Content-and-Structure Queries in an XML-based Information Retrieval System

Course Project, CS741: Non-traditional Databases
(Instructor: Prof. Frank W. Tompa)

Stefan Büttcher

School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

Abstract

Traditional information retrieval systems allow relevance ranking on the document level but do not offer any facilities to restrict the set of possibly relevant documents with respect to certain structural conditions on the sub-document level. XML database systems, on the other hand, support structural query languages but do not allow for content-based relevance ranking.

We present an extension to the MultiText framework that supports the combination of structural and content-related queries in the form of content-and-structure queries. Content scoring is realized by using the Okapi BM25 scoring function, while the structure of the query results can be defined using both XPath and MultiText's query language GCL. This hybrid approach allows to circumvent different shortcomings that are either specific to XPath or to GCL.

1 Introduction and Overview

In this paper, we present a new approach to the problem of content-and-structure queries, i.e. the combination of structure-less, content-based relevance ranking techniques with structural query languages such as XPath [Con99] or GCL [CCB94b].

In contrast to many other contributions that try to integrate relevance ranking into structural query languages, we start from the position of traditional information retrieval methods and extend them so that structural requirements can be imposed on the results of a query. The main reason for this approach is that for most applications the quality of the information retrieval part of the system is more critical than the ability to specify arbitrary structural constraints.

We take structural components from two sources: XPath and GCL. This is necessary because both languages have different shortcomings and the combination of both allows us to circumvent these shortcomings. We chose to use XPath instead of XQuery because, although the expressive power of XQuery is superior to that of XPath, the additional functions offered by XQuery, such as sorting and the construction of new elements, are unnecessary for content-and-structure queries.

In the following section, we give a brief summary of known full-text relevance search extensions to XML query languages and show their limits. In sections 3 and 4, we give a short introduction into both XPath and traditional relevance ranking. Section 5 describes the MultiText framework and a number of necessary extensions to the framework. In section 6, we present a new query language, based on XPath and GCL, that combines relevance ranking and structural queries. Section 7 gives a summary of our research and shows possible future work in this area.

This paper describes parts of the Wumpus information retrieval system, which is based on the MultiText indexing/search framework. The Wumpus system is still in development and is intended to become a multifunctional desktop search system. It can be downloaded from the author's website: http://stefan.buettcher.org/cs/wumpus/

2 Related Work

Many attempts have been made to unify structural (boolean) queries and content-based (relevance) queries recently.

Theobald and Weikum [TW00] presented a relevance ranking mechanism that is based on the XML-QL language. Fuhr and Grossjohann added relevance ranking to the XQL query language [FG00]. Fegaras [Feg04] proposed the integration of relevance scores into boolean predicates of XQuery. The relevance ranking is based on traditional TF*IDF components, but also takes into account new factors, such as element nesting level, stemming from the nature of XML documents.

The World Wide Web Consortium is currently developing a full-text search extension to XPath 2.0 and XQuery 1.0. Their progress can be watched on the W3C website [Con03].

Amer-Yahia et al. [AYBS04] developed TeXQuery a full-text search extension to XQuery that, according to their claims, fulfills all the requirements for a full-text search extension, as defined by the World Wide Web Consortium in their work towards a full-text search standard. Rys [Rys03] gives an overview of the integration of full-text search mechanisms into XML query languages.

All the above approaches have in common that they start from a structural query language – XPath or XQuery – and add full-text relevance search capabilities to the original language. However, we believe that for many applications the full power of traditional search techniques from IR is desirable, so rather than adding full-text extensions to structural query languages, we are proposing a method to add structural elements to a traditional relevance query.

Furthermore, the strict structure of XML might not be suited very well to deal with realistic information retrieval problems. Overlapping document regions and sequences of XML elements (retrieving only parts of elements as opposed to entire elements) cannot be dealt with in native XML/XQuery. However, a response such as "pages 4-8 *or* pages 7-10" makes perfect sense in information retrieval.

Moreover, when structural queries and relevance ranking are to be combined, structural elements can be introduced at two points:

1. the part of the text collection returned by the query processor (the query result) must meet certain structural conditions;
2. the query elements used to perform the relevance ranking may not only be individual terms but can contain complex structural queries themselves.

