

Flexible Querying of XML Documents

Krishnaprasad Thirunarayan and Trivikram Immaneni

Department of Computer Science and Engineering
Wright State University, Dayton, OH 45435, USA.
t.k.prasad@wright.edu, immaneni.2@wright.edu

Abstract. Text search engines are inadequate for indexing and searching XML documents because they ignore metadata and aggregation structure implicit in the XML documents. On the other hand, the query languages supported by specialized XML search engines are very complex. In this paper, we present a simple yet flexible query language, and develop its semantics to enable intuitively appealing *extraction* of relevant fragments of information while simultaneously falling back on *retrieval* through plain text search if necessary. We also present a simple yet robust relevance ranking for heterogeneous document-centric XML.

1 Introduction and Motivation

Popular search engines (such as Google, Yahoo!, MSN Search, etc) index document text and retrieve documents efficiently in response to easy to write queries. Unfortunately, these search engines suffer from at least two drawbacks: (a) They ignore information available in the metadata / annotations / XML tags¹. (b) They are oblivious to the underlying aggregation structure implicit in the tree-view of XML documents. On the other hand, specialized search engines for querying XML documents require some sophistication on the part of the user to formulate queries.

In this paper, we present a query language that is simple to use and yields results that are more meaningful than the corresponding text search would yield on an XML document. Specifically, it facilitates exploitation of metadata to *extract* relevant fragments of information without sacrificing the ability to fall back on *retrieval* through plain text search.

Example 1. Consider the following fragment² of SIGMOD record [2].

```
- <article>
  <title>A Note on Decompositions of Relational Databases.</title>
  - <authors>
    <author position="00">Catriel Beeri</author>
    <author position="01">Moshe Y. Vardi</author>      </authors>    </article>
```

¹ Actually, search engines do analyze the content associated with the META-element, the TITLE-element, etc, and factor in information implicit in the text fonts and anchor text (link analysis), for relevance ranking an HTML document [1].

² The word "Database" in the second record appears as "Data Base" in the original dataset.

```

- <article>
<title>Implementation of a Time Expert in a Data Base System.</title>
- <authors>
  <author position="00">Ricky Overmyer</author>
  <author position="01">Michael Stonebraker</author> </authors> </article>

```

Searching for “articles by Stonebraker” should retrieve the above page, and better yet, cull out information about the title, co-authors, etc, of Stonebraker’s articles while ignoring articles by others. Similarly, searching for “articles with database” in their title should ideally yield all the above articles.

Example 2. Consider the following fragment of Mondial database [2]. Observe that a lot of factual data is captured via attribute bindings.

```

<mondial>
<continent id="f0_119" name="Europe" /> ...
<country id="f0_149" name="Austria" capital="f0_1467" population="8023244"
  datacode="AU" total_area="83850" population_growth="0.41"
  infant_mortality="6.2" ... government="federal republic" ...>
  <name>Austria</name> ...
- <province id="f0_17447" name="Vienna" ...>
  - <city id="f0_1467" country="f0_149" province="f0_17447" ...>
    <name>Vienna</name> <population year="94">1583000</population> </city>
  </province> ...
<languages percentage="100">German</languages>
<encompassed continent="f0_119" percentage="100" /> ...
<border length="784" country="f0_220" /> ...
</country> ...
</mondial>

```

Searching for “Austria” should fetch the above record from the database from which further geographic information about its cities and provinces can be determined. The internal name/code for Austria in the database can be used to infer additional information related to bordering countries, border lengths, etc.

Example 3. Consider the following heterogeneous XML fragment [4].

```

<document>
<article id="1">
  <author><name>Adam Dingle</name></author>
  <author><name>Peter Sturmh</name></author>
  <author><name>Li Zhang</name></author>
  <title>Analysis and Characterization of Large-Scale Web Server Access
    Patterns and Performance</title>
  <year>1999</year>
  <booktitle>World Wide Web Journal</booktitle> </article>
<article id="2" year="1999">
  <author name="A. Dingle" ></author>
  <author name="E. Levy" ></author>
  <author name="J. Song" ></author>
  <author name="D. Dias" ></author>
  <title>Design and Performance of a Web Server Accelerator</title>
  <booktitle> Proceedings of IEEE INFOCOM </booktitle> </article>
<article id="3">
  @inproceedings{IMN97,
    author="Adam Dingle and Ed MacNair and Thao Nguyen",
    title="An Analysis of Web Server Performance",
    booktitle="Proceedings of the IEEE Global Telecommunications
      Conference (GLOBECOM)",
    year=1999} </article>
</document>

```

It contains information about articles expressed in three different ways. A robust search strategy should ideally deliver all the three records when articles by “Dingle” are sought.

