A Coherent Query Language for XML

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
http://www.cs.wright.edu/~tkprasad
(Voice) 937-775-5109 (FAX) 937-775-5133

October 26, 2007

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. Our approach combines and generalizes several available techniques to obtain precise and coherent results.

Index Terms: Query Languages, XML/RDF, Information Search and Retrieval, Indexing and Search.

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 most of the textual

information available in the metadata / annotations / XML tags[1]. (b) They are oblivious to the underlying aggregation structure implicit in the tree-view of XML documents. On the other hand, specialized query languages and search engines for querying XML documents require some sophistication on the part of the user to formulate queries [2, 3].

In this paper, we present with illustrative examples, design and implementation of a keyword-based XML query language that is simple to use and that 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. This work builds on the XML query language proposed in [4].

Consider the following motivating example of querying an heterogeneous XML document. It describes titles and authors of publications. A publication can be a book or an article. An edited book can contain chapters with separate titles and authors. An article's author may be missing.

Example 1

```
<publications>
   <book>
     <title> Modern Information Retrieval </title>
     <author> Ricardo Baeza-Yates  </author>
     <author> Berthier Ribeiro-Neto </author>
           <chapter>
              <title> Digital Libraries </title>
              <author> Edward A. Fox </author>
              <author> Ohm Sornil </author>
           </chapter>
    </book>
   <article>
     <title>The Anatomy of a Large-Scale Hypertextual Web Search Engine
     </title>
     <author> Sergey Brin </author>
     <author> Lawrence Page </author>
    </article>
   <article>
```

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

```
     <title> An Algorithm for Suffix Stripping </title>
     <author> M.F.Porter </author>
    </article>
   <article>
     <title> Indexing by Latent Semantic Analysis </title>
    </article>
</publications>
```

For the query that seeks author-title pairs, our semantics provides eight possible answers:

```
     <title> Modern Information Retrieval </title>
     <author> Ricardo Baeza-Yates </author>

     <title> Modern Information Retrieval </title>
     <author> Berthier Ribeiro-Neto </author>

              <title> Digital Libraries </title>
              <author> Edward A. Fox </author>

              <title> Digital Libraries </title>
              <author> Ohm Sornil </author>

    <title>The Anatomy of a Large-Scale Hypertextual Web Search Engine </title>
    <author> Sergey Brin </author>

    <title>The Anatomy of a Large-Scale Hypertextual Web Search Engine </title>
    <author> Lawrence Page </author>

     <title> An Algorithm for Suffix Stripping </title>
     <author> M.F.Porter </author>

     <title> Indexing by Latent Semantic Analysis </title>
```

Observe that the author(s) and the title of the same entity (that is, article, book, chapter, etc) are grouped together, while the author(s) and the title of different entities are not, even when one entity is nested within the other (such as chapter within book). This is accomplished by exploiting *limited homogeneity* in the data (for example, articles) and the *explicit structural similarity* in the heterogeneous records (for example, book vs chapter vs article). Note that author and title are implicitly optional. So, for the last answer, a response for author is not mandatory. However, a richer query that requires both author and title to be present is supported, to eliminate the following "author-less" answer.

```
<title> Indexing by Latent Semantic Analysis </title>
```

The query language and its semantics also provides a mechanism to determine co-authors when present, eliciting the following four query answers.

```
<author> Ricardo Baeza-Yates  </author>
<author> Berthier Ribeiro-Neto </author>

         <author> Edward A. Fox </author>
         <author> Ohm Sornil </author>

<author> Sergey Brin </author>
<author> Lawrence Page </author>

<author> M. F. Porter </author>
```

Notice how the co-authors of a book are segregated from the co-authors of a chapter in the book and from the co-authors of an article. A single author entity (that is, M. F. Porter case) is permitted, but the "maximality" requirement eliminates the undesirable appearance of other authors (such as Sergey Brin) as single author answers. Furthermore, the semantics enables grouping of the title of a book and its "nested" chapter together when necessary.

```
<title> Modern Information Retrieval </title>
         <title> Digital Libraries </title>

<title>The Anatomy of a Large-Scale Hypertextual Web Search Engine </title>

<title> An Algorithm for Suffix Stripping </title>

<title> Indexing by Latent Semantic Analysis </title>
```

The present work builds on the seminal work of Cohen et al [6] on XSEarch, a search engine for XML documents. The query language for heterogeneous documents 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[2] and coherent results (that is, containing semantically related pieces of information). The query language accommodates XML-attributes and their string values, and incorporates precision improvements suggested by Li et al [10] and recall improvements suggested by Guo et al [24] among other things. We avoid discussing relevance ranking issues, to focus on the semantics of the core query language. Specifically, the query semantics has the following additional benefits:

[2]Our use of the terms "precise" and "precision" is meant to capture the aspect of "compactness" or "minimal size complete coverage" or "specificity", and should not be confused with the standard IR term "precision" meaning a fraction of retrieved documents (or XML fragments) that are relevant to the searcher.

- Coherent query responses are generated not only for homogeneous XML documents but also for heterogeneous XML documents. Our approach also provides a reasonable interpretation of repeated query terms as explained in Section 3.6.

- 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. For example, see Mondial database [5] in which a lot of factual data is captured via attribute bindings.

- 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 [26], it is still quite common to see authoring variations that mix the two, based on the idiosyncratic preferences and views of the authors of web documents. (For example, see heterogeneous XML documents in [8]). Treating the following patterns as "equivalent" can improve query *recall*: `<T A="s"/>` and `<T> <A> s </A> </T>`.

- Semantic Web formalisms such as RDF, OWL, etc [27, 9, 28] build on XML and make extensive use of attributes in XML serialization of RDF model [9]. In fact, the two forms — `<T A="s"/>` and `<T> <A> s </A> </T>` — 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.

- 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. Our formalization attempts to be robust with respect to such refinements of a document.

In Section 2, we present other related works. In Section 3, we discuss the details of the proposed XML query language, develop its semantics, and illustrate its use through examples. Specifically, we present a novel approach to obtain precise and coherent results. In Section 4, we provide algorithmic details of a prototype implementation. In Section 5, we conclude with suggestions for future work.