The first part is addressed appropriately by most XML query languages. However, although there are suggestions related to term proximity and the order in which query terms appear in a candidate document, none of the existing approaches covers the second part in an adequate way.

3 The XPath Query Language

Since our work is in parts based on XPath, we briefly present its main syntactic components. The XML Path Language (XPath), version 1.0, has been defined by the World Wide Web Consortium (W3C) as a W3C standard in 1999 [Con99]. A detailed description, as well as many examples, can be found on W3C's website.

The XPath query language allows the selection of XML elements within an XML document along so-called axes. Among the various axes defined by XPath are the `child` and `parent` axes, which can be used to select all child or parent nodes of a set of nodes within an XML document, respectively. The evaluation of an XPath expression starts with a set of currently selected nodes that only contains the root node of a document and successively applies all steps of the expression to the current node set, yielding a new set in every step.

Executing the query

doc("xml.xml")/child::*/child::name,

for instance, would result in all grandchildren of the root node of the document stored in the file `xml.xml` that have the type `name`. In addition to the "/" operator that is used to select XML nodes along certain axes, the "[]" operator can be used to perform a filtering operation on the current set of nodes. For example,

doc("xml.xml")/child::product[@price < 100]

selects all `product` nodes whose `price` attribute value is less than 100.

XPath does not support any notion of relevance. A node may either meet the query criteria or not. If two nodes both meet the criteria, it is not possible to say which one is better and should thus be presented to the user first.

4 Traditional Relevance Ranking

Traditional relevance ranking is usually performed on a collection of *documents*. With respect to a certain information need (e.g., "I want to know everything about King Henry VIII."), an information retrieval system has to answer the question: "What are the most relevant documents?", effectively ranking all documents in the collection in their order of relevance to the query.

The definition of what a document is may be chosen arbitrarily to meet the specific requirements of a certain task: a PDF file, a paragraph in a book, or a minute of spoken data could all be documents. However, once this decision has been made, it is final. After the collection has been indexed, only entire documents may be ranked.

Various ways to circumvent this restriction have been studied. Among these techniques are passage scoring methods that rank *passages within documents* instead of entire documents [KZ97], [CCT00], [CCL01]. Passage scoring is especially important when the documents in the text collection are very large (PDF files containing several hundred pages, for instance) and a system's response that a document is relevant is of questionable utility to the user because she does not know which parts of the document are the actually relevant parts.

We do not make use of any passage scoring technique but instead perform relevance ranking on a dynamically generated document collection, defined using a structural query language. The scoring method employed is the standard Okapi BM25 function introduced by Robertson et al. [RWJ+94] [RWB98].

In BM25, every query term T is assigned a term weight

$$w_T = \ln\left(\frac{|\mathcal{D}| - |\mathcal{D}_T| + 0.5}{|\mathcal{D}_T| + 0.5}\right), \quad (1)$$

where $\mathcal{D}$ is the set of all documents in the corpus, and $\mathcal{D}_T$ is the set of documents containing the term T. For a given query $\mathcal{Q} = \{T_1, ..., T_n\}$, the score of a document D is computed using the formula

$$\sum_{T \in \mathcal{Q}} w_T \cdot q_T \cdot \frac{d_T \cdot (1 + k_1)}{d_T + k_1 \cdot ((1 - b) + b \cdot \frac{len_D}{len_{avg}})}, \quad (2)$$

where d_T is the number of occurrences of the term T in the document D, len_D is the length of D, len_{avg} is the average document length in the corpus, and q_T is the query-specific relative weight of the term T. Usually, q_T equals the number of occurrences of T in the original query. In fact, experiments have shown that this is the optimal choice for many applications [JWR00].

The remaining parameters in formula 2 were chosen to be $k_1 = 1.2$ and $b = 0.75$ in our implementation. However, these values are highly dependent on the text collection, as they can be used to give higher preference on either long or short documents [Fuj04] [JWR00].

5 The MultiText Framework

The MultiText information retrieval system implements a framework for both structured text search and relevance ranking [CCB94a] [CC00]. However, there is almost no interaction between both parts of the framework. In particular, the set of documents in a text collection has to be defined at indexing time and cannot be selected differently from query to query.

