Structural Feedback for Keyword-Based XML Retrieval

Ralf Schenkel and Martin Theobald

Max-Planck-Institut für Informatik, Saarbrücken, Germany
{schenkel, mtb}@mpi-inf.mpg.de

Abstract. Keyword-based queries are an important means to retrieve information from XML collections with unknown or complex schemas. Relevance Feedback integrates relevance information provided by a user to enhance retrieval quality. For keyword-based XML queries, feedback engines usually generate an expanded keyword query from the content of elements marked as relevant or nonrelevant. This approach that is inspired by text-based IR completely ignores the semistructured nature of XML. This paper makes the important step from pure content-based to structural feedback. It presents a framework that expands a keyword query into a full-fledged content-and-structure query. Extensive experiments with the established INEX benchmark and our TopX search engine show the feasibility of our approach.

1 Introduction

1.1 Motivation

XML has seen increasing importance recently to represent large amounts of semistructured or textual information in digital libraries, intranets, or the Web, so information retrieval on XML data is growing more and more important. XML search engines employ the ranked retrieval paradigm for producing relevance-ordered result lists rather than merely using XPath or XQuery for Boolean retrieval. An important subset of XML search engines uses keyword-based queries [2, 8, 31], which is especially important for collections of documents with unknown or highly heterogeneous schemas. However, simple keyword queries cannot exploit the often rich annotations available in XML, so the results of an initial query are often not very satisfying.

Relevance Feedback is an important way to enhance retrieval quality by integrating relevance information provided by a user. In XML retrieval, existing feedback engines usually generate an expanded keyword query from the content of elements marked as relevant or nonrelevant. This approach that is inspired by text-based IR completely ignores the semistructured nature of XML. This paper makes the important step from content-based to structural feedback. We extend the well-established feedback approach by Robertson and Sparck-Jones [21] to expand a keyword-based query into a possibly complex content-and-structure query that specifies new constraints on the structure of results, in addition to "standard" content-based query expansion. The resulting expanded query has

M. Lalmas et al. (Eds.): ECIR 2006, LNCS 3936, pp. 326–337, 2006.

weighted structural and content constraints and can be fed into a full-fledged XML search engine like our own TopX engine [25, 26].

As an example, consider the keyword query (query 204 from the INEX benchmark [12]) `moldovan semantic networks`. Without additional knowledge, it is unclear that the term "moldovan" actually refers to the author of a paper about semantic networks. Additionally, it is very unlikely that the author name and the terms "semantic network" occur in the same element, as author names are usually mentioned in different places than the content of articles. A query with constraints on both content and structure would probably yield a lot more relevant results, but it is impossible to formulate a query like the following without knowledge of the underlying schema:

```
//sec[about(.//au, ``moldovan") and about(., ``semantic networks")]
```

The techniques presented in this paper automatically construct a content-and-structure query from a keyword-based query, exploiting relevance feedback by a user. This paper makes the following important contributions: (1) It presents a formal framework to integrate different classes of query expansions, beyond content-based feedback, into XML retrieval, (2) it presents the implementation of four expansion classes, and (3) it evaluates the performance of the techniques, showing a huge gain in effectiveness with the established INEX benchmark [12].

The primary goal of this paper is to show that structural feedback helps to enhance result quality. The paper does not claim to present the ultimately best implementation of structural feedback, but opens a whole design space and presents variants that give reasonably good results.

1.2 Related Work

Relevance feedback has already been considered for document retrieval for a long time, starting with Rocchio's query expansion algorithm [22]. Ruthven and Lalmas [23] give an extensive overview about relevance feedback for unstructured data, including the assessment of relevance feedback algorithms.

Relevance feedback in XML IR is not yet that popular. Of the few papers that have considered it, most concentrate on query expansion based on the content of elements with known relevance [5, 14, 24, 30]. Some of these focus on blind ("pseudo") feedback, others on feedback provided by users. Pan et al. [16] apply user feedback to recompute similarities in the ontology used for query evaluation.

