states that the node u is preferred to be relabeled to labels l_1, ..., l_n when needed;
- $Reject(u, l_1, ..., l_n)$, where $u \in \mathcal{Q}.\mathcal{T}.V$, specifies a list of unacceptable labels for the node u;
- $RelaxOrder(r_1, ..., r_n)$, where $r_i \in \mathcal{Q}.\mathcal{R}$ ($1 \leq$ i $\leq$ n), specifies the relaxation orders among the relaxation constructs in R: r_1 should be applied before r_2 and so on so forth[1];
- $UseRType(rt_1, ..., rt_k)$, where $rt_i \in$ {*NodeRelabel, NodeDelete, EdgeRelax*} ($1 \leq$ i $\leq$ k ≤ 3), specifies the set of relaxation types allowed to be used. By default, all three relaxation types may be used.

A stop condition $\mathcal{S}$ is either:

- $AtLeast(n)$, where n is a positive integer, specifies the minimum number of answers to be returned; or

[1]Note that the RelaxOrder clause is different from Lemma 2. Lemma 2 states changing the orders among relaxation operations in a sequence does not change the relaxed twig produced by the sequence; while RelaxOrder clause control the orders of relaxed twigs generated.

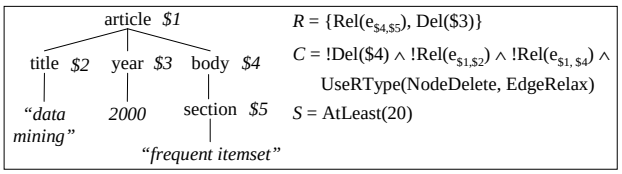


Figure 4: An example of a relaxation-enabled XML query

- $d(\mathcal{Q}.\mathcal{T}, T') \leq \tau$, where T' stands for a relaxed twig and τ a distance threshold, specifies that the relaxation should be terminated when the distance between the original twig and a relaxed twig exceeds the threshold.

Fig. 4 presents an example of a relaxation-enabled query. The minimum number of answers to be returned for the query is 20. If there are insufficient exactly matched answers available, then the edge between *body* and *section* may be generalized and the node *year* may be deleted when needed. The relaxation controls specify that the node *body* cannot be deleted during relaxation. For instance, a *section* about "*frequent itemset*" in an article's appendix part is irrelevant. Also, the edge between *article* and *title*, i.e., *article/title*, and the edge between *article* and *body*, i.e., *article/body*, cannot be generalized. For instance, an article with a reference to another article with a title on "*data mining*" is irrelevant. Finally, only *edge relaxation* and *node deletion* relaxation types may be used.

A relaxation-enabled XQuery, termed as RLXQuery, has been developed. RLXQuery extends the standard XML query language (XQuery) with relaxation constructs, controls and stop conditions presented above. Interested users may refer to [17] for details.

5 XML Relaxation Index

In this section, we present a relaxation index structure that provides systematic and scalable guidance to the relaxations of XML queries.

Given a query $\mathcal{Q}$, there are an exponential number of relaxed queries for $\mathcal{Q}$ (Lemma 3). For example, a twig with 5 nodes may have 10 relaxation operations applicable to the twig, which in the worst case may have 2^{10} (i.e., 1024) relaxed twigs. Therefore, deriving relaxed queries online may not be very efficient when there are many relaxation operations applicable to a query. Further, during query relaxation, we usually need to compute similarities (or distances) between twigs, which often requires information from XML data. For example, in [7], each relaxed twig is associated with an "inverse document frequency," which requires the computations of the number of "documents" that exactly match the relaxed twig. Online computing twig similarities or distances may not be scalable. Thus, we need a systematic approach for scalable query relaxation.

Many queries to the same XML dataset usually share the same or similar tree structures but with different content conditions. For example, the INEX 05 [4] test collection contains 17 distinct content-and-structure queries, in

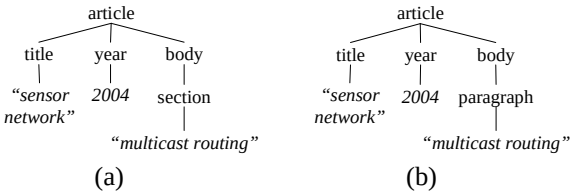


Figure 5: An example of re-using relaxed twigs for relaxing queries with the same tree structure

which 5 queries (about 30%) use the same structure. Structurally relaxed twigs for a query $\mathcal{Q}$ may be re-used to guide the relaxations of queries with the same structure as that of $\mathcal{Q}$. For example, the structure of the twig in Fig. 5(a) is the same as that of the twig in Fig. 1. Thus, the twig in Fig. 2(a), a relaxed twig of Fig. 1, may be used for relaxing the twig in Fig. 5(a), as shown in Fig. 5(b). Further, the similarity between the twig structure in Fig. 1 and that in Fig. 2(a) can be used to measure the structure closeness of the relaxed twig in Fig. 5(b) to that in Fig. 5(a). This motivates us to build relaxation indexes for frequently used twig structures, which may be obtained using data mining techniques. We then use the relaxation indexes to guide the relaxations of queries with the same (or similar) structures as those frequently used twig structures.