MultiText indexes a text collection on the sub-document level, which allows for structured queries, such as term sequences ("to be or not to be") and term proximity queries ("MultiText" within 3 words from "Waterloo"). In the remainder of this section, we give a brief review of the structural operators defined in the MultiText framework and introduce two new operators that are necessary to remove undesirable limitations, caused by MultiText's *shortest substring* rule, and to process XPath queries.

5.1 Structural Queries in MultiText

The MultiText framework is based on the *shortest substring* paradigm of the generalized concordance lists proposed by Clarke et al. [CCB94a] [CCB94b]. An index extent $[S, E]$ is a result of a query Q iff

1. $[S, E]$ matches the query conditions and
2. there is no other extent $[S', E'] \subseteq [S, E]$ that is a result of Q.

This rule implies that query results may overlap but may not be nested. Obviously, this paradigm is not compatible with XPath queries, which may very well have nested results. In section 5.4, we show how to deal with this contradiction.

MultiText supports the combination of queries to construct new queries that impose structural requirements on the text passages matching the resulting query. A *basic query* consists of a single term T and asks for all occurrences of T within the text collection (i.e., all minimal text passages that contain the term). Starting with the basic queries, new queries can be defined in a recursive way. Let A and B be two MultiText queries and let $\mathcal{R}_A$ and $\mathcal{R}_B$ denote their result sets. Then:

- $(A \wedge B)$ asks for all minimal passages containing at least one $R_A \in \mathcal{R}_A$ and at least one $R_B \in \mathcal{R}_B$.
- $(A \vee B)$ asks for text ranges containing at least one $R_A \in \mathcal{R}_A$ or one $R_B \in \mathcal{R}_B$.
- $(A..B)$ asks for all minimal passages that start with an $R_A \in \mathcal{R}_A$ and end with an $R_B \in \mathcal{R}_B$.
- $(A > B)$ asks for all minimal passages $R_A \in \mathcal{R}_A$ that contain at least one $R_B \in \mathcal{R}_B$. Similarly, $(A \not> B)$ asks for for all text ranges $R_A \in \mathcal{R}_A$ that do not contain an $R_B \in \mathcal{R}_B$.
- $(A < B)$ asks for all minimal passages $R_A \in \mathcal{R}_A$ that are contained in at least one $R_B \in \mathcal{R}_B$. $(A \not< B)$ is analogous to $(A \not> B)$.

5.2 Combining Structural Queries and Relevance Ranking

As mentioned in section 2, structural components may be introduced at two different points:

1. two impose certain requirements on the possible results;
2. two define the query elements that are used to rank the results.

```
<person>
  <name>Henry VIII</name>
  <title>King</name>
  <parents>
    <person>
      <name>Henry VII</name>
      <title>King</name>
    </person>
    <person>
      <name>Elizabeth</name>
      <title>Princess of York</name>
    </person>
  </parents>
</person>
```

Figure 1: Nested XML elements of the same type. The `person` element describing Henry VIII. is inaccessible to GCL.

The MultiText framework already supports relevance queries of the second type: When BM25 is used to score documents from a text collection, the set of query terms is not restricted to actual terms, but every $T \in \mathcal{Q}$ in formula 2 can in fact be an arbitrarily complex query, constructed using the rules described in section 5.1.

Support for queries of type 1 is not difficult to implement and has been added to the Wumpus system. In MultiText, entire documents are scored. The set of all documents in the text collection can be expressed by the query

“`<document>`” .. “`</document>`”

in *GCL*, the MultiText query language. The `<document>` tags may either already have been present in the document or may have been added as virtual text elements at indexing time to define document boundaries. So, scoring documents is in fact only a special case of scoring the results of a structural query and we may modify the query language so that it supports arbitrary GCL-compatible extents to be scored.