Example 4. Similar issues arise in the context of XML representation of binary relationships and XML serialization of RDF model, as illustrated below [15]. Specifically, “same” information may be expressed differently based on the preferences and views of the authors of web documents.

```
<course name="Discrete Mathematics">
  <lecturer>David Billington</lecturer> </course>
<lecturer name="David Billington">
  <teaches>Discrete Mathematics</teaches> </lecturer>
<teachingOffering>
  <lecturer>David Billington</lecturer>
  <course>Discrete Mathematics</course> </teachingOffering>
```

Our work builds on and extends the seminal work of Cohen et al [3] on XSearch, a search engine for XML documents. XSearch’s query language embodies the simplicity of Google-like search interface (easing the task of query formulation) while exploiting the hierarchical structure of nested XML-elements to deliver precise and coherent results (that is, containing semantically related pieces of information). However, their query language ignores XML-attributes entirely. In this paper, we smoothly extend XSearch’s query language to accommodate XML-attributes and their string values with the following benefits:

- XML documents with attributes can now be queried. Observe that, in the document centric applications of XML, information bearing strings are in text nodes, while in the data centric applications of XML, information bearing strings are associated with attributes.
- In spite of having general rules of thumb about when to use XML-elements and when to use XML-attributes for expressing a piece of information [14], it is still quite common to see authoring variations that mix the two. For instance, one can find the following “equivalent” pattern in use: `<T A="s"/>` and `<T> <A> s </T>`. Accommodating this variation can improve query *recall*.
- Semantic Web formalisms such as RDF, OWL, etc [16, 15, 17] build on XML and make extensive use of attributes. In fact, the above two forms are equivalent in RDF. So a simple XML Search Engine that can deal with attributes will be a welcome addition to the toolset till customized search engines for RDF and OWL become commonplace.

Furthermore, in many applications, XML documents may get progressively refined via a sequence of annotaters. For instance, in an initial step, an entire name in a text may be recognized and enclosed within `<name> ... </name>` tags, while in a subsequent step, it may be refined by delimiting the first name and the last name using `<firstName> ... </firstName>` and `<lastName> ... </lastName>` tags respectively.

In Section 2, we present selected related works. In Section 3, we discuss the proposed XML query language, develop its semantics, and illustrate it through

examples. In Section 4, we conclude with suggestions for future work. (Figures, other related works and implementation details have been skipped due to space limitations.)

2 Selected Related Work

Florescu et al [4] present an extension of the XML-QL language that supports querying of XML documents based on its structure and its textual content, facilitating search of XML documents whose structure is only partially known, or searching heterogeneous XML document collections. The implementation uses an off-the-shelf relational database system. In XIRCL, Fuhr and Grojohann [5] incorporate different answer granularity, robust query matching, and IR-related features such as relevance ranking using probabilistic models. Meyer et al [6] describe a search engine architecture and implementation based on XIRCL. Carmel et al [7] use XML fragments as queries instead of inventing a new XML query language, and extend the traditional vector space model for XML collections. The weight associated with an individual term depends on its context, and the ranking mechanism is used to deal with imperfect matches and high recall. In comparison to all these approaches, our query language is less expressive, the weighting mechanism and ranking is simpler, given that the query answer has already culled out the relevant answer. Schlieder and Meuss [8] reduce XML querying to tree matching by simulating attributes via elements and strings via word-labelled node sequences, and adapt traditional IR techniques for document ranking. Our approach resembles this work in spirit, but again the query language is simpler. For example, queries involving an element name and a keyword has different interpretation which is robust with respect to the level of annotations. In contrast with approaches so far, Theobald and Weikum [9] focus on heterogeneous documents, path-based queries and semantic-similarity search conditions. Grabs and Schek [10] and Guo et al [13] try to capture the intuition that the content that is more distant in a document tree is less important than the one that is close to the context node while determining term weights and relevance. On the other hand, we want to preserve the semantic impact of a piece of text irrespective of the level of annotations surrounding it. Li et al [11] propose an extension to XQuery that marries the precision of XQuery with the convenience of a keyword-based XML query language. Catania et al [12] provide a nice review of the indexing schemes employed for querying XML documents.