5.1 XML Type Abstraction Hierarchy - XTAH

Our goal for constructing relaxation index structures is two-folded: 1)re-using relaxed twigs to guide the relaxation of queries with the same (or similar) structures as those frequently used twig structures; and 2)efficiently searching relaxed twigs for queries with any relaxation constructs and/or controls, which are essentially boolean combinations of relaxation operations.

To this end, we propose an XML relaxation index structure, called XML Type Abstraction Hierarchy (XTAH). An XTAH for a frequently-used twig structure T, denoted as XT_T, is a hierarchical cluster of the relaxed twigs of T based on their corresponding relaxation operations and distances. More specifically, an XTAH is a multi-level labeled cluster with two types of nodes: internal and leaf nodes. A leaf node is a relaxed twig of T. An internal node represents a cluster of relaxed twigs using similar relaxation operations. The label of an internal node is the common relaxation operations (or types) used by the relaxed twigs in the cluster. The higher level an internal node in an XTAH, the more general the label of the internal node, the less relaxed the twigs in the internal node. Such an organization provides two significant advantages: 1)we can efficiently locate relaxed twigs satisfying a given relaxation construct and/or control by traversing to internal nodes whose labels satisfy the relaxation constructs and/or controls; and 2)we can relax a query at different granularities by traversing up and down an XTAH.

Fig. 6 shows a sample XTAH for the twig in Fig. 1.[2] For ease of references, we associate each node in the XTAH with an unique ID, where the IDs of internal nodes are prefixed with *I* and the IDs of leaf nodes are prefixed with *L*. Each leaf node L_i is a relaxed twig of the twig in Fig. 1. Each internal node I_j represents a group of relaxed twigs that are closer to each other and use similar relaxation operations. For example, all the relaxed twigs in the group represented by the internal node I_4 use the relaxation operation $Rel(e_{\$1,\$2})$.

Given a relaxation operation r, let I_r be an internal node with a label $\{r\}$. That is, I_r represents a group of relaxed twigs whose common relaxation operation is r. Due to the tree-like organization of clusters, each relaxed twig belongs to only one cluster (i.e., one internal node), while a relaxed twig may use multiple relaxation operations. Thus, it may be the case that not all the relaxed twigs that use the relaxation operation r are within the group of I_r. For example, the relaxed twig L_5, which uses two operations $Rel(e_{\$1,\$2})$ and $Rel(e_{\$4,\$5})$, is not included in the internal node that represents$\{Rel(e_{\$4,\$5})\}$, i.e., I_6. This is because the relaxed twig L_5 may belong to either group I_4 and group I_6 but is closer to the twigs in group I_4.

To support efficient searching or pruning of relaxed twigs that use a relaxation operation r, we add a virtual link from the internal node I_r to an internal node I_j where I_j is not a descendant node of I_r but the relaxed twigs within I_j use the relaxation operation r. By doing so, relaxed twigs using the relaxation operation r are either within the group I_r or within the groups that are connected to I_r by virtual links. For example, the internal node I_6 is connected to the internal nodes I_{25} and I_{36} via virtual links. All the relaxed twigs using the relaxation operation $Rel(e_{\$4,\$5})$ are within the groups represented by internal nodes I_6, I_{25} and I_{36}.

5.2 Assigning Internal Representatives in XTAH

The relaxation process for a query $\mathcal{Q}$= $\{\mathcal{T}, \mathcal{R}, \mathcal{C}, \mathcal{S}\}$ is essentially an interactive process of finding a relaxed query that is (next) closest to the original query and satisfies the relaxation controls $\mathcal{C}$ (i.e., the best relaxed query) until either the stop condition $\mathcal{S}$ is met or $\mathcal{Q}$ is no longer relaxable. To achieve this, a brute force approach is to check all the relaxed twigs on the leaf level, which is obviously inefficient. To remedy this condition, we propose to assign representatives to internal nodes, where a representative summarizes the distance characteristics of all the relaxed twigs covered by the internal node. Internal node representatives facilitate the searching for the best relaxed query by traversing an XTAH in a top-down fashion, where the path is determined by the distance properties of the representatives.

We propose to use M-tree [13] for assigning representatives to internal nodes in an XTAH. M-tree provides an efficient access method for similarity search in "metric space," i.e., where object similarities are defined by a distance function with non-negativity, symmetry and triangle properties. Given a tree organization of data objects where all the data objects are at the leaf level, M-tree assigns a data object covered by an internal node I to be the representative object of I. Each representative object stores the covering

[2]Due to space limit, we only show part of the XTAH here.