2 Other Related Works

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfy an information need from within large collections (usually stored on computers) [12]. In contrast, data retrieval is finding answers of a structured nature that satisfy a well-defined query from within large structured collections. Typically, IR involves issues such as quality of results, partial matching, relevance-based ranking of results, indexing and retrieval of unstructured data, etc. In contrast, DR involves issues such as efficiency of result computation, exact matching, no results or too many results, indexing and retrieval of structured data, etc. IR techniques are usually justified on probabilistic grounds, and were initially used in library applications. In contrast, DR techniques are usually formalized using algebra/logic, and were initially used in accounting applications.

XML query languages can be broadly classified into two groups: (a) Structure-based query languages (inspired by the database community and data-centric applications) and (b) Keyword-based query languages (inspired by the information retrieval community and the document-centric applications). The former group of languages can be further classified into: (a.1) Navigational languages that use path expressions to refer to XML fragments (for example, XQL [11], Quilt [13], XQuery [3], etc) and (a.2) Positional languages that use XML patterns to refer to XML fragments (for example, XML-QL [14], Xcerpt [15], etc). Bailey et al [16] provide an up-to-date survey of structure-based (non-IR) XML query languages, and we urge interested readers to consult their work for details. They also discuss the design space for the XML query languages and compare the various languages on the basis of their support for: (i) selection and extraction of XML data, (ii) restructuring of XML data, and (iii) aggregation, combination, and inferences from XML data. In fact, XQuery [3] has become the de facto standard for structure-based query language because it is flexible, expressive, reliable, and full, with wide support from both industry and academia.

Florescu et al [8] present the design and implementation of a keyword-based extension of XML-QL language (using *contains* predicate) that supports querying of XML documents based on their structure and their textual content. This facilitates searching of an XML document whose structure is only partially known, or searching heterogenous XML document collections. The implementation uses an off-the-shelf relational database system.

In XIRCL, Fuhr and Grojohann [17] incorporate different answer granularity, content-based searching, robust query matching (by equating use of an element or an attribute for expressing the same content), and IR-related features such as relevance ranking using probabilistic models. Meyer et al [18] describe a search engine architecture and implementation

based on XIRCL.

Carmel et al [19] 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.

Schlieder and Meuss [20] 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 a different interpretation in our approach that is robust with respect to the level of annotations.

In contrast with approaches so far, Theobald and Weikum [21] focus on heterogenous documents, path-based queries and semantic-similarity search conditions. Grabs and Schek [22] 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 [10] propose an extension to XQuery that marries the precise query formulation abilities of XQuery with the convenience and intuitive semantics of keyword-based XML query language. Specifically, they show how their semantics is robust with respect to variations in XML Schema, and how it can be efficiently implemented. Our approach lacks the precision that full XQuery exhibits, but it can incorporate improvements in precision for keyword-based query languages as suggested by Li et al [10]. In our opinion, a simple keyword-based query language that facilitates "human-in-the-loop" for query refinement seems appropriate for quick browsing and searching of XML documents, compared with the effort required for query formulation in XQuery. However, XQuery-like languages are indispensable when dealing with large datasets and/or expressive queries.

Guo et al [24] describe a novel scheme for ranking keyword-based search results over hyperlinked XML documents that takes into account (i) the nested structure of XML documents, for result specificity, (ii) hyperlinks structure in the form of IDREFs and XLinks, for "pageranking", and (iii) proximity among keywords with respect to distance between them and distance to an ancestor, for improving precision. In contrast, our approach ignores hyperlinks in XML documents. Similarly to Guo et al's approach, our "completeness" criteria computes all possible answers as explained in Section 3.5.

Cohen et al [7] provide a *comprehensive framework* for developing interconnection semantics for keyword-based querying of XML documents with IDREFs, and relevant computational complexity results for verifying an answer to a query or enumerating all answers to a query. The two definitions of interconnectedness we combine, and the notion of completeness we advocate, can individually be represented in their framework, among various other choices. However, the repeated-term queries and their proposed semantics (e.g., queries for determining co-authors) cannot be captured straightforwardly via the automatically generated uniquely labelled patterns of Cohen et al [7].

Catania et al [23] provide a nice review of the indexing schemes employed for querying XML documents. In terms of their classification, our implementation uses "Structural Join Indexing" because it is based on Apache Lucene APIs [30], a high-performance Java infrastructure for building fulltext search engine.

3 Query Language

The standard search engine query is a list of optionally signed keywords. For querying XML documents, Cohen et al [6] 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].

3.1 Query Syntax

Definition 1 *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*.*

Informally, ':' corresponds to *with attribute* or *with value*, and '::' corresponds to *nested to any depth*.

[3]Word adjacency information can be captured using phrases in the standard way, that is, delimited by double quotes. However, there is no explicit support for specifying proximity information or "window of separation" between two words, in the query language.

Definition 2 *A* query *is a sequence of optionally signed search terms. Signed search terms* must *be present in the retrieved results, while unsigned search terms* should *be present in the retrieved results.*

That is, unsigned search terms must be present if satisfying information is available. Otherwise, it can be absent. The rationale is to get *maximal information* from the available data corresponding to unsigned search terms. This interpretation is formally captured in Definition 5 of query answer candidate given in Section 3.5.

Note that ',' is used to separate terms in a sequence.

3.2 XML Datamodel

Conceptually, an XML document is modelled as an ordered tree [29]. For our purposes, XML tree contains the following types of nodes[4]:

- *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. Each attribute node has a string-value.

We ignore *namespace* nodes, *processing instruction* nodes, and *comment* nodes, and the ability to create additional internal *links* between tree nodes[5].

[4]In this paper, a subtree is always rooted at an element node. The root node associated with a subtree of a tree (resp. a subelement of an element) is a descendant of the root node associated with the tree (resp. element).

[5]From the perspective of specifying the query semantics, there is no harm in allowing internal links. These have been excluded here only because our implementation prematurely committed to tree structure, thereby requiring some duplication for testing with DAGs generated by IDREFs. An alternative implementation allowing multiple parents can enable querying of more general XML documents.

3.3 Single Search Term Satisfaction

We specify when a search term is *satisfied* by an XML subtree. 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 </A> </T>`). Additionally, the notion of satisfaction is robust with respect to further refinements through semantic annotations (for example, `<T> s </T>` and `<T> <A> s </A> </T>` satisfy the query `T::s`).