Even fewer papers have considered structural query expansion [9, 10, 15, 18, 19]. Mihajlovič et al. [15, 18, 19] proposed deriving the relevance of an element from its tag name, but could not show any significant gain in retrieval effectiveness. Additionally, they considered hand-tuned structural features specific for the INEX benchmark (e.g., the name of the journal to which an element's document belongs), but again without a significant positive effect. In contrast, we propose a general approach for feedback that can be applied with the INEX data, but does not rely on any INEX-specific things.

Hlaoua and Boughanem [9] consider common prefixes of relevant element's paths as additional query constraints, but don't provide any experimental evaluation of their approach.

The work of Hsu et al. [10] is closest to our approach. They use blind feedback to expand a keyword-based query with structural constraints derived from a neighborhood of elements that contain the keywords in the original query. Our approach considers the whole document instead of only a fragment, can generate constraints with negative weight, and integrates also content-based constraints.

2 Formal Model and Notation

2.1 Data Model

We consider a fixed corpus of D XML documents with E elements. For such a document d, $E(d)$ denotes the set of elements of the document; for an element e, $T(e)$ denotes its tag name and $d(e)$ the document to which it belongs.

The *content* $c(e)$ of an element e is the set of all terms (after stopword removal and optional stemming) in the textual content of the element itself and all its descendants. (Note that XML retrieval engines usually use this content model, while boolean languages like XPath or Xquery typically only use the content of the element itself.) For each term t and element e, we maintain a weight $w_e(t)$. This can be a binary weight ($w_e(t) = 1$ if the term occurs in e's content and 0 otherwise), a tf-idf style [13] or a BM25-based [1, 29] weight that captures the importance of t in e's content. The *content* $c(d)$ of a document d is defined as the content $c(r)$ of its root element r.

We maintain a number of statistics about the occurrence of terms in documents and elements: The *document frequency* df_t of a term t is the number of documents in which the term appears in the content. Analogously, the *element frequency* ef_t of a term t is the number of elements in which the term appears in the content.

2.2 Queries and Relevance of Results

We use an extended version of INEX's query language NEXI [27]. NEXI basically corresponds to XPath restricted to the `descendants-or-self` and `self` axis and extended by an IR-style `about` predicate to specify conditions that relevant elements should fulfil. The wildcard symbol '*' matches any tag and can be used to formulate keyword queries in NEXI. We extend NEXI with additional weights for each content constraint. A typical extended NEXI query looks like the following:

```
//article[about(.,``0.8*XML")//*[about(//p,``0.4*IR -0.2*index")]
```

The result granularity of such a query are elements. The relevance of an element with respect to a query relevance model is measured binarily, i.e., an element is either relevant or nonrelevant.

3 Expanding Keyword-Based Queries

We studied the content-and-structure queries from INEX to find patterns that are regularily used in such queries to describe relevant elements, in addition to

content conditions on the result element. A canonical example for such a query is the following:

```
//article[about(.,‘‘XML") and about(//bib,‘‘numbering")]//sec[about(.,IR)
and about(//par,index)]
```

that is a content-and-structure version of the simpler keyword query “XML IR index numbering”. In contrast to the keyword query, the structured query specifies a tag (or, more generally, a set of tags) that relevant elements should have (“I am interested in sections about ’IR’”). Additionally, this query contains constraints on the content of descendants of relevant elements (“sections with a paragraph about ’index’”), the content of ancestors (“sections in articles about ’XML’”), and the content of descendants of ancestors (“sections in articles that cite a paper about ’numbering’”).

As such a content-and-structure query specifies much more precisely the conditions that relevant elements must satisfy, we can expect that a search engine will return more relevant results for a content-and-structure query than for the keyword query, provided that the content-and-structure query correctly captures the same information need as the keyword query.