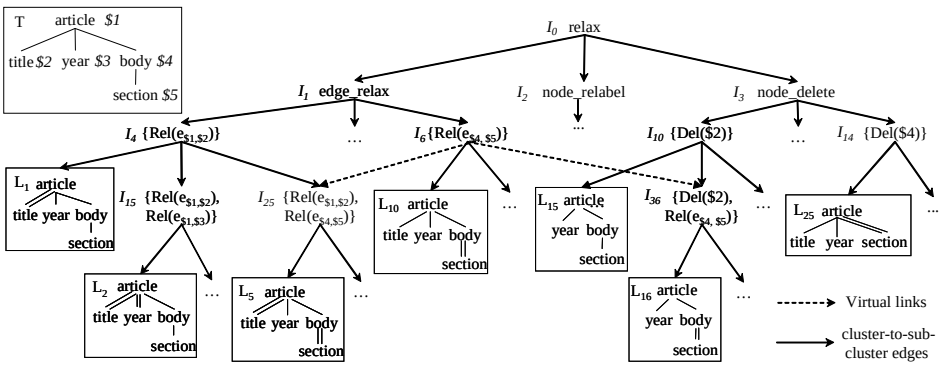


Figure 6: An example of XML relaxation index structure for the twig T

radius of the internal node, i.e., the maximal distance between the representative object and any data object covered by the internal node. These covering radii are then used in determining the path to a data object on the leaf level that is (next) closest to a query object during similarity searches.

Many policies are available in M-tree for promoting a data object on the leaf level to be an internal node's representative. In XTAH, given an internal node I, we select a relaxed twig represented by a leaf node within the subtree rooted at I to be the representative of I if the relaxed twig has the minimal covering radius. For an XTAH internal node I, we use O_I to denote its representative object and $cr(O_I)$ to represent the covering radius of the representative object. Let d be a distance function that satisfies the non-negativity, symmetry and triangle properties. Then the distance between a query $\mathcal{Q}$ and any relaxed twig L covered by an internal node I has the following upper and lower bounds:

$$d(\mathcal{Q}, O_I) - cr(O_I) \leq d(\mathcal{Q}, L) \leq d(\mathcal{Q}, O_I) + cr(O_I) \quad (1)$$

These distance bounds are useful in determining the searching paths. For example, given a query $\mathcal{Q}$ and two internal nodes I_i and I_j, suppose that the upper bound of the distance between $\mathcal{Q}$ and any twig covered by I_i is less than the lower bound of the distance between $\mathcal{Q}$ and any twig covered by I_j. In this case, the relaxed twig that is closest to the query cannot be within the group represented by the internal node I_j. Thus, internal node I_j can be pruned from the searching path. Due to the space limit, we do not include the detailed algorithm of utilizing covering radii and distance bounds in searching the (next) best relaxed query, which are similar to those presented in [13].

5.3 XTAH-Guided Query Relaxation Process

With the introductions of XTAH and its internal node representatives, we now discuss how to use an XTAH for a twig T to guide the relaxations of a query $\mathcal{Q}$ where $\mathcal{Q}.\mathcal{T}$ has the same structure as T.

Given a query $\mathcal{Q} = \{\mathcal{T}, \mathcal{R}, \mathcal{C}, \mathcal{S}\}$, an XTAH-guided query relaxation process consists of three steps:

- First, it updates the XTAH based on relaxation controls in the query $\mathcal{Q}$. More specifically, it prunes XTAH nodes from searching based on relaxation controls such as non-relaxable twig nodes (or edges), unacceptable twig node relabels, or disallowed relaxation types. For example, the relaxation controls in the sample query (Fig. 4) state that only *node deletion* and *edge generalization* may be used for deriving approximate answers. Thus, any XTAH node that is either within the group I_2, representing *node relabel*, or connected to I_2 by virtual links, will be disqualified from searching. Similarly, the internal nodes I_{14} and I_{14}, which represent groups of relaxed twigs using relaxation operations $Rel(\$4)$ and $Rel(e_{\$1,\$2})$ respectively, will be pruned from searching based on the relaxation controls. This step can be efficiently processed using XTAH internal node labels.

- After updating the XTAH, the process first repeatedly searches for the (next) best relaxed query based on relaxation constructs in the query until either the stop condition is met or all the relaxation constructs have been used. For example, the query in Fig. 4 contains two relaxation constructs: generalizing the edge between nodes *body* and *section* (i.e., $Rel(e_{\$4,\$5})$), deleting the node *year* (i.e., $Del(\$3)$). Thus, the process will select the (next) best relaxed query from the XTAH internal nodes that represent the two relaxation constructs respectively. This step can also be efficiently processed by using XTAH internal node labels as well as representatives.

- If further relaxation is needed, the process will then repeatedly searches for the next best relaxed query using relaxation constructs in addition to those specified in the query (i.e., $\mathcal{Q}.\mathcal{R}$). The process terminates when either the stop condition holds or the query reaches its relaxation limit. For example, for the query in Fig. 4, the process may search for relaxed queries using relaxation operations such as deleting node *section* or generalizing the edge between nodes *body* and *section*. This step can be efficiently processed by utilizing the distance information associated with XTAH internal node representatives.

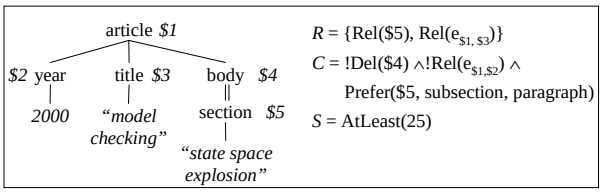


Figure 7: A query with its twig similar to that in Fig. 1

5.4 Relaxing Queries with Similar Structures

In this subsection, we discuss how to use an XTAH for a twig T to guide the relaxations of queries with structures "similar"to that of T.