Observe that a text node / attribute value can be defined to contain a keyword in several different ways. Some of the alternative definitions for when a keyword is *contained* in a text node / attribute value are: (i) if the keyword is a prefix of a word in the text node, or (ii) the keyword is identical to the string in the text node, or (iii) the keyword belongs to the text node viewed as a bag of words, or (iv) the keyword matches a word in the text node after case conversion, stemming, and synonym expansion, etc. Furthermore, in order to enable equivalence between attribute-value form and subelement-text node form discussed earlier, we need to process the string in the attribute's value and the text in the corresponding text node similarly. Specifically, interpretation (i) seems appropriate for data-centric XML where attribute values may be treated as atomic, while interpretation (ii) and (iii) seem appropriate for document-centric XML where the attribute values may be English phrases. We consciously avoid resolving this debate here and focus on the essentials.

Definition 3 *We define when a search term is* satisfied *by an XML subtree (by cases) as follows.*

- 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.
- 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 subelement 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.

Informally, a search term query is satisfied by an XML tree if the tree contains query relevant information. Observe that a query can use detailed knowledge of XML document structure (for example, via *e:a:k* etc), or have the flexiblity to express textual search (for example, via *::k*, etc).

The above definition makes explicit all possible cases that arise. Furthermore, it captures (i) equivalence between attribute-value pair and its rendition as entity-text pair, and (ii) equivalence among XML trees obtained though refinement via additional annotations.

We illustrate the notion of satisfaction through concrete examples, setting the stage for defining "most preferred" answers.

Example 2 Consider the following heterogeneous XML document [8].

```
<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. Thus, all the three `article`s satisfy the search term `article::Dingle`. These trees also satisfy the search terms `:author:`, `title::`, `::Dingle`, etc. Furthermore, the search terms `::Dingle` and `author::` are satisfied by the following three fragments, while the search term `author::Dingle` eliminates the last one (as both `author` and `Dingle` belong to the text content) and the search term `:author:dingle` eliminates the last two. The equivalence between attribute-value pair and its translation via entity-text pair is respected.

```
<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, and furthermore, all the subtrees in the collection must be meaningfully related. To materialize the absence of a satisfying tree for an optional search term, the *null* value has been introduced. The notion of satisfaction can be formalized via subtree sequences [6].

Definition 4 *A query* $Q(t_1, t_2, ..., t_m)$ *is* satisfied *by a sequence of subtrees and null values* $(T_1, T_2, ..., T_m)$*, if*

- *For all* $1 \leq i \leq m$*: if* t_i *is a signed/plus/required term, then* $T_i \neq null$*.*
- *For all* $1 \leq i \leq m$*: if* $T_i \neq null$*, then* t_i *is satisfied by* T_i*.*

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

Note that, as it stands, the "," operator implicitly captures the boolean connective "AND", while unsigned search term approximates the boolean connective "OR". Furthermore, it is straightforward to extend the query language to support the boolean connectives "AND" and "OR" with the desired semantics. We have made this fact explicit in the paper now.

3.5 Query Answer

Informally, the definition of query answer tries to capture the intuitive notions of precision, completeness, adequacy, and coherence of extracted results.

Tentatively, a *query answer candidate* for a query $Q(t_1, t_2, ..., t_m)$ 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 $\mathcal{U}$.

Consider the subtree sequences $(P_1, P_2, ..., P_m)$ and $(U_1, U_2, ..., U_m)$ such that $\mathcal{U} = \{U_i \mid 1 \leq i \leq m \wedge U_i \neq null\}$ and $\mathcal{P} = \{P_i \mid 1 \leq i \leq m \wedge P_i \neq null\}$.

Tentatively, $\mathcal{P}$ is *preferred to* $\mathcal{U}$ if and only if

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

Note that $\mathcal{P}$ is *preferred to* $\mathcal{P}$, according to the above definition.

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 overly specific.

Example 3 For the query `+::pWord, +::qWord` and the XML tree:
`<A> pWord <B> pWord qWord </B> pWord qWord </A>`,
the satisfying XML subtree `{ <B> pWord qWord </B> }` is preferred over
`{ <A> pWord <B> pWord qWord </B> pWord qWord </A> }` because the former is embeddable in the latter. However, note that the second tree has "hits" independent of those in the first subtree. We overcome this limitation shortly.

Example 4 There is some asymmetry in the treatment of elements and attributes here. For the query `+::pWord, +::qWord` and the XML tree: `<A> <B> pWord qWord </B> </A>`, the set `{ <B> pWord qWord </B> }` is the query answer candidate, but not
`{ <A> <B> pWord qWord </B> </A> }`,
while, for the XML tree `<A B="pWord qWord"> </A>`, the set `{ <A B="pWord qWord"> </A> }` is the query answer candidate.

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.

Example 5 For the query `::pWord` and the XML tree: `<A> <B> pWord qWord </B> </A>`, the satisfying XML subtree `{ <B> pWord qWord </B> }` is preferred over the empty set `{ }`, due to adequacy requirement. However, the query `::rWord` and the same XML tree yields the empty set `{ }` as a viable query answer candidate.

In order to remedy the problem alluded to in Example 3, we define *query answer candidate* as follows:

Definition 5 *A* query answer candidate *for a query $Q(t_1, t_2, ..., t_m)$ with respect to an XML tree T is a collection of XML subtrees $\mathcal{U}$ of T such that $\mathcal{U}$ satisfies Q, and*

1. *(Preference) there does not exist another (distinct) satisfying collection of XML subtrees $\mathcal{P}$ such that*

   (a) *(Precision)* $\forall i : t_i$ *is a term* $\Rightarrow$ (P_i *is a subtree of* U_i), *OR*

   (b) *(Adequacy[Maximal Information])*
   $\forall i : t_i$ *is an unsigned term* $\Rightarrow$ $(P_i = U_i) \vee (P_i \neq null)$.

OR

2. *(Completeness)*
   $\exists i : t_i$ *is a term, AND* $\exists ER_i$:
   ($(U_1, ..., U_i, ...U_m) - ER_i$ *is a query answer candidate* Q *w.r.t.* $T - ER_i$)),
   where $ER_i = \{R_i \mid R_i$ *is a subtree of* $U_i \wedge$
   $(R_1, ..., R_i, ..., R_m)$ *is a query answer candidate for* $Q\}$.

Note that $((T_1, ..., T_m) - STs)$ denotes the subtree sequence obtained by deleting subtrees in STs from the subtrees T_i's.

