CoXML: A Cooperative XML Query Answering System

Shaorong Liu[1] and Wesley W. Chu[2]

[1] IBM Silicon Valley Lab, San Jose, CA, 95141, USA
shaorongliu@gmail.com
[2] UCLA Computer Science Department, Los Angeles, CA, 90095, USA
wwc@cs.ucla.edu

Abstract. The heterogeneity nature of XML data creates the need for approximate query answering. In this paper, we present an XML system that cooperates with users to provide user-specific approximate query answering. The key features of the system include: 1) a query language that allows users to specify approximate conditions and relaxation controls; 2) a relaxation index structure, XTAH, that enables the system to provide user-desired relaxations as specified in the queries; and 3) a ranking model that incorporates both content and structure similarities in evaluating the relevancy of approximate answers. We evaluate our system with the INEX 05 test collections. The results reveal the expressiveness of the language, show XTAH's capability in providing user-desired relaxation control and demonstrate the effectiveness of the ranking model.

1 Introduction

The growing use of XML in scientific data repositories, digital libraries and Web applications has increased the need for flexible and effective XML search methods. There are two types of queries for searching XML data: content-only (CO) queries and content-and-structure (CAS) queries. CAS queries are more expressive and thus yield more accurate searches than CO queries. XML structures, however, are usually heterogeneous due to the flexible nature of its data model. It is often difficult and unrealistic for users to completely grasp the structural properties of data and specify exact query structure constraints. Thus, XML approximate query answering is desired, which can be achieved by relaxing query conditions.

Query relaxation is often user-specific. For a given query, different users may have different specifications about which conditions to relax and how to relax them. Most existing approaches on XML query relaxation (e.g., [1]) do not provide control during relaxation, which may yield undesired approximate answers. To provide user-specific approximate query answering, it is essential for an XML system to have a relaxation language that allows users to specify their relaxation control requirements and to have the capability to control the query relaxation process.

Furthermore, query relaxation returns a set of approximate answers. These answers should be ranked based on their relevancy to both the structure and content conditions of the posed query. Many existing ranking models (e.g., [2], [3]) only measure the content similarities between queries and answers, and thus are inadequate for ranking approximate answers that use structure relaxations. Recently, [4] proposed a family of structure

scoring functions based on the occurrence frequencies of query structures among data without considering data semantics. Clearly, using the rich semantics provided in XML data in design scoring functions can improve ranking accuracy.

To remedy these shortcomings, we propose a new paradigm for XML approximate query answering that places users and their demands in the center of design approach. Based on this paradigm, we develop a cooperative XML system that provides user-specific approximate query answering. More specifically:

- First, we develop a relaxation language that allows users to specify approximate conditions and control requirements in queries (e.g., preferred or unacceptable relaxations, non-relaxable conditions and relaxation orders). (Section 3)
- Second, we introduce a relaxation index structure that clusters twigs (as introduced in Sec 2.1) into multi-level groups based on relaxation types and distances. By such clustering, the index structure enables a systematic control of the relaxation processing based on users' specifications in queries. (Section 4)
- Third, we propose a semantic-based tree editing distance to evaluate XML structure similarities based on not only the number of relaxations but also relaxation semantics. We also develop a model that combines both structure and content similarities in evaluating the overall relevancy [5].
- Finally, our experimental studies using the INEX 05 benchmark test collection[3] demonstrate the effectiveness of our proposed methodology.

2 XML Query Relaxation Background

2.1 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 tree-like structures. Thus, we use twig as the basic query model. Fig. 1(a) 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*." Each twig node is associated with an unique *id*, shown in italic beside the node. The IDs are not needed when all the node labels are distinct. The text under a node is the content constraint on the node.

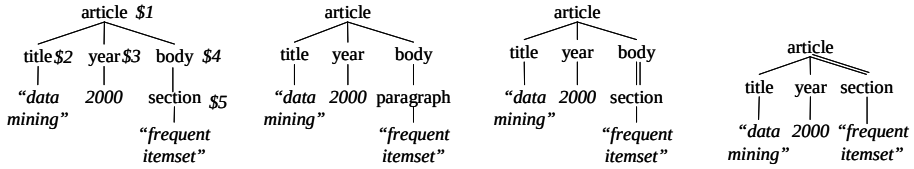
For a twig T, we use $T.V$ and $T.E$ to represent its nodes and edges respectively. For a twig node v ($v \in T.V$), we use $v.label$ to denote the node label. We use $e_{u,v}$ to denote the edge from nodes u to v, either parent-to-child ("/") or ancestor-to-descendant ("//").