We first define the canonical form of a twig, which is useful for describing twigs with similar structures. We assume that domain knowledge is available that predefines an unique label for each set of similar node labels.

Definition 6 ***The Canonical Form of A Twig*** *Given a twig T, the canonical form of the twig, denoted as T^c, can be obtained from T by: 1) replacing any ancestor-to-descendant edge ('//') in T to a parent-to-child edge ('/'); 2) changing the label of each node in T to its predefined unique label; and 3) removing the text of each node if any.*

Given a twig T, we use $E(T)$ to denote the set of twigs that have the same canonical forms as T. By Definition 6, the canonical form of a twig T has the most restricted structure constraints as compared to the twigs in $E(T)$. Thus, given a twig in $E(T)$, any of its relaxed twigs is also a relaxed twig of T^c. Therefore, an XTAH for T^c can be used to guide the relaxation of any query $\mathcal{Q}$ where $\mathcal{Q}.\mathcal{T} \in E(T)$.

Fig. 7 presents a sample query, which searches for articles published in *2000* with a title on "*model checking*" and a body section on "*space state explosion*." If there are insufficient number of exactly matched answers available, the *section* node may be relabeled and it is preferred to be relabeled to either *subsection* or *paragraph*. The canonical form of this twig is the same as the twig used in the sample XTAH (Fig. 6).[3] Therefore, the sample XTAH can be used to guide the relaxations of the query. Since the edge between nodes *body* (i.e., $4) and *section* (i.e., $5) is ancestor-to-descendant, the relaxation operation $Rel(e_{\$4,\$5})$ is inapplicable to the query. Therefore, we first prune the internal node I_6, whose label is $Rel(e_{\$4,\$5})$, as well as the internal nodes that are connected to I_6 via virtual links, i.e., I_{25} and I_{36}, from searching. All the twigs within these three internal nodes use the relaxation operation $Rel(e_{\$4,\$5})$, thus they cannot be used for relaxing the query. After pruning the three internal nodes, we follow the process presented in Section 5.3 to derive approximate answers for the query.

Similarly, given a twig T, an XTAH built for its canonical form T^c can also be used to guide the relaxations of twigs whose canonical forms are subtrees of T^c. The process is similar to the one presented in the example above.

[3]For simplicity, in this example, we assume that the unique label of each label is itself.

6 XML Ranking

Query relaxation often generates a list of approximate answers, which need to be ranked before being returned to users. A query contains both structure and content conditions. Thus, we shall rank an approximate answer based on its relevancies to both the structure and content conditions in the query. Many XML ranking models have been proposed(e.g., [14], [18]). Most, however, only measure the content similarity between an answer and a query and thus, they are inadequate for ranking approximate answers that use structure relaxations. Therefore, we need a similarity metric for evaluating the relevancy of an answer to the structure conditions in a query, i.e., *structure relevancy*. In this following, we first present how to measure structure relevancy and then discuss how to combine structure relevancy with content similarity.

6.1 Semantics-Oriented Structure Distance

We define the structure relevancy of an answer $\mathcal{A}$ to a query $\mathcal{Q}$ to be the structure similarity between the twig $\mathcal{Q}.\mathcal{T}$ and the least relaxed twig T', where the answer exactly matches the structure of T'. Due to the tree-structure of the twigs, the structure similarity between two twigs can be measured in a way similar to tree editing distance metrics(e.g., [26]). Thus, we measure the structure distance between an answer $\mathcal{A}$ and a query $\mathcal{Q}$, denoted as $struct\_dist(\mathcal{A}, \mathcal{Q})$, as the editing distance between the twig $\mathcal{Q}.\mathcal{T}$ and the least relaxed twig T', denoted as $d(\mathcal{Q}.\mathcal{T}, T')$, which is the total costs of the relaxation operations that relax $\mathcal{Q}.\mathcal{T}$ to T':

$$struct\_dist(\mathcal{A}, \mathcal{Q}) = d(\mathcal{Q}.\mathcal{T}, T') = \sum_{i=1}^{n} cost(r_i) \quad (2)$$

where the sequence of relaxation operations $r_1; ...; r_n$ transforms $\mathcal{Q}.\mathcal{T}$ into T'; and $cost(r_i)$ ($0 \leq cost(r_i) \leq 1$) is the cost of a relaxation operation r_i.

Existing edit distance algorithms do not consider the operation cost. Assigning equal cost to each operation is simple, but does not distinguish the semantics of different operations. Thus, we propose a semantics-oriented cost model for measuring the cost of each relaxation operation.