Informally, completeness criteria[6] permits "non-minimal" query answer candidates that contain subtrees (satisfying some terms) *not covered* by other "smaller" query answer candidates. $(R_1, ..., R_i, ..., R_m)$ in the definition ranges over the latter, while the construction ($(U_1, ..., U_i, ...U_m) - ER_i$) tries to uncover the former. (See examples below for further clarification.) Recall also that for the subtree sequence $(U_1, U_2, ..., U_m)$, the corresponding set of subtrees is $\mathcal{U} = \{U_i \mid 1 \leq i \leq m \wedge U_i \neq null\}$.

In what follows, we use the set and the sequence representations interchangeably for convenience, whenever it is clear from the context which is meant.

The precision and completeness criteria captures preference for the smallest fragment of a subtree that satisfies the query while simultaneously accommodating every possible occurrence of an answer. Specifically, it allows non-overlapping answers in a tree that also contains subtrees with other answers. To illustrate the subtleness involved, let us analyze the examples shown in Figure 1.

Example 6 Consider the query `::pWord, ::qWord` and the XML tree:
`<A> pWord qWord <B> pWord qWord </B> <C> rWord qWord pWord </C> </A>`,
and the three XML subtrees given below:

Trees-1: `{ <B> pWord qWord </B> }`

Trees-2: `{ <C> rWord qWord pWord </C> }`

Trees-3: `{ <A> pWord qWord <B> pWord qWord </B> <C> rWord qWord pWord </C> </A> }`

According to tentative definition, (Trees-1) and (Trees-2) are regarded as query answer candidates, while (Trees-3) is not, because (Trees-1) and (Trees-2) are embedded in (3). However, according to Definition 5, all the three subtrees (Trees-1), (Trees-2), and (Trees-3)

[6]Ironically, the declarative specification of the completeness criteria seems complex. However, its purpose, as illustrated through the example, is clear and its implementation, as discussed later, is clean.

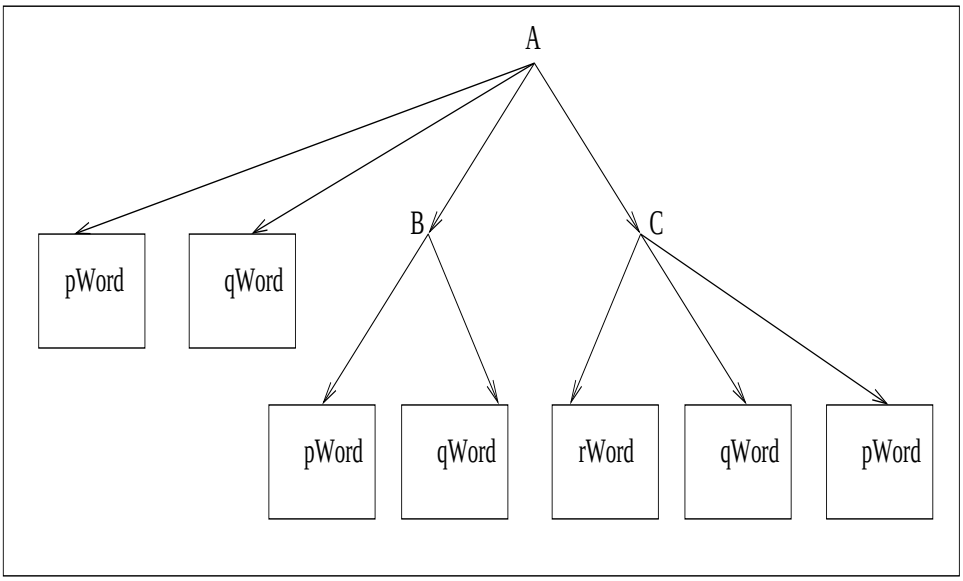

Figure 1: Illustration of Precision and Completeness

are regarded as query answer candidates. Specifically, (Trees-3) contains an occurrence of `pWord qWord` that is distinct from the ones found in (Trees-1) and (Trees-2), and hence been included as query answer candidate.

Observe also that, for certain queries, the precision criteria can yield an answer that seems overly specific. For example, for the query `B::pWord` and the XML tree: `<A> <B> pWord </B> </A>`, the answer is: `<B> pWord </B>`

Definition 6 *(Coherence) A* query answer *for a query $Q(t_1, t_2, ..., t_m)$ with respect to an XML tree T is a collection of XML subtrees $\mathcal{U}$ of T such that $\mathcal{U}$ is a query answer candidate and $\mathcal{U}$ is interconnected.*

3.6 Coherence via Interconnectedness

Cohen et al [6] and Li et al [10] present two different definitions of interconnectedness. Li et al's definition is more stringent than Cohen et al's definition, and seems to be a better fit for heterogeneous or Schema-free XML documents. The distinction between these two definitions is analogous to the distinction between name equivalence and structural equivalence of types in programming languages [25]. Two variables are said to be *name (type) equivalent* if the two variables have been explicitly declared as having the same type name. As explained below, Cohen et al [6] exploit the homogeneity in the element names in an XML document. Two variables are said to be *structure (type) equivalent* if the two

variables (recursively) contain fields of the same structure type. As explained below, Li et al [10] exploit the structural similarity to deal with heterogeneity. Our approach combines and generalizes these two proposals to capture coherence implicit in an XML document.

Recall that a node c is an *ancestor* of node a in a tree T if node c is on the (directed) path from node a to the root of the tree T. A node c is a *common ancestor* of nodes a and b in a tree T if node c is an ancestor of both nodes a and b in the tree T. A node c is a *least common ancestor* of nodes a and b in a tree T if node c is a common ancestor of nodes a and b in the tree T, and further, any common ancestor of nodes a and b in a tree T is an ancestor of c.

Definition 7 [Cohen et al:] *Two subtrees T_a and T_b are said to be* interconnected*, if the path from their roots to the lowest common ancestor in the tree does not contain two distinct nodes with the same element label, or the only distinct nodes with the same element label are these roots.*

The intuition behind this notion of interconnectedness 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 belong to a subtree that can be associated with a single physical entity. (See Figure 2(A).)