This extension to the MultiText relevance ranking capabilities is sufficient for most realistic scenarios, as GCL can be used to express most of the constraints that can be expressed in XPath. Three major problems remain:

- GCL is incompatible with certain aspects of XML, in particular with nestings of nodes of the same type, as shown in Figure 1, as it
  - cannot distinguish between different nesting levels and

```
<person name="Henry VIII" title="King">
  <parents>
    <person name="Henry VII" title="King">
    <person name="Elizabeth" title="Princess of York">
  <parents>
</person>
<newpage/>
<person name="Peter" title="Czar">
  <parents>
    ...
  </parents>
</person>                                          (a)
```

```
<person><attr!name>Henry VIII</attr!name>
  <attr!title>King</attr!title>
  <parents>
    <person><attr!name>Henry VII</attr!name>
      <attr!title>King</attr!title></person>
    <person><attr!name>Elizabeth</attr!name>
      <attr!title>Princess of York</attr!title></person>
  </parents>
</person>
<newpage></newpage>
<person><attr!name>Peter</attr!name>
...                                                      (b)
```

Figure 2: XML data (a) before and (b) after preprocessing.

  – cannot return elements that contain other elements of the same type.

- It is impossible two specify ranking criteria that involve two different relevance scoring processes, e.g. "Return all books that contain a chapter talking about King Henry VIII. and another chapter that covers Queen Victoria.".

In the following two sections, we show how to solve the first two problems by extending GCL and adding full XPath support to the MultiText framework. A solution to the third problem (multiple scoring processes) is presented in section 6 as part of a new, unified query language.

5.3 Element Sequences

In section 2, we described the inability to deal with element sequences, such as "pages 4-8", as one major shortcoming of XPath-based solutions to the problem of content-and-structure queries. Unfortunately, the same holds for GCL, which – due to the nature of the underlying shortest substring paradigm – cannot be used to retrieve sequences of adjacent XML elements.

It is not difficult, though, to add support for this kind of query by introducing a new operator:

"`<page>`" $../N$ "`</page>`"

selects sequences of up to N `page` elements as long as they are immediately adjacent. That is, the new query

"`<page>`" $../2$ "`</page>`"

can be translated to two traditional GCL expressions:

"`<page>`" .. "`</page>`"

and

"`<page>`" .. "`</page><page>`" .. "`</page>`",

which are then evaluated one after the other.

It has to be pointed out, however, that the introduction of the new "$../N$" operator is dangerous and might break the framework because the implementations of many of the other operators defined in GCL rely on the fact that for two extents $[S_1, E_1]$ and $[S_2, E_2]$ that are results of the same GCL expression, we have:

$$S_1 \leq S_2 \;\Leftrightarrow\; E_1 \leq E_2.$$

This is not true for "$../N$", as we may have nested extents within the same result set. Therefore, we have to restrict the use of the new operator to the topmost level of a GCL expression, i.e. no other GCL operator may use the results of the new operator.

5.4 Adding XPath Support

The structural queries that can be constructed using the rules from section 5.1 cannot be used to find the *parent* or the *children* of a given XML element. Within the MultiText framework, it is possible to find elements that contain a certain element or that are contained within a element, i.e. to walk along the `ancestor` or `descendant` axis:

("`<containee>`" .. "`</containee>`")
    < ("`<container>`" .. "`</container>`")

However, it is not possible to tell if a `descendant` of a certain element is a `child` or not, since

Sequence #	1	2	3	4	5	6	7	8	9	10
Input token	`<person>`	`<attr!name>`	`Henry`	`VIII`	`</attr!name>`	`<attr!title>`	`King`	`</attr!title>`	`<parents>`	`<person>`
Virtual tokens	`<level!1>`	`<level!2>` `<attr!>`			`</level!2>` `</attr!>`	`<level!2>` `<attr!>`		`</level!2>` `</attr!>`	`<level!2>`	`<level!3>`

Figure 3: Augmenting the input stream with special tokens for XPath support.

this decision requires additional knowledge about other XML tags within the container element. The same holds for the `parent`, `attribute`, `preceding-sibling` and `following-sibling` axes, as they all require information about the nesting level of a particular node.

In order to include these types of structural queries into our framework, we have to implement several extensions, both at indexing and at query time.

5.4.1 XPath at Indexing Time

First of all, we assume that the input XML document comes in a slightly modified form: XML attributes are not allowed but have to be encoded as ordinary XML elements instead. To avoid ambiguity, the resulting XML tags have a special form, as shown in Figure 2. Furthermore, empty elements may not occur in their abbreviated `<empty/>` form, but have to appear as `<empty></empty>`. Both requirements can easily be met by running a preprocessor on the input document before performing the actual indexing.