Before we introduce how we model the semantics of each relaxation operation, we shall first introduce how we model the semantics of XML data nodes. Given an XML document collection D, we represent the semantics of each data node v_i as a N vector $\{w_{i1}, w_{i2}, ..., w_{iN}\}$, where N is the total number of distinct terms in D and w_{ik} is the weight of the k^{th} term in the text of v_i. The weight of a term may be computed using Vector Space Model[21]. With this representation, the more similar the two vectors, the more semantically closer the two nodes. For example, the text of a *section* node has more overlap with that of a *paragraph* node than with the text of a *figure* node. Thus, a *section* node is semantically closer to a *paragraph* node than a *section* node to a *figure* node.

We now introduce the cost model for each relaxation operation with regard to a twig T as follows:

- Node Relabel $Rel(u, l)$

A node relabel operation changes the label of a node u in a twig from $u.label$ to a new label l. Intuitively, the more similar the new label is to the original label in semantics, the less the cost of the operation will be. The similarity between two labels, $u.label$ and l, denoted as $sim(u.label, l)$, can be measured as the cosine similarity between their corresponding vector representations in XML data. Thus, the cost of a relabel operation is:

$$cost(Rel(u, l)) = 1 - sim(u.label, l) \quad (3)$$

For example, in the INEX 05 data, the similarity between the vector representing *section* nodes and the vector representing *paragraph* nodes is 0.99, while the similarity between the vector for *section* nodes and the vector for *figure* nodes is 0.38. Thus, it is more expensive to relabel to the node *section* in Fig. 1 to *paragraph* than to *figure*.

- Node Deletion $Del(u)$

The more similar a node u that has a parent node v (i.e., v/u or v/u in the twig) to its parent node v in semantics, the less the cost of deleting the node u. For example, if we delete the *section* node from the twig in Fig. 1, we relax the query *searching for body sections on "frequent itemset"* to *searching for bodies on "frequent itemset."* The closer a *section* node in an article' *body* part to a *body* node in an article, the less the cost of deleting the *section* node. Let $V_{v/u}$ and V_v be the two vectors representing the union of the vectors of the nodes in XML data satisfying v/u and v respectively. The similarity between v/u and v, denoted as $sim(v/u, v)$, can be measured as the cosine similarity between the two vectors $V_{v/u}$ and V_v. Thus, we model the cost of a node deletion as:

$$cost(Del(u)) = 1 - sim(v/u, u) \quad (4)$$

For example, in the INEX 05 data, the similarity between the vector for *section* nodes inside *body* nodes and the vector for *body* nodes is 0.99, while the similarity between the vector for *keyword* nodes inside *article* nodes and the vector for *article* nodes is 0.2714. Thus, deleting the *keyword* node in Fig. 1 costs more than deleting the *section* node.

- Edge Generalization $Rel(e_{v,u})$

The closer a node u that has a parent node v (i.e., v/u) to a node u that has an ancestor node v (i.e., $v//u$) in semantics, the less the cost of the edge relaxation $Rel(e_{v,u})$. Let $V_{v/u}$ and $V_{v//u}$ be two vectors representing the union of the vectors of the nodes in XML data satisfying v/u and $v//u$ respectively. The similarity between v/u and $v//u$, denoted as $sim(v/u, v//u)$, can be measured as the cosine similarity between vectors $V_{v/u}$ and $V_{v//u}$. Thus, the cost for an edge generalization can be measured as:

$$cost(Rel(e_{v,u})) = 1 - sim(v/u, v//u) \quad (5)$$

For example, relaxing *article/title* in Fig. 1 to *article//title* makes the title of an article's author an approximate match for *article/title*. Since the similarity between an article's title and an author's title is low, the cost of generalizing *article/title* to *article//title* may be high.

6.2 XML Ranking Function

Given a query $\mathcal{Q}$, the relevancy of an answer $\mathcal{A}$ to the query $\mathcal{Q}$, denoted as $sim(\mathcal{A}, \mathcal{Q})$, is a function of two factors: the structure distance between the answer and the query, i.e., $struct\_dist(\mathcal{A}, \mathcal{Q})$, and the content similarity between the answer and the query, denoted as $cont\_sim(\mathcal{A}, \mathcal{Q})$. Intuitively, the larger the structure distance, the less the relevancy; the larger the content similarity, the more the relevancy. Thus, we combine the two factors in a way similar to the one used in XRank[15] for combining element rank with distance as follows:

$$sim(\mathcal{A}, \mathcal{Q}) = \alpha^{struct\_dist(\mathcal{A},\mathcal{Q})} * cont\_sim(\mathcal{A}, \mathcal{Q}) \quad (6)$$

where α is constant between 0 and 1. When the structure distance is zero, i.e., exact structure match, the relevancy of the answer $\mathcal{A}$ to the query $\mathcal{Q}$ is determined by the their content similarity. When the answer does not exactly match the query structure, the relevancy of the answer decreases as the structure distance increases. We use our extended vector space model for measuring content similarity, which has been proven effective in past research[18].

7 Experimental Evaluations

7.1 Datasets & Query Set

We use INEX 05 test collection for evaluating the quality of approximate query answering provided by our system. The document collections, around 500MByte in size, consists of over 12,000 scientific articles from IEEE Computer Society Journals. Each article contains an average of 1532 elements and each element has an average depth of 6.9.