Definition 8 [Li et al:] *Two subtrees T_a and T_b are said to be* interconnected*, if the path from T_a's root to their lowest common ancestor in the tree does not contain another node that is the lowest common ancestor of T_a and a distinct subtree T'_b, where T'_b has the same root element label as T_b.*

The intuition behind this notion of interconnectedness is that if the common ancestor can be viewed as a collection containing multiple entities with similar structure, as evidenced by subtrees containing similarly labelled descendant nodes, then interconnected nodes belong to a subtree that can be associated with a single physical entity. In particular, the formal definition of interconnectedness specifies what constitutes an "overriding" association (that is, T_a and T'_b, or T'_a and T_b) in a subtree that can eliminate "erroneous" association (that is, T_a and T_b) in a larger tree. (See Figure 2(B).)

For example, consider Figure 2. In (A), according to both Cohen et al's notion and Li et al's notion, nodes in sets {`Title1, Author1`} and {`Title2, Author2`} are interconnected, while the nodes in sets {`Title1, Author2`} and {`Title2, Author1`} are *not* interconnected. However, in (B), according to Cohen et al's notion, nodes in sets {`Title3,`

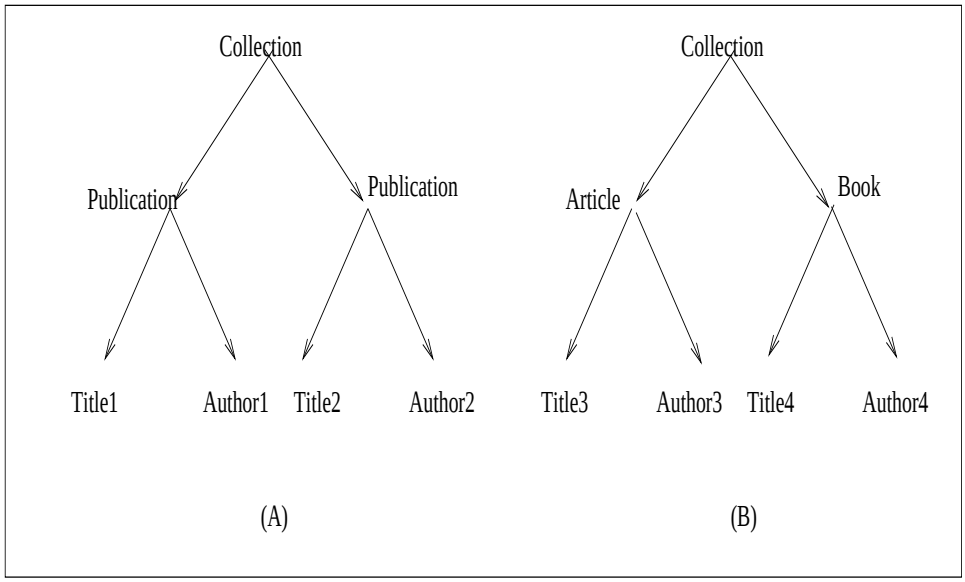

Figure 2: Interconnection : (A) Homogeneous (B) Hetergeneous

`Author3`}, {`Title4, Author4`}, {`Title3, Author4`} and {`Title4, Author3`} are all interconnected, while, according to Li et al's notion, only nodes in sets {`Title3, Author3`} and {`Title4, Author4`} are interconnected, but nodes in sets {`Title3, Author4`} and {`Title4, Author3`} are *not* interconnected. In other words, Li et al's approach yields intuitively better answers than Cohen et al's approach for the example shown in Figure 2(B).

It is possible to get "the best of both the worlds" by combining and generalizing Definition 7 and Definition 8 as follows:

Definition 9 *Two subtrees T_a and T_b are said to be* interconnected*: (i) if the path from their roots to the lowest common ancestor in the tree does not contain two distinct nodes with the same element label, or the only distinct nodes with the same element label are these roots,* or*, (ii) if the path from T_a's root to their lowest common ancestor in the tree does not contain another node that is the lowest common ancestor of T_a and a distinct subtree T'_b, where T'_b has the same root element label as T_b.*

Specifically, the above notion of interconnectedness allows us to use similar element labels to identify different entities of the same type *even when the structure of the entities do not coincide, for example, due to incomplete information or the presence of optional features*, while falling back on the structural information to glean coherence when element labels alone are inadequate. See Example 9.

In order to appreciate the subtleness implicit in the definition of interconnectedness given in Definition 9, consider the heterogeneous XML document of Example 1.

Example 7 For the query `author::, title::`, the eight possible query answers are:

```
<title> Modern Information Retrieval </title>
<author> Ricardo Baeza-Yates </author>

<title> Modern Information Retrieval </title>
<author> Berthier Ribeiro-Neto </author>

          <title> Digital Libraries </title>
          <author> Edward A. Fox </author>

          <title> Digital Libraries </title>
          <author> Ohm Sornil </author>

<title>The Anatomy of a Large-Scale Hypertextual Web Search Engine </title>
<author> Sergey Brin </author>

<title>The Anatomy of a Large-Scale Hypertextual Web Search Engine </title>
<author> Lawrence Page </author>

 <title> An Algorithm for Suffix Stripping </title>
 <author> M.F.Porter </author>

 <title> Indexing by Latent Semantic Analysis </title>
```

Observe how the author(s) and the title of the same "real-world" entities cohere together including the case of nested entities. This is due to the fact that we use homogeneity (for example, articles) and the structural similarity (for example, book vs chapter vs article). The query terms are optional, and so, for the last answer, a binding for `author::` is not necesssary. On the other hand, for the query `+author::, +title::`, there are only seven possible query answers.

Example 8 To determine the co-authors, one can form the *repeated term* query `author::, author::`, eliciting 4 query answers.

```
<author> Ricardo Baeza-Yates  </author>
<author> Berthier Ribeiro-Neto </author>

          <author> Edward A. Fox </author>
          <author> Ohm Sornil </author>
```

```
<author> Sergey Brin </author>
<author> Lawrence Page </author>

<author> M. F. Porter </author>
```

The answers do not change for the query `+author::, +author::` or `+author::, +author::, +author::`. Specifically, the single author tree (that is, M. F. Porter case) is regarded as satisfying the query, and the query language is not capable of expressing its elimination. However, maximal information requirement does eliminate other authors (such as Sergey Brin) from appearing as single author answers.

Example 9 For the queries `+title::, +title::` and `title::, title::`, we get 4 query answers shown below. Note how the titles of the book and the chapter are grouped together.

```
<title> Modern Information Retrieval </title>
         <title> Digital Libraries </title>

<title>The Anatomy of a Large-Scale Hypertextual Web Search Engine </title>

<title> An Algorithm for Suffix Stripping </title>

<title> Indexing by Latent Semantic Analysis </title>
```

Similarly, the query `+author::, +title::, +author::` yields four answers: one book, one chapter, and only two articles.