2.2 Query Relaxation

In the XML model, there are two types of query relaxations: value relaxation and structure relaxation. Value relaxation, successfully used in relational models (e.g., [6]), is orthogonal to structure relaxation. In this paper, we focus on structure relaxation. Many structure relaxation types have been proposed ([7], [8], [1]). We use the following three structure relaxation types, similar to the ones in [1], which capture most of the relaxation types proposed in previous work.

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

– **Node Relabel** A node can be relabeled to similar or equivalent labels according to domain knowledge. For example, the twig in Fig. 1(a) can be relaxed to that in Fig. 1(b) by relabeling node *section* to *paragraph*.
– **Edge Generalization** A parent-to-child edge can be generalized to an ancestor-to-descendant edge. For example, the twig in Fig. 1(a) can be relaxed to that in Fig. 1(c) by relaxing *body/section* to *body//section*.
– **Node Deletion** A node v may be deleted to relax the structure constraint. If v is an internal node, then the children of v will be connected to the parent of v with ancestor-to-descendant edges. For instance, the twig in Fig. 1(a) can be relaxed to that in Fig. 1(d) by deleting the internal node *body*. We assume that the root node of a twig cannot be deleted since it represents the search context.



(a) A sample twig (b) Node relabel (c) Edge generalization (d) Node deletion

Fig. 1. A sample twig and its relaxed twigs

Given a twig T, a *relaxed twig* can be generated by applying one or more relaxation operations to T. Let m be the number of relaxation operations applicable to T, then there are at most $\binom{m}{1}$ + ... + $\binom{m}{m}$ = 2^m relaxation combinations, i.e., 2^m relaxed twigs.

3 XML Query Relaxation Language

A number of XML approximate search languages have been proposed. Most extend standard query languages with constructs for approximate text search (e.g., XIRQL [3], TeXQuery [9]). XXL [10] provides users with constructs for users to specify both approximate structure and content conditions, which however, does not allow users to control the relaxation process. Users may often want to specify their preferred or rejected relaxations, non-relaxable query conditions, or to control the relaxation orders among multiple relaxable conditions.

To remedy this shortcoming, we propose an XML relaxation language that allows users to both specify approximate conditions and to control the relaxation process. A relaxation-enabled query $\mathcal{Q}$ is a tuple $(\mathcal{T}, \mathcal{R}, \mathcal{C}, \mathcal{S})$, where:

– $\mathcal{T}$ is a twig as described as Section 2.1;
– $\mathcal{R}$ is a set of relaxation constructs specifying which conditions in $\mathcal{T}$ may be approximated when needed;
– $\mathcal{C}$ is a boolean combination of 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 are: we first search for answers exactly matching the twig; we then test the stop condition to check whether relaxation is needed. If not, we repeatedly relax the twig based on the relaxation constructs and control until either the stop condition is met or the twig cannot be further relaxed.

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, control and stop condition respectively. Note that a twig is required to specify a query, while relaxation constructs, control and stop condition are optional. When only a twig is present, we iteratively relax the query based on similarity metrics until the query cannot be further relaxed.

A relaxation construct for a query $\mathcal{Q}$ is either a specific or a generic relaxation operation in any of the following forms:

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

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

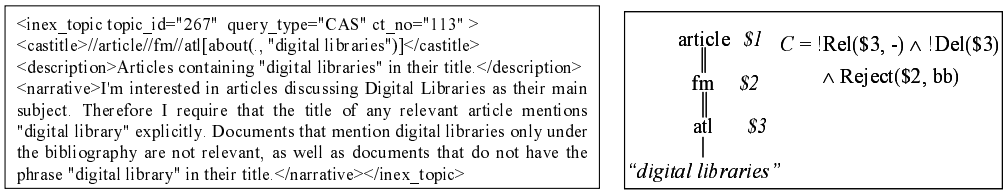
- Non-relaxable condition $!r$, where $r \in \{rel(u, -),\ del(u),\ gen(e_{u,v}) \mid u,\ v \in \mathcal{Q}.\mathcal{T}.V,\ e_{u,v} \in \mathcal{Q}.\mathcal{T}.E\}$, specifies that node u cannot be relabeled or deleted, or edge $e_{u,v}$ cannot be generalized;
- $Prefer(u, l_1, ..., l_n)$, where $u \in \mathcal{Q}.\mathcal{T}.V$ and l_i is a label ($1 \leq$ i $\leq$ n), specifies that node u is preferred to be relabeled to the labels in the order of (l_1, ..., l_n);
- $Reject(u, l_1, ..., l_n)$, where $u \in \mathcal{Q}.\mathcal{T}.V$, specifies a set of unacceptable labels for node u;
- $RelaxOrder(r_1, ..., r_n)$, where $r_i \in \mathcal{Q}.\mathcal{R}$ ($1 \leq$ i $\leq$ n), specifies the relaxation orders for the constructs in R to be (r_1, ..., r_n);
- $UseRType(rt_1, ..., rt_k)$, where $rt_i \in \{$*node_relabel*, *node_delete*, *edge_generalize*$\}$ ($1 \leq$ 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
- $d(\mathcal{Q}.\mathcal{T}, T') \leq \tau$, 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.

We now present an example of using our relaxation language to express INEX 05 topic 267 (Fig. 2(a)). The topic consists of three parts: *castitle* (i.e., the query formulated in an XPath-like syntax), *description* and *narrative*. The *narrative* part contains 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 this topic using our relaxation language as in Fig. 2(b).

We have developed a GUI interface for users to specify relaxations. Users may first input the twig using an XPath-like syntax. Based on the input twig, the interface automatically generates a set of relaxation candidates. Users can then specify relaxation constructs and controls by selecting relaxations from the candidate set.



(a) (b)

Fig. 2. Topic 267 in INEX 05 (a) & specifying the topic with our relaxation language (b).

4 XML Relaxation Index

4.1 XML Type Abstraction Hierarchy - XTAH

Several approaches for relaxing XML or graph queries have been proposed ([7], [4], [11], [1], [12]). Most focus on efficient algorithms for deriving top-k approximate answers without relaxation control. To remedy this condition, we propose an XML relaxation index structure, XML Type Abstraction Hierarchy (XTAH), that clusters relaxed twigs into multi-level groups based on relaxation types and distances. Each group consists of twigs using similar types of relaxations. Thus, XTAH enables systematic relaxation control based on users' specifications. For example, *Reject* can be implemented by pruning groups of twigs with unacceptable relaxations. *RelaxOrder* can be implemented by selecting the relaxed twigs from groups based on the specified order.

An XTAH for a twig structure T, denoted as XT_T, is a hierarchical cluster that represents relaxed twigs of T at different levels of relaxations based on the types of operations used by the twigs and the distances between them. 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 that use similar operations and are closer to each other by a given distance metric. The label of an internal node is the common relaxation operations (or types) used by the twigs in the cluster. The higher level an internal node in the XTAH, the more general the label of the node, the less relaxed the twigs in the internal node.

Fig. 3 shows an XTAH for the sample twig in Fig. 1(a).[4] For ease of reference, 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 *T'*.

Given a relaxation operation r, let I_r be an internal node with a label $\{r\}$. That is, I_r represents a cluster 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, while the 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 I_r. For example, the relaxed twig T'_2, which uses two operations $gen(e_{\$1,\$2})$ and $gen(e_{\$4,\$5})$, is not included in the internal node that represents$\{gen(e_{\$4,\$5})\}$, I_7. This is because T'_2 may belong to either group I_4 or group I_7 but is closer to the twigs in group I_4.

To support efficient searching or pruning of relaxed twigs in an XTAH that use an operation r, we add a virtual link from internal node I_r to internal node I_k where I_k is

[4] Due to space limitations, we only show part of the XTAH here.