Our feedback framework aims at generating a content-and-structure query from a keyword query, exploiting relevance feedback provided by a user for some results of the keyword query. This section presents the core elements of our feedback framework. We start with the formal model for relevance feedback, discuss possible expansions of a query, show how the possibly best expansions can be selected, and how an expanded query is generated.

3.1 Feedback Model

We consider a keyword query $q = \{q_1, \ldots, q_p\}$ with a set $E = \{e_1, \ldots, e_l\}$ of results with known relevance, i.e., elements for which a user has assigned an exhaustivness value $e(e)$ and a specificity value $s(e)$. We say that an element e is *relevant* for the query if both $e(e)$ and $s(e)$ are maximal (i.e., have the value 3 in INEX), yielding a set $E^+ = \{e_1^+, \ldots, e_R^+\}$ of relevant elements and a set $E^- = \{e_1^-, \ldots, e_N^-\}$ of nonrelevant elements.

Note that even though this paper considers only binary relevance, it is possible to extend the mechanism presented here to approaches where relevance is measured with a probability-like number between 0 and 1, for example by representing E^+ and E^- as probabilistic sets.

3.2 Candidates for Query Expansion

Following the discussion in the beginning of this section, we derive the following classes of candidates for query expansion from an element with known relevance:

- all terms of the element’s content together with their score (C candidates),
- all tag-term pairs of descendants of the element in its document, together with their score (D candidates),

- all tag-term pairs of ancestors of the element in its document, together with their score (A candidates), and
- all tag-term pairs of descendants of ancestors of the element in its document, together with their score and the ancestor's tag (AD candidates).

The system can be extended with additional classes of candidates like tags, twigs, or paths, which is subject to future work.

Formally, we consider for an element e

- the set $C(e) = \{t \in c(e)\}$ of its C candidates, i.e., the terms in its content,
- the sets $A(e)$ of A and $D(e)$ of D candidates, i.e., all triples $(a, T(e'), t)$ where e' is an element in e's document that is an ancestor/a descendant of e, a is the relative position of e and e' (ancestor or descendant), $T(e')$ is the tag name of e', and t is a term in the content of e', and
- the set $AD(e)$ consists of its AD candidates, i.e., triples $(T(e'), T(e''), t)$ where e' is an ancestor of e, e'' is a descendant of e', and $t \in c(e'')$.

The candidate set $\Gamma(e) := C(e) \cup A(e) \cup D(e) \cup AD(e)$ is the set of all candidates for element e. We extend the notion of frequencies from terms to candidates as follows: The element frequency $ef(c)$ of a candidate c is the number of elements e for which $c \in \Gamma(e)$, and its document frequency $df(c)$ is the number of documents that contain at least one element e with $c \in \Gamma(e)$.

3.3 Weights for Expansion Candidates

To weight the different candidates, we apply a straight-forward extension of the well-known Robertson-Sparck-Jones weight [21] to element-level retrieval in XML. The weight $w^+_{RSJ}(c)$ of a candidate c is computed analogously to Robertson and Sparck-Jones with binary weights:

$$w^+_{RSJ}(c) = \log \frac{r_c + 0.5}{R - r_c + 0.5} + \log \frac{E - ef_c - R + r_c + 0.5}{ef_c - r_c + 0.5}$$

Here, for a candidate c, r_c denotes the number of relevant elements which contain the candidate c in their candidate set, R denotes the number of relevant elements, and E the number of elements in the collection. As the RSJ weights do not yield useful values if there are no relevant results at all, we additionally compute another weight for each term that captures its importance within the nonrelevant results:

$$w^-_{RSJ}(c) = \log \frac{n_c + 0.5}{N - n_c + 0.5} + \log \frac{E - ef_c - N + n_c + 0.5}{ef_c - n_c + 0.5}$$

where n_c denotes the number of nonrelevant elements which contain the candidate c in their candidate set and N denotes the number of nonrelevant elements. The weight $w_{RSJ}(c)$ of the candidate is then $w^+_{RSJ}(c)$ if $R > 0$ and $-w^-_{RSJ}(c)$ otherwise, so we consider our new weight only if there are no relevant results at all.