Example 10 Note that for the query `author::, title::`, and the following heterogeneous XML document (where the title of the book is missing and the authors of the article are missing):

```
<publications>
   <book>
     <author> Ricardo Baeza-Yates  </author>
     <author> Berthier Ribeiro-Neto </author>
    </book>
   <article>
     <title> Indexing by Latent Semantic Analysis </title>
    </article>
</publications>
```

we do get the "unfortunate" pairings

```
<author> Ricardo Baeza-Yates  </author>
<title> Indexing by Latent Semantic Analysis </title>

<author> Berthier Ribeiro-Neto </author>
<title> Indexing by Latent Semantic Analysis </title>
```

Even though these query answers do not model "reality" (because the book's authors are paired with the title of an unrelated article), these pairings are still considered reasonable as there is nothing in the data to imply incoherence. For instance, unless a background ontology provides information that `book` and `article` are similar (that is, they are two kinds of publications), there is no way to determine that `book` and `article` elements delimit information about two separate real-world entities.

3.6.1 Computing Interconnectnedness for multi-term query

For $m \geq 2$: the sequence of subtrees $(U_1, U_2, \ldots, U_m)$ are interconnected *if and only if* ($\forall i, j \in [1, 2, \ldots, m]$: U_i and U_j are interconnected).

4 Algorithmic and Implementation Details

We describe the algorithmic details of XML dataset indexing and query answering, and our prototype based on the open source text indexing and search API, Lucene [30]. The purpose of providing these details is to show that the semantics can be implemented in a straightforward manner. In fact, the prototype enabled us to analyze the subtleties associated with the query semantics and the expressive power of the query language. For ease of prototyping, we mapped the problem of XML indexing and querying to one of text indexing and querying supported by Lucene. To obtain significant improvements in performance, we should replace Lucene with custom XML indexing and search system built from scratch.

4.1 Indexing

XML documents are indexed to enable efficient search for query answers. The answers are eventually a set of XML fragments. Each XML fragment of an XML document can be represented using two pieces of information: the name of the XML document, and the path from the root of the entire XML document to the element node that is the root of the XML fragment. Thus, the indexing data structure should map each "word" to a list of pairs `(XML-document-filename, set-of-access-paths-from-document-root)`. Observe

that this is analogous to a posting list for a keyword which contains `(text-filename, set-of-positions-of-hits-in-text)` that captures positional indexing in traditional IR. The names of the nodes on the access path from the document root to a node is also the sequence of names of the ancestor elements of the node along with their locations with respect to their siblings. For concreteness, in Example 1, the location of the hit for the query `author::Sornil` is the XPath `/publications[1]/book[1]/chapter[1]/author[2]` and for the query `title::Latent` is the XPath `/publications[1]/article[3]/title[1]`.

Assume that '{...}' denotes a set, and a *position* of a node of an XML document captures the access path from the document root to the node (similarly to XPath). Also, let us call the form of the word that is indexed as the *significant keyword*. This form depends on the notion of containment of a keyword in a text node or an attribute value. For example, it can be an entire word, or a prefix of a word, or a non-stop word, or a stem. Then:

- Each element e's occurrence in the document f at the position pe is encoded in the mapping of the element name $map(e)$ as $\{..., (f, \{..., pe, ...\})\}$.
- Each attribute a's occurrence in the document f, with its parent element located at the position pe, is encoded in the mapping of the attribute name $map(a)$ as $\{..., (f, \{..., pe, ...\})\}$.
- Each significant keyword sk occurring in an attribute value av in the document f, with the associated attribute located at the position pa, is encoded in the mapping of the stem $map(sk)$ as $\{..., (f, \{..., pa, ...\})\}$. The position retains information about the attribute name to enable distinguishing between values associated with different attributes.
- Each significant keyword sk occurring in a text node t in the document f, with the parent element located at the position pe, is encoded in the mapping of the stem $map(sk)$ as $\{..., (f, \{..., pe, ...\})\}$.

Note that the mappings of the elements, the attributes, the attribute values, and the keywords should be stored separately because the query language makes a distinction between them. These mappings can be implemented using techniques that differ in terms of the space-time trade-offs they make, such as by using hash tables, or by using a full-text indexing and search engine infrastructure.

4.2 Precise and Complete Query Satisfaction

To specify precise and complete answers to a query term, we consider the most general query form $+e : a : k$. (In what follows, we use k also to stand for significant keyword.)

Step 1: Determine `map(e), map(a)`, and `map(k)`. Recall that `map(_)` yields a list of pairs, where each pair is of the form `(filename, set-of-access-paths)`.

Step 2: From the cross-product `map(e) x map(a) x map(k)`, (i) filter triples that belong to the same XML fragment by checking that each component of the triple originates from the same file, and the access paths for the three components are in (proper) prefix relationship, and (ii) return the pair `(filename, set-of-satisfying-access-paths)` associated with the elements of the filtered triples, where the satisfying access path yields the tree that contains element e, attribute a, and keyword k. The path will eventually be used to determine XML fragments.

The specifications for the other queries are similar (in fact a specialization of the above code). Recall that, for the query `+T:A:`, the XML documents

`<T A="s"/>` and `<T> <A> s </A> </T>`

eventually return the XML fragments

`<T A="s"/>` and `<T> <A> s </A> </T>` respectively,

thus behaving similarly. On the other hand, for the query `:A:`, the XML documents `<T A="s"/>` and `<T> <A> s </A> </T>` eventually return the XML fragments `<T A="s"/>` and `<A> s </A>` respectively, which are not the same. The latter difference arises because the smallest returned subtree must be rooted at an element.

Optional query terms can succeed even if there are no non-trivial satisfying XML subtrees, while required query terms will fail under similar circumstances.