3 Query Language

The standard search engine query is a list of optionally signed keywords. For querying XML documents, Cohen et al [3] allow users to specify labels and keyword-label combinations that must or may appear in a satisfying XML document fragment. Our queries allow keywords, element names, and attribute names, optionally with a plus (“+”) sign. Intuitively, element/attribute names

relate to type information/metadata, while keywords relate to concrete values/data.

3.1 Query Syntax

Formally, a *search term* has one of the following forms: $e:a:k$, $e:a:$, $:a:k$, $e::k$, $e::$, $:a:$, $::k$, $l:k$, $l:$ and $:k$, where e is an element name, a is an attribute name, l is an element/attribute name, and k is a keyword (string). Furthermore, $l:k$ is interpreted as $l::k$ or $:l:k$, $l:$ is interpreted as $l::$ or $:l:$, and $:k$ is interpreted as $::k$. A *query* is a sequence of optionally signed search terms. Signed search terms are *required* to be present in the retrieved results, while unsigned search terms are *desirable* in the retrieved results.

3.2 XML Datamodel

Conceptually, an XML document is modelled as an ordered tree [17]. For our purposes, XML tree contains the following types of nodes³:

- *Root node*: The root node is the root of the tree. The element node for the document element is a child of the root node.
- *Element node*: There is an element node for every element in the document. The children of an element node are the element nodes and the text nodes (for its content).
- *Text node*: Character data is grouped into text nodes.
- *Attribute node*: An element node can have an associated set of attribute nodes; the element is the parent of each of these attribute nodes; however, an attribute node is not technically a child of its parent element. Each attribute node has a string-value.

3.3 Single Search Term Satisfaction

We specify when a search term is *satisfied* by an XML subtree⁴. In contrast to Cohen’s work, our approach abstracts from differences in the representation of a piece of information either as an attribute-value pair of an element or as the element’s subelement enclosing the value text (for example, `<T A="s"/>` and `<T <A> s </T>`), among other things. Additionally, it addresses the situation where the text of an XML document can be further refined through annotation.

- The search term $e:a:k$ is satisfied by a tree containing a subtree with the top element e that is associated with the attribute a with value containing k , or a subelement a with descendant text node containing k .

³ We ignore *namespace* nodes, *processing instruction* nodes, and *comment* nodes, and the ability to create additional internal *links* between tree nodes.

⁴ In this paper, a subtree is always rooted at an element node.

- The search term $e:a$: is satisfied by a tree containing a subtree with the top element e that is associated with the attribute a , or subelement a .
- The search term $:a:k$ is satisfied by a tree containing a subtree with a top element that is associated with the attribute a with value containing k , or a subelement a with descendant text node containing k .
- The search term $e:k$ is satisfied by a tree containing a subtree with the top element e and that has
 - an attribute associated with the value containing k , or
 - a descendant element with an associated attribute value containing k , or
 - a descendant text node containing k .
- The search term $e::$ is satisfied by a tree containing a subtree with the top element name e .
- The search term $:a$: is satisfied by a tree containing a subtree with a top element that is associated with the attribute a or a subelement a .
- The search term $::k$ is satisfied by a tree containing
 - a subtree with a top element that is associated with an attribute value containing k , or
 - the descendant text node containing k .

Observe that a query can use detailed knowledge of XML document structure (for example, via $e:a:k$ etc), or have the flexibility to express textual search (for example, via $:k$, etc). For the purposes of the above definition, a tree contains itself.