We use the content-and-structure (CAS) queries in INEX 05 for our experimental studies. A CAS query is expressed in XPath with extensions of *about* functions, which are used to specify content conditions. The structure conditions in a query are further classified into two types: support and target. A support specifies where to search and a target suggests what to return. INEX 05 contains four sub-tasks for each CAS query based on whether to approximately or strictly match its target and support structure conditions in a query. Post-analysis of the relevance assessments in INEX 05 [24] concluded that there are in fact only two different interpretations of the structure conditions in a query: whether to strictly or approximately match the target structure condition. Thus, in our studies, we strictly match the support conditions and approximately match the target conditions, i.e., the so called VSCAS subtask in INEX 05.

The INEX 05 test collection contains a set of 17 official CAS queries and another set of 30 unofficial single-branch CAS queries. Since only 7 queries in the VSCAS subtask from the first set has relevance assessments, we use queries from both sets for our experimental studies.

<inex_topic topic_id="267" query_type="CAS" ct_no="113" >
<castitle>//article//fm//atl[about(., "digital libraries")]</castitle>
<description>Articles containing "digital libraries" in their title.</description>
<narrative>I'm interested in articles discussing Digital Libraries as their main subject. Therefore I require that the title of any relevant article mentions "digital library" explicitly. Documents that mention digital libraries only under the bibliography are not relevant, as well as documents that do not have the phrase "digital library" in their title.</narrative>
</inex_topic>

Figure 8: Topic 267 in INEX 05

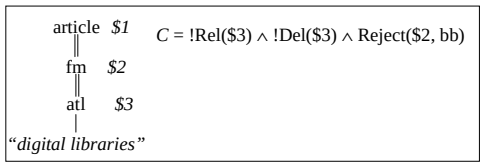


Figure 9: Representing topic 267 using our query relaxation language

7.2 Relevance Assessment

We use the relevance assessment in INEX 05 as the "gold standard" for evaluating the accuracy of approximate answers. The relevance assessment is obtained by asking each query author to judge an element's relevancy based on two aspects: 1) how much the element discusses the query; and 2) how much the element focuses on the query. This two-dimension relevancy value is then further combined into a single value between 0 and 1 using either a strict function, which discards partially relevant results, or a generalized function, which rewards partially relevant results. We use the generalized function in our experimental studies.

7.3 Test Runs

We run three sets of experiments. The first set compares the effectiveness of the semantics-oriented distance function with the uniform-cost distance function (i.e., assigning uniform cost 1 to each relaxation operation). The second set tests the effectiveness of relaxation language and the relaxation approach by comparing the results with relaxation controls with the results without using relaxation controls. The third set compares the effectiveness of our system with other systems participated in INEX 05.

We use single-branch queries in the first experiment set. We exclude queries with wildcards as target conditions, in which any element is an exact match, queries with non-relaxable structure conditions; and queries with no relevance assessment. 22 queries are used in the first experiment set. We use the function in Equation(6) to rank experimental results with both uniform-cost and semantics-oriented structure distances. We test our ranking function with the constant α varied from 0.1 to 0.9.

For the second set of experiments, we use queries in which users have explicit specifications regarding structure approximations. Only one query provides such specification, i.e., topic 267 as shown in Fig. 8. The topic consists of three parts: castitle, description and narrative. The information contained in the *narrative* part is the detailed description of a user's information needs and is used for judging result relevancy. The topic author considers an article's title, i.e., *atl*, non-relaxable and regards titles about "digital libraries" under the bibliography part, i.e., *bb*, irrelevant. Based on this narrative, we formulate the query using our relaxation language in Fig. 9. We ran this query with and without relaxation controls to evaluate the effectiveness of the relaxation controls in our language.

For the third set of experiments, we use the official multiple-branch queries, which have more relaxation variations. We exclude queries with no relevance assessments available in the VSCAS subtask. Only four multiple-branch queries have these properties (topic 256, 264, 275 and 284). We test our system using these four queries and compare our results with the best results from the official submissions in INEX 05 for the VSCAS subtask.

α	0.1	0.3	0.5	0.7	0.9
UCost	0.2584	0.2616	0.2828	0.2894	0.2916
SCost	0.3319	0.3190	0.3196	0.3068	0.2957
+%	28.44	21.94	13.04	6	4.08

Table 1: Comparisons of the evaluations for the results using semantics-oriented vs. uniform-cost distance functions for nxCG@10

Topic 256: //article[about(.//p, "data embedding")]//p[about(., watermarking)]
Topic 264: article[about(., "machine learning") and about(.//sec, "mutual information criterion")]
Topic 275: article[about(., abs, "data mining")]//sec[about(., "frequent itemsets")]
Topic 284: //article[about(.//bdy, thread implementation) and about(.//bdy, operating system)]

7.4 Evaluation Results

We use the official INEX 05 evaluation metrics to evaluate our experimental results: normalized extended cumulative gain (nxCG). nxCG is a user-oriented evaluation metric, similar to the precision/recall metric used in traditional IR. For a given rank i, the value of nxCG@i reflects the relative gain the user accumulated up to that rank, compared to the gain the user could have obtained if the system would have produced the optimum best ranking. For any rank, the normalized value of 1 represents the ideal performance. We use the INEX evaluation software, EvalJ[3], for computing the nxCG values for our experimental results.