4.3 Query Answers : Multiple Query Terms and the Interconnected Relationship

To specify query answers to conjoined queries, we begin with two query terms case. We also provide as comments additional code to ensure that optional query terms will succeed with non-trivial subtree bindings when such bindings exist, and succeed with no bindings when such bindings do not exist. This is to ensure that adequacy/maximal information criteria is met by the query answer. Recall that the query result associated with a query term so far is a set of pairs, where each pair is of the form `(filename, element-access-path)`. To determine the XML fragments that satisfy the two terms `s` and `t`, we compute the common prefix of the access-path components of the result (in the above code) for `s` and `t` with the same filename component, and check for interconnectness, to ensure that the two terms are satisfied by an XML fragment corresponding to a single entity. (The common prefix is the

access path to the lowest common ancestor of the roots of the subtrees (XML fragments) satisfying `s` and `t`.) For example, to implement Cohen et al's definition, we ensure that the two non-overlapping parts of the access paths have disjoint labels.

In what follows, `common-prefix(p,q)` yields the prefix string common to paths `p` and `q` (from the root of the same XML tree) and `remainder-suffix(p,q)` yields a string `r` such that `concatenation(q,r) = p`. `is-label-disjoint(aps,apt)` takes two access paths, and verifies that the labels on the nodes (with the exception of the last nodes) of the two paths are disjoint. For instance, assuming that the paths are represented as alternating sequence of node labels (`n's, m's`) and their "sibling" positions (`a's, b's`),

```
is-label-disjoint( [<m1,a1>,<m2,a2>,...,<mj,aj>,<m,a>],
                   [<n1,b1>,<n2,b2>,...,<ni,bi>,<n,b>] )     if and only if
{m1, m2, ..., mj} is disjoint from {n1, n2, ..., ni}.
```

Note that the terminal nodes on the two paths can have the same label (that is, $\mathtt{n} = \mathtt{m}$) and be part of the same "higher-level" entity.

The following pseudo-code combines and generalizes Cohen et al's definition and Li et al's definition of interconnectedness. (The pseudo-code is explicitly dealing only with required query terms. The commented lines can be used for handling optional query terms. The `pairing-found` predicate ensures that an optional query term can be satisfied without generating any bindings for it, if no bindings exist for it.)

```
query-answer := {};

//* foreach  stp in union(result(s),result(t))
//*          pairing-found(stp) := false;

foreach    (sp,tp)    in     result(s) x result(t)
begin
  apsp := access-path(sp);
  aptp := access-path(tp);
  if (filename(sp) = filename(tp))   and
    is-label-disjoint(remainder-suffix(apsp,common-prefix(apsp,aptp)),
                      remainder-suffix(aptp,common-prefix(apsp,aptp)))
      // Cohen et al's constraint above
  then
      // Li et al's constraint below
```

```
      overriden := false;
      foreach sp0 in result(s)
      begin
        apsp0 := access-path(sp0);
        if ( is-prefix( common-prefix(apsp,aptp), common-prefix(apsp0,aptp) )
             and (apsp0 != aptp) )
        then begin overriden := true;
                   goto FINAL;
             end;
      end;
      foreach tp0 in result(t)
      begin
        aptp0 := access-path(tp0);
        if ( is-prefix( common-prefix(apsp,aptp), common-prefix(apsp,aptp0) )
            and (apsp != aptp0) )
        then begin overriden := true;
                   goto FINAL;
             end;
      end;
      FINAL:
//*       pairing-found(sp) := true;
//*       pairing-found(tp) := true;
       query-answer += if overridden
                       then {}
                       else {( filename(sp), { apsp, aptp } ) )};
   end; // if
end; // foreach

//* foreach  sp in result(s)
//*  if  not pairing-found(sp) and optional(t)
//*             query-answer += {sp};
//* foreach  tp in result(t)
//*  if  not pairing-found(tp) and optional(s)
//*             query-answer += {tp};
```

Note that, for the conjoined query `+P:N:V, N:V`, the XML document

`<Q A="B"> <P N="V"> <N> V </N> </P> </Q>`

yields as the query answer the XML fragment

`<P N="V"> <N> V </N> </P>`.

4.3.1 Multiple Query Terms

We can modularize the above code as a predicate: $is\_interconnected(s\_term, Us\_tree, t\_term, Ut\_tree)$, where *is_interconnected* takes two terms and a pair of corresponding satisfying trees as input, and determines if the two subtrees are interconnected.

The query answer for a query containing only *required* terms can be computed as follows, where t_i's are the terms, u_i's are the corresponding XML trees, and $\wedge$ is the logical-and operation:

$$query\_answer(t_1) = results(t_1)$$

$$query\_answer(t_1, \ldots, t_n) =$$
$$\{\ (U_1, \ldots, U_n) \quad | \quad (U_1, \ldots, U_n) \in results(t_1) \times \ldots \times results(t_n)$$
$$\wedge \quad \wedge_{i=1}^{n} \ \wedge_{j=i+1}^{n} is\_interconnected(t_i, U_i, t_j, U_j)\ \}$$

The same code can be used to compute "tentative" query answers (not guaranteed to be maximal information-wise or without duplicates) for a query containing *optional* terms by making (a) $results(t_i)$ contain *null* if t_i is optional and (b) *is_interconnected* return *true* when one of the tree arguments is *null*. Tentative query answers can be compacted to obtain query answers (which contain only maximal information tree sequences without duplicates), by removing tree sequences that are included in the others.

4.4 Output

The XML fragment can be displayed by providing its coordinates in terms of the pair `(fileName, access-path-from-root)`, and also by extracting the text from the XML document using an "XPath processor". In fact, both these can be displayed side by side. It is unreasonable space-wise to explicitly store the entire "XPath to XML fragment" mapping in comparison to determining the necessary XML fragment at query-time. However, the overall performance can be improved if the query frequency is skewed and frequently occurring access paths to an XML fragment can be cached. Furthermore, the access path can itself be manipulated by the user to navigate to larger XML fragments that enclose the returned result.

The Lucene-based prototype was tested for correctness on a number of complex examples including those discussed earlier, and for general applicability and performance on large datasets such as SIGMOD Record (468 KB), Mondial (1.7 MB), and DBLP (337 MB). The exercise was useful in clarifying subtlities associated with the expressive power of the language, especially coherence in the context of heterogeneous XML documents, repeated term queries, etc, and obtain *relative* quantitative measures of space-time tradeoffs for indexing and search. For a practical deployment, it is necessary to develop efficient low-level indexing strategies from scratch, rather than build on Lucene. Given that indexing can be done offline (and even incrementally), this seems reasonable.