Example 1 (cont.) The search term $author::$ is satisfied by

```
<author position="00">Catriel Beerl</author>
<author position="01">Moshe Y. Vardi</author>
<author position="00">Ricky Overmyer</author>
<author position="01">Michael Stonebraker</author>
```

and also by trees containing such subtrees. The most preferred tree satisfying $:Stonebraker$ is `<author position="01">Michael Stonebraker</author>`. Similarly, the most preferred trees satisfying $title::Data$ are

```
<title>A Note on Decompositions of Relational Databases.</title>
<title>Implementation of a Time Expert in a Data Base System.</title>
<title>Problems of Optimistic Concurrency Control in Distributed Database Systems.</title>
```

Example 2 (cont.) The search term $country::Austria$ is satisfied by

```
<country id="f0_149" name="Austria" capital="f0_1467" population="8023244"
  datacode="AU" total_area="83850" population_growth="0.41"
  infant_mortality="6.2" ... government="federal republic" ...> ...
</country>
```

The search term $:name:Vienna$ is satisfied by

```
<province id="f0_17447" name="Vienna" ...>
  <city id="f0_1467" country="f0_149" province="f0_17447" ...>
    <name>Vienna</name> <population year="94">1583000</population> </city>
  </province> ...
```

but $name::Vienna$ is satisfied only by a part of it, that is, $\langle name \rangle Vienna \langle /name \rangle$. The latter is also the most preferred tree satisfying $name::Vienna$. (Observe that this query response seems to “verify existence” as opposed to “extract answer”.)

Example 3 (cont.) All the three *articles* given in Example 3 in Section 1 satisfy the search term $article::Dingle$. Similarly, the search term $:Dingle$ satisfies the following three records, while the search term $author::Dingle$ misses the last one (as they both belong to the text content).

```
<author><name>Adam Dingle</name></author>
<author name="A. Dingle" ></author>
<article id="3">
  @inproceedings{IMN97,
    author="Adam Dingle and Ed MacNair and Thao Nguyen",
    title="An Analysis of Web Server Performance",
    booktitle="Proceedings of the IEEE Global Telecommunications
      Conference (GLOBECOM)", year=1999}
</article>
```

3.4 Query Satisfaction

In order for a collection of XML subtrees to satisfy a query, *each* required (resp. optional) search term in the query must (resp. should) be satisfied by *some* subtree in the collection. The notion of satisfaction can be formalized via subtree sequences.

Similarly to Cohen et al [3], a query $Q(t_1, t_2, \dots, t_m)$ is satisfied by a sequence of subtrees and null values (T_1, T_2, \dots, T_m) , if

- For all $1 \leq i \leq m$: if t_i is a signed/plus/required term, then T_i is not the null value.
- For all $1 \leq i \leq m$: if T_i is not the null value, then t_i is satisfied by T_i .

We also say that the set $\{ T_i | 1 \leq i \leq m \wedge T_i \text{ is non-null} \}$ satisfies Q .

3.5 Query Answer

The definition of query answer captures precision, adequacy and coherence of extracted results. A *query answer candidate* for a query $Q(t_1, t_2, \dots, t_m)$ with respect to an XML tree T is a collection of XML subtrees \mathcal{U} of T such that \mathcal{U} satisfies Q , and furthermore, there does not exist another (distinct) satisfying collection of XML subtrees \mathcal{P} that is preferred to \mathcal{U} . (Note that \mathcal{P} is *preferred to* \mathcal{U} , according to the following definition.)

Consider the subtree sequences (P_1, P_2, \dots, P_m) and (U_1, U_2, \dots, U_m) such that $\mathcal{U} = \{ U_i | 1 \leq i \leq m \wedge U_i \neq \text{null} \}$ and $\mathcal{P} = \{ P_i | 1 \leq i \leq m \wedge P_i \neq \text{null} \}$. \mathcal{P} is *preferred to* \mathcal{U} if and only if

1. (Precision) $\forall i : t_i \text{ is a term} \Rightarrow (P_i = U_i) \vee (P_i \text{ is embeddable / is a subtree of } U_i)$, or
2. (Adequacy[Maximal Information]) $\forall i : t_i \text{ is an unsigned term} \Rightarrow (P_i = U_i) \vee (P_i \neq \text{null})$.