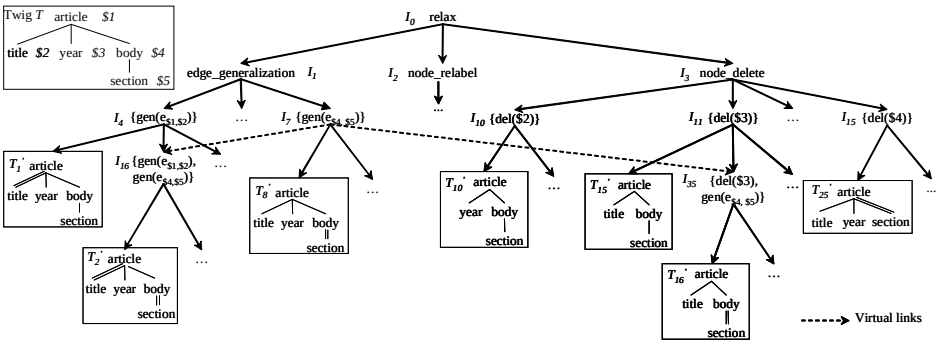


Fig. 3. An example of XML relaxation index structure for the twig T

not a descendant of I_r but all the twigs within I_k use operation r. By doing so, relaxed twigs that use operation r are either within group I_r or within the groups connected to I_r by virtual links. For example, internal node I_7 is connected to internal nodes I_{16} and I_{35} via virtual links. Thus, all the relaxed twigs using the operation $gen(e_{\$4,\$5})$ are within the groups I_7, I_{16} and I_{35}.

XTAH provides several significant advantages: 1) we can efficiently relax a query based on relaxation constructs by fetching relaxed twigs from internal nodes whose labels satisfy the constructs; 2) we can relax a query at different granularities by traversing up and down an XTAH; and 3) we can control and schedule query relaxation based on users' relaxation control requirements. For example, relaxation control such as non-relaxable conditions, *Reject* or *UseRType* can be implemented by pruning XTAH internal nodes corresponding to unacceptable operations or types.

Due to space limitations, the algorithm for building XTAH is not presented here. Interested readers should refer to [5] for details.

4.2 XTAH-Guided Query Relaxation Process

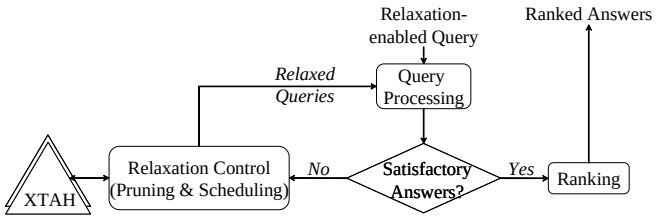


Fig. 4. Query relaxation control flow

Fig. 4 presents the control flow of a relaxation process based on XTAH and relaxation specifications in a query. The *Relaxation Control* module prunes irrelevant XTAH groups corresponding to unacceptable relaxation operations or types and schedules relaxation operations such as *Prefer* and *RelaxOrder*, as specified in the query.

More specifically, the process first searches for exactly matched answers. If there are enough number of answers available, there is no need for relaxation and the answers are returned. Otherwise, based on the relaxation control, the algorithm prunes XTAH internal nodes that correspond to unacceptable operations such as non-relaxable twig nodes (or edges), unacceptable node relabels and rejected relaxation types. This step can be efficiently carried out by using internal node labels and virtual links. After pruning disqualified internal groups, based on relaxation constructs and control such as *RelaxOrder* and *Prefer*, the *Relaxation Control* module schedules and searches for the relaxed query that best satisfies users' specifications from the XTAH. This step terminates when either the stop condition is met or all the constructs have been processed. If further relaxation is needed, the process then iteratively searches for the relaxed query that is closest to the original query by distance, which may use relaxation operations in addition to those specified in the query. This process terminates when either the stop condition holds or the query cannot be further relaxed. Finally, the process outputs approximate answers.