The time to determine the XPaths of the result fragments is only a fraction of the time taken to extract and display the corresponding XML file fragments. The time required to compute query answers depends on the form of the query and the dataset. For instance, in the context of SIGMOD dataset, the time required for a simple query returning small answer set such as `author::Philip` is very different from the time required for complex ones returning large answer set such as `article::`, `author:position:01` and `+author::`, `+title::`. The answer set display time is an order of magnitude larger. Note also that routine web search queries against the DBLP dataset return small answer sets.

For XHTML documents, the query language enables accessing title, searching for keywords, and using element/attribute to limit text around a keyword (such as itemized/enumerated list). However, lacking bracketing construct for sections in XHTML, the text between successive headings alone cannot be displayed. Similarly, lacking flexible extraction of all and only specified elements within another element, a table of contents from headings cannot be composed.

5 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 a such a way that the answers try to provide complete and relevant information, and is 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, or use the access path information of the hit to manually explore the neighborhood of the hit. That is, if the query response does not contain adequate information beyond the query terms, the user may have to intervene by expanding the query, or exploring the neighborhood of the XML subtree manually. To summarize, the proposed query language enables naive users to employ keywords to access XML document fragments, while allowing more advanced users, who know or discover XML Schema used, a means to express partial information in formulating more precise queries.

Another possible approach to overcome the solution granularity problem is to specify what kinds of subtrees (in terms of root labels) should be allowed in the answers. Given that there are several possible characterizations of what constitutes a keyword hit, based on the view of an XML document (data-centric vs document-centric), the query language can be enriched to enable the users to program-in their choices based on the context, as opposed to hard-coding a single definition of "keyword-containment-in-text". Coherence can be improved by using an ontology, or by incorporating transformations such as stemming, term expansion using synonyms, etc. However care must be taken to verify that such transformations do not cause semantically different elements to be confused, for instance, `author` and `authors` in SIGMOD dataset.

Design, analysis, implementation, and evaluation of XML retrieval systems is still in its infancy relative to the well-established field of information retrieval from text documents. In order to understand the practical benefits of the current work, it is still necessary to develop necessary metrics for evaluation, and carry out detailed, scientific experimentation with XML benchmarks along the lines proposed in Lalmas and Tombros [31].

Our implementation of indexing and search of XML documents is reasonably efficient and effective for experimentation. The use of required/optional query terms and the notions of interconnectedness allow us to prune the query results. In future, we plan to develop a simple yet robust relevance ranking strategy for heterogeneous document-centric XML fragments, to further organize the query results. For this approach to be viable on Web scale, to be incorporated in Web Search Engine, more research is required on low-level indexing strategies for efficient storage and retrieval of index information and of the corresponding XML fragments, and for relevance ranking of XML fragments.

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.searchtools.com/info/xml-resources.html`, Retrieved 10/2007.

[3] `http://www.w3.org/TR/xquery/`, Retrieved 10/2007.

[4] Thirunarayan, K. and Immaneni, T.: *Flexible Querying of XML Documents*, In: Proceedings of the 16th International Symposium on Methodologies for Intelligent Systems, Eds: F. Esposito and Z. Ras, Vol. 4203, LNAI/LNCS, September 2006, pp. 198-207.

[5] `http://www.cs.washington.edu/research/xmldatasets/`, Retrieved 10/2007.

[6] 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), 2003, pp. 45-56.

[7] Cohen, S., Kanza, Y., Kimelfeld B., and Sagiv, Y.: *Interconnection Semantics for Keyword Search in XML*, In: The 2005 ACM International Conference on Information and Knowledge Management (CIKM), Bermen (Germany), 2005, pp. 389-396.

[8] 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.

[9] Antoniou, G., and van Harmelen, F.: *A Semantic Web Primer*, The MIT Press, 2004.

[10] Li, Y., Yu, C. and Jagadish, H. V.: *Schema-Free XQuery*, In: Proceedings of the VLDB Conference, 2004, pp. 72-83.

[11] `http://www.ibiblio.org/xql/`, Retrieved 10/2007.

[12] Manning, C. D., Raghavan, P., and Schtze, H., Introduction to Information Retrieval, `http://informationretrieval.org/`, Retrieved 10/2007.

[13] Chamberlin, D., Robie, J., and Florescu, D.: *Quilt: An XML Query Language for Heterogeneous Data Sources*, In: Proceedings of WebDB 2000 Conference, Lecture Notes in Computer Science, vol. 1997, 2000, pp. 1-25.

[14] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D., *XML-QL: A Query Language for XML*, In: Proceedings of the 8th WWW Conference, 1998, `http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/`, Retrieved 10/2007.

[15] Berger, S., Bry, F., Schaffert, S., and Wieser, C.: *Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data*, In: Proceedings of 29th International Conference on Very Large Data Bases, 2003, pp. 1053-1056.

[16] Bailey, J., Bry, F., Furche, T., and Schaffert, S.: *Web and Semantic Web Query Languages: A Survey*, Reasoning Web, First International Summer School 2005, Eds: Eisinger, N., and Maluszynski, J., LNCS 3564, 2005, pp. 35-133.

[17] 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.

[18] Meyer, H., Bruder, I., Weber, G. and Heuer, A: *The Xircus Search Engine*, 2003.

[19] 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 ACM SIGIR Second Workshop on XML and IR, 2002.

[20] 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.

[21] 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.

[22] Grabs, T., and Schek, H.: *Generating Vector Spaces On-the-fly for Flexible XML Retrieval*, In: The ACM SIGIR Second Workshop on XML and IR, 2002.

[23] Catania, B., Maddalena, A., and Vakali, A.: *XML Document Indexes : A Classification*, In: IEEE Internet Computing, 2005, pp. 64-70.

[24] 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.

[25] Scott, M. L.: *Programming Language Pragmatics*, Morgan Kaufmann Publishers, 2nd Edition, 2006.

[26] http://xml.coverpages.org/elementsAndAttrs.html, Retrieved 10/2007.

[27] Fensel, D., Hendler, J., Lieberman, H., and Wahlster, W. (Eds.): *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, The MIT Press, 2003.

[28] http://www.semanticweb.org/, Retrieved 10/2007.

[29] http://www.w3.org/TR/xpath#data-model, Retrieved 10/2007.

[30] http://lucene.apache.org/java/docs/, Retrieved 10/2007.

[31] Lalmas, M., and Tombros, A.: *Evaluating XML retrieval effectiveness at INEX*, ACM SIGIR Forum, Vol. 41, Issue 1, June 2007, pp. 40-57.