The precision criteria captures preference for the smallest subtree that is necessary to demonstrate satisfaction. Unfortunately, for certain keyword queries (such as $::k$), this can yield an answer that seems over-specific.

The adequacy or maximal information criteria captures preference for answers that provide information related to desirable terms, in addition to mandatory information for required terms.

A *query answer* for a query $Q(t_1, t_2, \dots, t_m)$ with respect to an XML tree T is a collection of XML subtrees \mathcal{U} of T such that \mathcal{U} is *interconnected*. Two subtrees T_a and T_b are said to be *interconnected* if the path from their roots to the lowest common ancestor does not contain two distinct nodes with the same element, or the only distinct nodes with the same element are these roots.

The intuition behind interconnected-ness is that if the common ancestor can be viewed as a collection containing multiple entities of the same type, as evidenced by the same node label, then interconnected nodes are related to the same “physical” entity. This can be viewed as coherence criteria.

Furthermore, we can display portions of the *query answer* that are not redundant (that is, skip embedded subtrees that may be repeated). This can be viewed as conciseness criteria.

Example 1 (cont.) The query *authors::, title::* returns three authors-title pairs, and the query *author::, title::* returns four author-title pairs.

Example 3 (cont.) The first two *articles* given in Example 3 in Section 1 match the query *author:name:Dingle, article::*.

Example 4 (cont.) The query *lecturer::Mathematics* results in:

```
<lecturer name="David Billington">
  <teaches>Discrete Mathematics</teaches> </lecturer>
```

while the query *lecturer::,Mathematics* results in three answers:

```
<course name="Discrete Mathematics">
  <lecturer>David Billington</lecturer> </course>
<lecturer name="David Billington">
  <teaches>Discrete Mathematics</teaches> </lecturer>
<lecturer>David Billington</lecturer>
<course>Discrete Mathematics</course>
```

3.6 Ranking Query Answers

In order to deal with document-centric applications of XML, we adapt the traditional TFIDF formula for weighting documents to rank order query answers, somewhat along the lines of Cohen et al [3]. Specifically, we capture the relevance of an XML subtree (containing text) to a query term (containing keyword). The term frequency of a keyword k in a tree T_n , is defined as:

$$tf(k, T_n) = \frac{count(k, T_n)}{\max\{count(i, T_n) \mid i \text{ in } words(T_n)\}},$$

where $count(k, T_n)$ is the number of times k is contained in the text/attribute nodes of the tree T_n , and $words(T_n)$ is the set of keywords contained in the

text/attribute nodes of the tree T_n . Similarly, the inverse document frequency of a keyword k for a subtree with root label / type t is defined as:

$$idf(k, t) = \log \left(1 + \frac{|T(t)|}{|\{T_n \in T(t) \mid k \text{ in } words(T_n)\}|} \right),$$

where $T(t)$ is the set of tree/attribute of type t .

In the context of queries involving elements and/or attributes and keywords, the *rank* of a query answer (T_1, T_2, \dots, T_m) for the query $Q(t_1, t_2, \dots, t_m)$ is

$$\sum \{ tfidf(T_i, t_i) \mid 1 \leq i \leq m \}.$$

For $t_i = t : k \vee t_i = t : _ : k \vee t_i = _ : t : k$, $tfidf(T_i, t_i) = tf(k, T_i) * idf(k, t)$. For $t_i = :: k$, in the absence of context, the inverse document frequency of a keyword k can be changed to:

$$idf(k) = \log \left(1 + \frac{|T|}{|\{T_n \in T \mid k \text{ in } words(T_n)\}|} \right),$$

where T is the set of text/attribute nodes.

This reduces to traditional TFIDF approach for keyword search if a collection of text documents are glued into a single XML document creating one text node per text document. Observe also that the relative rank of the modified XML subtrees does not change if the XML documents are refined by embedding new element tags. Similarly, the relative ranks may be preserved when the XML documents are augmented with attribute-value pairs that reflect the semantics of a piece of text.