When the actual indexing takes place, the indexing algorithm uses MultiText's ability to index different tokens at the same position and introduces special symbols that can later be used to determine the nesting level of a given node within an XML document. More precisely, the indexing algorithm keeps track of the current nesting level and inserts a `<level!`K`>` / `</level!`K`>` pair for every pair of XML tags in the input stream, where K is the nesting level of the current token. In addition, it inserts an `<attr!>` / `</attr!>` pair whenever it encounters an attribute node.

This process is shown in Figure 3 and helps us find children, parents and attributes at query time.

5.4.2 XPath at Query Time

After a text collection has been indexed, XPath queries may be executed to obtain subsets of the entire collection. We will show how XPath expressions can be evaluated within the MultiText framework. For simplicity, we introduce a new operator:

The query $(A \cdot B)$ asks for all passages $P \in \mathcal{R}_A \cap \mathcal{R}_B$, where $\mathcal{R}_X$ denotes the result set of a query X.

The different XPath axes may now be implemented within the extended MultiText framework by using the information stored inside the `<level!`N`>` lists. Although in general, we might have self-nested element types (such as `name` in Figure 1), this is not the case any more for a fixed nesting level.

Therefore, we may implement the `ancestor` axis using the following algorithm:

```
ancestor:    // input: element $[S, E]$ on level $l$
  $R := \emptyset$
  for $i := l - 1$ downto 1 do
    $R := R \cup$
      $\mathcal{R}((``\texttt{<level!}i\texttt{>}"..``\texttt{</level!}i\texttt{>}") > [S, E])$
  return $R$
```

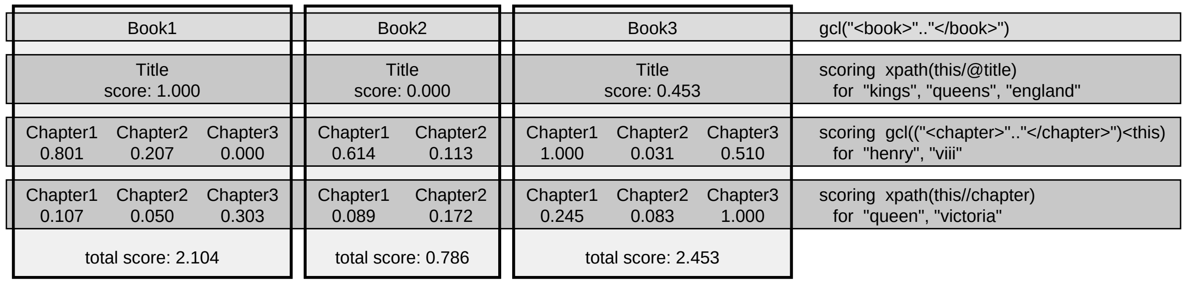
Implementations for the other XPath axes look similar and always process the result candidates in a level-by-level fashion. Typed XPath steps, such as `child::parents` are realized by accessing the results of the expression

$$((``\texttt{<parents>}" \cdot ``\texttt{<level!}i\texttt{>}")\ ..\ (``\texttt{</parents>}" \cdot ``\texttt{</level!}i\texttt{>}"))$$

instead of (“`<level!`i`>`”..“`</level!`i`>`”).

Similarly, the `attribute` axis is realized by combining `<attr!>` and `<level!`$(l+1)$`>` (where l is the nesting level of the current element) in the same way as above, i.e. using the $\cdot$ operator.

```
XGCL Query:
  @cas-rank  gcl("<book>".."</book>)  by                                              // selects all book elements for ranking
     scoring  xpath(this/@title)  for  "kings", "queens", "england"  using  BM25      // scores title attributes of all books  (selection: XPath; scoring: BM25)
     scoring  gcl(("<chapter>".."</chapter>")<this)  for  "henry", "viii"  using  QAP // scores all chapters of all books  (selection: GCL; scoring: QAP)
     scoring  xpath(this//chapter)  for  "queen", "victoria"  using  BM25              // scores all chapters of all books  (selection: XPath; scoring: BM25)
```