Table 1 and Table 2 present the evaluation results for the first set of experiments using nxCG@10 and nxCG@25 respectively. The second and third rows in each table show the evaluation results using a uniform-cost and a semantics-oriented distance function respectively. The last row illustrates the performance improvements of the semantics-oriented distance function compared with the uniform-cost distance function. The results in these two tables ver-

α	0.1	0.3	0.5	0.7	0.9
UCost	0.2490	0.2490	0.2577	0.2616	0.2508
SCost	0.2880	0.2822	0.2749	0.2692	0.2608
+%	15.68	13.32	6.7	2.9	4.0

Table 2: Comparisons of the evaluations for the results using semantics-oriented vs. uniform-cost distance functions for nxCG@25

	With-relaxation-control	No-relaxation-control
nxCG@10	1.0	0.1013
nxCG@25	0.8986	0.2365

Table 3: Comparisons of the evaluation for the results with relaxation controls vs. without relaxation controls (α = 0.1)

topic	nxCG@10		nxCG@25		nxCG@50	
	Top-1	SCost	Top-1	SCost	Top-1	SCost
256	0.4293	0.4248	0.4733	0.5555	0.4693	0.4956
264	0.0	0.0069	0.0	0.0033	0.0739	0.0027
275	0.7715	0.638	0.589	0.5922	0.6369	0.5985
284	0.0	0.1259	0.0	0.1233	0.0	0.1233
average	0.3002	0.2989	0.2656	0.3186	0.2950	0.3050

Table 4: Comparisons of the evaluations for our results (α = 0.1) vs. the official INEX 05 top-1 results in the VSCAS subtask

ify that the semantics-oriented distance function outperforms the uniform-cost distance function. For example, the semantics-oriented distance function outperforms the uniform-cost function by 28.44% using nxCG@10 and 15.68% using nxCG@25 when α equals to 0.1.

We note that the performances of the semantics-oriented distance function increases when α is close to 0, while the performance of the uniform-cost distance function increases when α is close to 1. This is due to the differences in the value distributions of two distance functions. With the uniform-cost function, the distance between an approximate answer and a query is at least 1 because we need at least one operation to relax a query. With the semantics-oriented function, the distance between an approximate answer and a query are within the range of 0 and 1 if only one operation is used. (Most approximate answers for the queries in the first experiment set use only one relaxation operation.) The decay function $\alpha^{struct\_dist(\mathcal{A},\mathcal{Q})}$ (Fig. 10) differentiates values within 0 and 1 better when α is small. For example, given two approximate answers $\mathcal{A}_1$ and $\mathcal{A}_2$, suppose that $struct\_dist(\mathcal{A}_1, \mathcal{Q})$=0.05 and $struct\_dist(\mathcal{A}_2, \mathcal{Q})$ = 0.85. If α = 0.9, then there is very small difference between $\alpha^{0.05}$ (i.e., 0.99) and $\alpha^{0.85}$ (i.e., 0.91); while if α = 0.1, there are some difference between $\alpha^{0.05}$ (i.e., 0.89) and $\alpha^{0.05}$ (i.e., 0.14).

Table 3 presents the results for the second set of experiments using nxCG@10 and nxCG@25. The second column shows the evaluation results with relaxation controls and the third column presents the evaluation results without relaxation controls. We note that results with relaxation controls significantly outperform the results without relaxation controls. For example, the evaluation result with controls using nxCG@10 is 1, which is the perfect accuracy. The relaxation controls in the query inform the system to relax the query in a specific way, which in turn enables the system to provide results with more relevancy.

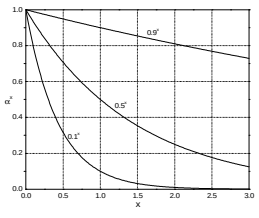


Figure 10: The decay function α^x

Table 4 presents the evaluation results for the third set of experiments using the semantics-oriented distance function and with α = 0.1. We present the evaluations of our results for each topic as well as the evaluations of the best results from the INEX 05 official submissions using nxCG@10, nxCG@25 and nxCG@50 respectively. We also include the corresponding average performance for the four queries in the last row. We observe that our results are comparable with the top-1 results at nxCG@10. Our results outperform the top-1 results by 20% at nxCG@25 and 3% at nxCG@50. The results reveal that the relaxation features in our system enables the system to retrieve approximate answers with more relevancy.

8 Related Work

A number of XML approximate search languages have been developed(e.g., [14], [25], [23], [11]). Many extend the standard query languages with constructs for specifying approximate search conditions. For example, [25] introduces *about* functions for users to specify approximate content conditions and [23] includes regular expressions for specifying approximate structure conditions. Our query language differs from existing languages in that our language allows user to both specify approximate search conditions and to control the approximate search process.