To implement this language, we need to build an index, mapping words to paths in the XML document tree, to enable retrieval of subtrees. To compute interconnectedness, parent relationship is essential. Our current implementation uses Lucene [18] for efficient indexing and search of XML documents and does not deal with ranking. Relative to traditional IR, we expect the size of the XML term-document matrix to be very large. Thus, instead of materializing all the TFIDF statistics for each subtree, it may be dynamically computed using the corresponding statistics for the text/attribute nodes contained in the subtree.

4 Conclusions and Future Work

The proposed XML query language has been designed with a straightforward syntax to ease query formulation, incorporating not only text data content but also metadata information in the source XML document. The query semantics has been designed in such a way that the answers provide relevant information and are robust with respect to various manifestations of the same information in terms of XML elements and XML attributes, to improve recall. In other words, the intuitive duality between the usages of elements and attributes has been taken into account. Unfortunately, in relation to traditional IR, certain keyword

queries may yield answers that are over-specific from user's perspective, necessitating inclusion of metadata information in the query to control the granularity of answers. One possible approach to overcome this problem is to specify what kinds of subtrees (in terms of root labels) should be allowed in the answers.

Acknowledgements: We thank the referees for their valuable feedback.

References

1. Brin, S. and Page, L.: The Anatomy of a Large-Scale Hypertextual Web Search Engine, In: Proceedings of the Seventh International Conference on World Wide Web, 1998, pp. 107 - 117.
2. <http://www.cs.washington.edu/research/xmldatasets/>, Retrieved June/2006.
3. Cohen, S., Mamou, J., Kanza, Y., and Sagiv, Y.: *XSEarch: A Semantic Search Engine for XML*, In: The 29th International Conference on Very Large Databases (VLDB). September 2003.
4. Florescu, D., and Donald Kossmann and Ioana Manolescu: *Integrating keyword search into XML query processing*, Computer Networks: The International Journal of Computer and Telecommunications Networking, Vol. 33, Issue 1-6, June 2000, pp. 119-135.
5. Fuhr, N., and Grojohann, K.: *XIRQL: A Query Language for Information Retrieval in XML Documents*, In: Proceedings of the 24th ACM SIGIR Conference, 2001, pp. 172-180.
6. Meyer, H., Bruder, I., Weber, G. and Heuer, A: *The Xircus Search Engine*, 2003.
7. Carmel, D., Maarek, Y.S., Mass, Y., Efraty, N., and Landau, G.M.: *An Extension of the Vector Space Model for Querying XML Documents via XML Fragment*, The 25th Annual ACM SIGIR Conference, 2002.
8. Schlieder, T. and Meuss, H.: *Querying and ranking XML documents*, Journal of the American Society for Information Science and Technology, Volume 53 , Issue 6, 2002, pp. 489-503.
9. Theobald, A., and Weikum, G.: *The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking*, 8th International Conference on Extending Database Technology (EDBT), LNCS 2287, 2002, pp. 477-495.
10. Grabs, T., and Schek, H.: *Generating Vector Spaces On-the-fly for Flexible XML Retrieval*, In: Proceedings of the 2nd XML and Information Retrieval Workshop, 25th ACM SIGIR Conference, 2002.
11. Li, Y., Yu, C. and Jagadish, H. V.: *Schema-Free XQuery*, In: The 30th International Conference on Very Large Databases (VLDB), 2004, pp. 72-83.
12. Catania, B., Maddalena, A., and Vakali, A.: *XML Document Indexes : A Classification*, In: IEEE Internet Computing, 2005, pp. 64-70.
13. Guo, L., Shao, F., Botev C., and Shanmugasundaram, J.: *XRANK: Ranked keyword search over XML documents*, In: Proceedings of ACM SIGMOD, 2003, pp. 16-27.
14. <http://xml.coverpages.org/elementsAndAttrs.html>, Retrieved June/2006.
15. Antoniou, G., and van Harmelen, F.: *A Semantic Web Primer*, The MIT Press, 2004.
16. Fensel, D., Hendler, J., Lieberman, H., and Wahlster, W. (Eds.): *Spinning the Semantic Web: Bringing the WWW to Its Full Potential*, The MIT Press, 2003.
17. <http://www.w3.org/TR/>, Retrieved June/2006.
18. <http://lucene.apache.org/java/docs/>, Retrieved June/2006.