Book1
Book2
Book3
gcl("<book>".."</book>")
Title score: 1.000
Title score: 0.000
Title score: 0.453
scoring xpath(this/@title) for "kings", "queens", "england"
Chapter1 0.801 Chapter2 0.207 Chapter3 0.000
Chapter1 0.614 Chapter2 0.113
Chapter1 1.000 Chapter2 0.031 Chapter3 0.510
scoring gcl(("<chapter>".."</chapter>")<this) for "henry", "viii"
Chapter1 0.107 Chapter2 0.050 Chapter3 0.303
Chapter1 0.089 Chapter2 0.172
Chapter1 0.245 Chapter2 0.083 Chapter3 1.000
scoring xpath(this//chapter) for "queen", "victoria"
total score: 2.104
total score: 0.786
total score: 2.453

Figure 4: Example XGCL query with three different relevance queries. Scores are normalized and the maximum from every scoring process is added to the final score of the target.

6 A Unified Content-and-Structure Query Language

This section describes a new, unified query language that supports content-and-structure queries involving structural components from both GCL and XPath, called *XGCL*.

The general syntax of content-and-structure queries in XGCL is of the following form:

@cas-rank *Target* by
  scoring *Element*$_1$ for *Query*$_1$ using *Method*$_1$
  scoring *Element*$_2$ for *Query*$_2$ using *Method*$_2$
  .....

The "@cas-rank" command is used to distinguish the query from other query types defined in the MultiText framework. A cas-rank query has the following components:

Target is an arbitrary XPath or GCL query (including new new operator for element sequence queries) that defines the passages within a text collection that are to be ranked and returned when the query is processed.

Element is either an XPath or a GCL expression that is used to select a set of text passages for a given target T. These passages are selected relative to T and are assigned a relevance score for the given *Query*.

Query is an ordinary relevance query, a sequence of GCL expressions.

Method is the scoring method to be used. Several scoring methods are supported by MultiText, including the Okapi BM25 function.

Every query can contain arbitrarily many (but at least one) *Element*/*Query*/*Method* triples. The final score of a passage $T \in \mathcal{R}_{Target}$ is computed by calculating the normalized sum of all scoring results.

This process is shown by the example in Figure 4. First, all `book` elements in the collection are selected. Then, for every `book` the `title` attribute is selected and scored using BM25 and global term statistics from all `title` attributes selected (this means that, in information retrieval terminology, the `title` attributes from all `book`s form the document collection for this scoring process).

As a next step, all chapters from all books are taken, considered a document collection again and scored using MultiText's QAP algorithm. The third scoring process takes again all chapters from all books and scores them using BM25. The scores from all three scoring processes are then combined to compute the final score of the target elements, the books in the text collection.

6.1 Multiple Evidence Combination

When a *Target* set is ranked it may happen that for a given target T there is more than one *Element* in the same scoring process. In the example shown in Figure 4, for instance, every `book` contains several `chapter` elements. Adding the relevance scores of all chapters to the target's final score is a bad idea because it favors irrelevant books with many chapters over relevant books with few chapters.

On the other hand, if we assume a correlation between relevance scores and the probability of being relevant to a given topic, it makes sense to give a higher score to a book that contains two relevant chapters than to a book that contains only one relevant chapter – supposing the number of chapters in both books are similar. However, at this point it is not clear at all what this combination of multiple sources of evidence could look like.

Our solution is to take the score of the highest-scoring element and to ignore the rest. The same strategy is used by MultiText's QAP passage scoring algorithm [CCL01]. When it is used to score documents by relevant passages within documents, it assigns the score of the highest-scoring passage to the whole document. Since both tasks are very similar and QAP performs reasonably well, we chose to adopt this method.

7 Conclusions and Future Work

We have presented a new query language, XGCL, that combines two structural query languages, XPath and GCL, with traditional relevance ranking. XGCL can be used to express a great number of content-and-structure queries. Our approach differs from most approaches to the content-and-structure problem in so far as we start from the IR point of view instead of the XPath/XQuery position and try to insert structural components into a traditional retrieval system.