5 Experimental Evaluations

5.1 Experiment Setup

We have implemented the CoXML system in Java, which consists of a relaxation language parser, an XTAH builder, a relaxation controller and a ranking module. The ranking model evaluates the relevancy of an answer $\mathcal{A}$ to a query $\mathcal{Q}$, denoted as $sim(\mathcal{A}, \mathcal{Q})$, based on two factors: the structure distance between $\mathcal{A}$ and $\mathcal{Q}$, $struct\_dist(\mathcal{A}, \mathcal{Q})$, and the content similarity between $\mathcal{A}$ and $\mathcal{Q}$, denoted as $cont\_sim(\mathcal{A}, \mathcal{Q})$, as shown in (1).

$$sim(\mathcal{A}, \mathcal{Q}) = \alpha^{struct\_dist(\mathcal{A},\mathcal{Q})} * cont\_sim(\mathcal{A}, \mathcal{Q}) \quad (1)$$

where α is a constant between 0 and 1; $cont\_sim(\mathcal{A}, \mathcal{Q})$ is an extended vector space model for evaluating XML content similarity [2]; and $struct\_dist(\mathcal{A}, \mathcal{Q})$ is a tree editing distance metric that evaluates relaxation cost based on its semantics [5].

We use INEX 05 document collection, content-and-structure queries and relevance assessment ("gold standard") to study the effectiveness of approximate answers returned by our system. We use the INEX 05 evaluation metric to evaluate experimental results: normalized extended cumulative gain (nxCG). For a given rank i, the value of nxCG@i reflects the relative gain an 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 i, the ideal nxCG@i performance is 1.

5.2 Experimental Results

The first experiment evaluates the effectiveness of our semantic-based tree editing distance for evaluating structure similarity. We used the 22 single-branch content-and-structure queries in INEX 05 for the experiment. Table 1 presents the nxCG@10 evaluation results (averaged over the 22 queries) with the semantics-based tree editing distance as compared to that with uniform-cost tree editing distance. The results validate that differentiating the operation cost improves relaxation performance.

Cost Model \ α	0.1	0.3	0.5	0.7	0.9
Uniform	0.2584	0.2616	0.2828	0.2894	0.2916
Semantic	0.3319	0.3190	0.3196	0.3068	0.2957

Table 1. The nxCG@10 evaluations of the first experiment results with semantic vs. uniform tree editing distance.

Control? \ Metric	nxCG@10	nxCG@25
Yes	1.0	0.8986
No	0.1013	0.2365

Table 2. The evaluations of the second experiment results with vs. w/o relaxation controls (α = 0.1).

The second experiment tests the effectiveness of relaxation control by comparing the results with relaxation control against the results without relaxation control for topic 267 in INEX 05 (Fig. 2). The evaluation result in Table 2 demonstrates that relaxation specifications enable the system to control the relaxation process and thus yield results with greater relevancy.

6 Conclusion

In this paper, we have developed an XML system that cooperates with users to provide user-specific approximate query answering. More specifically, we first introduce a relaxation language that allows users to specify approximate conditions and relaxation control requirements in a posed query. We then propose a relaxation index structure, XTAH, that clusters relaxed twigs into multi-level groups based on relaxation types and their inter-distances. XTAH enables the system to provide user-desired relaxation control as specified in the query. Our experimental studies with INEX 05 test collection reveal the expressiveness of the relaxation language and the effectiveness of using XTAH for providing user-desired relaxation control.

References

1. S. Amer-Yahia, S. Cho, and D. Srivastava. XML Tree Pattern Relaxation. In *EDBT*, 2002.
2. S. Liu, Q. Zou, and W.W. Chu. Configurable Indexing and Ranking for XML Information Retrieval. In *SIGIR*, 2004.
3. N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *SIGIR*, 2001.
4. S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and Content Scoring for XML. In *VLDB*, 2005.
5. W. W. Chu and S. Liu. CoXML: A Cooperative XML Query Answering System. In *The Encyclopedia of Computer Science and Engineering, Edit by B. Wah*. John Wiley & Sons, Inc, 2007.
6. W.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.
7. Y. Kanza and Y. Sagiv. Flexible Queries Over Semistructured Data. In *PODS*, 2001.
8. T. Schlieder. Schema-Driven Evaluations of Approximate Tree Pattern Queries. In *EDBT*, 2002.
9. S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. TeXQuery: A Full-Text Search Extension to XQuery. In *WWW*, 2004.
10. A. Theobald and G. Weikum. Adding Relevance to XML. In *WebDB*, 2000.
11. A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive Processing of Top-k Queries in XML. In *ICDE*, 2005.
12. I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries on Heterogeneous Data Sources. In *VLDB*, 2001.