Searching XML data repositories is an active area of research ([11], [9], [16], [7], [20], [8], [6], [10], [19]). Most existing work on XML approximate query answering focus on efficient algorithms for deriving top-k answers based on the relaxation strategies. For example, [7] proposed a DAG structure to organize relaxed twigs and use a matrix to represent a query to speed up top-k processing. Our XTAH differs from the DAG structure in that XTAH clusters relaxed twigs into groups based on their relaxation operations and distances, while the DAG structure organizes relaxed twigs by the "superset" relationships among relaxed twigs. The DAG structure is efficient for deriving relaxed queries without user-specific relaxation specifications, while our XTAH is useful for relaxing queries with relaxation specifications.

There exists a large body of work on XML ranking([15], [18], [14], [11], [7]). Many focus on evaluating content similarity. JuruXML[11] uses a path similarity measure based on the lengths of a query path and an element path. Such measure does not account for path semantics. Recently, [7] proposed a family of structure scoring functions based on the occurrence frequencies of query structures in XML data. Our structure function, based on tree editing distances, differentiates twigs that use different operations even though the they have the same occurrence frequency in data, which are considered equally relevant in [7]. We also propose a function to combine structure distance and content similarity in determining the overall relevancy.

9 Conclusion

The heterogeneous nature of XML data model creates the need for approximate query answering. In this paper, we present a cooperative XML (CoXML) system for user-specific approximate query answering. We first propose a relaxation-enabled query language that uses twigs as the basic query model and extends the model with relaxation constructs and controls. Such extensions allow users to include their personalized relaxation specifications and to control the relaxation process. We then develop a relaxation index structure called XML Type Abstraction Hierarchy (XTAH) for systematic and scalable relaxations. XTAH clusters its relaxed twigs for frequently-used twigs into groups based on relaxation operations and distances, where twigs in the same group are closer to each other and use similar relaxation operations. Such an index structure greatly facilitates the searching of relaxed twigs for any given relaxation constructs and/or controls. Futher, we propose a semantics-oriented function for evaluating XML structure similarity. Finally, we use the INEX 05 test collection to evaluate our system. The evaluation results reveal that allowing users to specify relaxation constructs and controls in queries is a useful feature, which enables the system to provide more relevant answers. The results also demonstrate that using the semantics-oriented distance function yields results with better relevancy than using the uniform-cost distance function. Further, compared to other systems in INEX 05, our relaxation features enable our system to retrieve approximate answers with more relevancy.

10 Acknowledgement

The research and development of CoXML has been a team effort. We would like to acknowledge our CoXML members,Tony Lee, Eric Sung, Anna Putnam, Christian Cardenas, Joseph Chen and Ruzan Shahinian, for their contributions in implementation and testing efforts.

References

[1] BerkeleyDB. http://www.sleepycat.com/.

[2] DB2XML. www.ibm.com/software/data/db2/.

[3] EvalJ. http://evalj.sourceforge.net/.

[4] INEX. http://inex.is.informatik.uni-duisburg.de/.

[5] Tamino. http://www.softwareag.com/tamino.

[6] S. Amer-Yahia, S. Cho, and D. Srivastava. XML Tree Pattern Relaxation. In *EDBT*, 2002.

[7] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and Content Scoring for XML. In *VLDB*, 2005.

[8] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In *SIGMOD*, 2004.

[9] J. K. B. Sigurbjornsson and M. de Rijke. Processing Content-Oriented Xpath Queries. In *CIKM*, 2004.

[10] P. Buneman, B. Choi, W. Fan, R. Hutchison, R. Mann, and S. Viglas. Vectorizing and Querying Large XML Repositories. In *ICDE*, 2005.

[11] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML Documents via XML Fragments. In *SIGIR*, 2003.

[12] W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson. CoBase: A Scalable and Extensible Cooperative Information System. *J. Intell. Inform. Syst.*, 6(11), 1996.

[13] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*, 1997.

[14] N. Fuhr and K. Groβjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *SIGIR*, 2001.

[15] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search Over XML Document. In *SIGMOD*, 2003.

[16] Y. Kanza and Y. Sagiv. Flexible Queries Over Semistructured Data. In *PODS*, 2001.

[17] S. Liu and W. W. Chu. RLXQuery: A Relaxation-enabled XML Query Language. In *UCLA Computer Science Department Technical Report*, 2006.

[18] S. Liu, Q. Zou, and W. Chu. Configurable Indexing and Ranking for XML Information Retrieval. In *SIGIR*, 2004.

[19] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries on Heterogeneous Data Sources. In *VLDB*, 2001.

[20] A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive Processing of Top-k Queries in XML. In *ICDE*, 2005.

[21] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[22] T. Schlieder. Schema-Driven Evaluations of Approximate Tree Pattern Queries. In *EDBT*, 2002.

[23] A. Theobald and G. Weikum. Adding Relevance to XML. In *WebDB*, 2000.

[24] A. Trotman and M. Lalmas. The Interpretation of CAS. In *INEX 05 Workshop*.

[25] A. Trotman and B. Sigurbjornsson. Narrowed Extended XPath I NEXI. In *INEX 04 Workshop*, 2004.

[26] K. Zhang and D. Shasha. Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *SIAM J. Comput.*, 18(6):1245– 1262, 1989.