We have extended the GCL query language so that it supports sequences of XML elements, which contradicts the original shortest substring paradigm of the MultiText framework, and presented a way to integrate XPath into that framework. The combination of XPath and GCL as query languages for structural constraints allows to express a greater number of constraints than each language alone would be able to specify.

Future work will primarily focus on the impact that content-and-structure queries can have on the effectiveness of ordinary document retrieval systems, especially in the domain of genomics-related text databases such as the Medline database of biomedical publications [oM04], which offers a rich variety of structural information due to various annotations.

We will also investigate different techniques of multiple evidence combination, as discusses in section 6.1 because this is one of the major unsolved problems in the domain of content-and-structure queries.

Furthermore, we will study possibilities to score sequences of non-successional elements and display the results in a uniform, user-friendly way.

References

[AYBS04] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. TeXQuery: A Full-Text Search Extension to XQuery. In *Proceedings of the 13th International Conference on World Wide Web*, pages 583–594. ACM Press, 2004.

[CC00] Charles L. A. Clarke and Gordon V. Cormack. Shortest-Substring Retrieval and Ranking. *ACM Trans. Inf. Syst.*, 18(1):44–78, 2000.

[CCB94a] Charles L. A. Clarke, Gordon V. Cormack, and Forbes J. Burkowski. An Algebra for Structured Text Search and a Framework for its Implementation. Technical report, University of Waterloo, August 1994.

[CCB94b] Charles L. A. Clarke, Gordon V. Cormack, and Forbes J. Burkowski. Schema-Independent Retrieval from Heterogeneous Structured Text. Technical report, University of Waterloo, November 1994.

[CCL01] Charles L. A. Clarke, Gordon V. Cormack, and Thomas R. Lynam. Exploiting Redundancy in Question Answering. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 258–276, November 2001.

[CCT00] Charles L. A. Clarke, Gordon V. Cormack, and Elizabeth A. Tudhope. Relevance Ranking for One to Three Term Queries. *Information Processing and Management*, 36(2):291–311, 2000.

[Con99] The World Wide Web Consortium. XML Path Language (XPath) Version 1.0, November 1999. W3C Recommendation, http://www.w3.org/TR/xpath.

[Con03] The World Wide Web Consortium. XQuery and XPath Full-Text Requirements, May 2003. W3C Working Draft, http://www.w3.org/TR/xquery-full-text-requirements/.

[Feg04] Leonidas Fegaras. XQuery Processing with Relevance Ranking. In *Proceedings of the Second International XML Database Symposium (XSym 2004)*, pages 51–65, 2004.

[FG00] N. Fuhr and K. Großjohann. XIRQL: An extension of XQL for Information Retrieval, July 2000. In ACM SIGIR Workshop On XML and Information Retrieval, Athens, Greece, 2000.

[Fuj04] S. Fujita. Revisiting Again Document Length Hypotheses. TREC 2004 Genomics Track Experiments at Patolis. In *Proceedings of the 13th Text REtrieval Conference (TREC 2004)*, November 2004.

[JWR00] Karen Spärck Jones, Steve Walker, and Stephen E. Robertson. A Probabilistic Model of Information Retrieval: Development and Comparative Experiments – Part 2. *Information Processing and Management*, 36(6):779–808, 2000.

[KZ97] Marcin Kaszkiel and Justin Zobel. Passage Retrieval Revisited. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 178–185. ACM Press, 1997.

[oM04] U.S. National Library of Medicine. PubMed, 2004. http://www.ncbi.nlm.nih.gov/pubmed.

[RWB98] S. Robertson, S. Walker, and M. Beaulieu. Okapi at TREC-7. In *Proceedings of the Seventh Text REtrieval Conference (TREC 1998)*, November 1998.

[RWJ$^+$94] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of the Third Text REtrieval Conference (TREC 1994)*, November 1994.

[Rys03] M. Rys. Full-Text Search with XQuery: A Status Report., 2003. In Intelligent Search on XML, Springer-Verlag, 2003.

[TW00] A. Theobald and G. Weikum. Adding Relevance to XML. In *Proceedings of the 3rd International Workshop on the Web and Databases (WebDB 2000)*, pages 35–40, 2000.