3.4 Selecting Expansion Candidates

The set of all possible expansion candidates is usually very large and contains many unimportant and misleading expansions, so we have to select the best b of them for generating the expanded query. This problem already exists for content-based expansion of keyword queries, and several possible weights have been proposed in the literature that go beyond naively ordering terms by their weight, like prefering terms that occur in many relevant elements [6], Porter's Algorithm [17], and the expected mutual information measure EMIM [28]. We use the so-called Robertson Selection Values (RSV) proposed by Robertson [20]. For a candidate c, its RSV has the form $RSV(c) = w_{RSJ}(c) \cdot (p - q)$, where $p = r_c/R$ is the estimated probability of the candidate occurring in a relevant element's candidate set and q is the probability that it occurs in a nonrelevant element's set. Unlike Robertson who assumed q to be negligible, we estimate $q = n_c/N$ as there are usually a lot more nonrelevant than relevant elements in the top results. We ignore candidates that occur only within the documents of elements with known relevance as they have no potential to generate more relevant results outside these documents. We order the union of the remaining candidates by their RSV and choose the top b of them, where b is a configuration parameter of the system. To be able to generate a valid NEXI query in the next step, we have to limit the A and AD candidates chosen to contain the same ancestor tag.

3.5 Generating an Expanded Query

Using the top-b candidates, we generate a content-and-structure query from the original keyword query. This expansion is actually straight-forward, and the generated query has the following general structure:

```
//ancestor-tag[A+AD constraints]//*[keywords+C+D constraints]
```

As an example, if the original query was 'XML' and we selected the A candidate (anc,article,'IR'), the AD candidate (article,bib,'index') and the D candidate (desc,p,'index'), the expanded query would be

```
//article[about(.,'IR') and about(//bib,'index')]//*[about(.,'XML')
and about(//p,'index')]
```

Each of the expansions is weighted, where the weight is the candidate's RSJ weight adjusted by a factor that depends on the candidate's class. C and D candidates help finding new relevant results, so they should get a high weight; we allow for C and D conditions at most the weight of all original keywords (to make sure that the new constraints don't dominate the query's results). As an example, for a query with four keywords and six C and D expansions, the factor for each expansion is $\frac{4}{6}$. On the other hand, A and AD conditions are satisfied by most – if not all – elements of a document, so they generate a huge amount of new result elements, most of which will be nonrelevant. Their weight should therefore be smaller than the weight of C and D conditions. We choose a fraction β of the accumulated weight of existing keyword conditions, with $\beta = 0.2$ in our experiments.

4 Architecture and Implementation

Figure 1 shows the high-level architecture of our extensible feedback framework. Each candidate class is implemented with a standard interface that allows a simple integration of new classes.

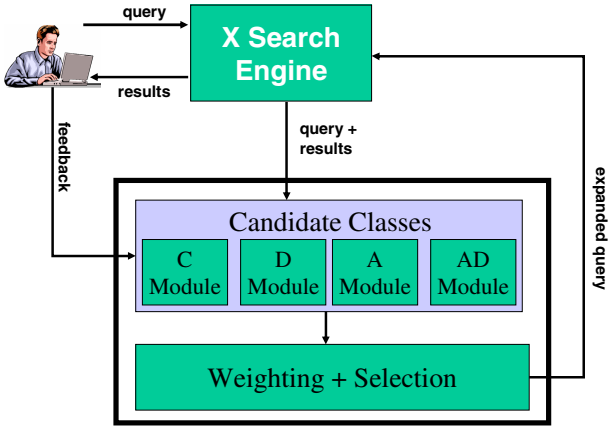

Fig. 1. Architecture of the feedback engine

The initial results of a query are presented to the user who gives positive or negative feedback to some of the results. This feedback is sent together with the query and its initial results to the feedback framework which forwards them to the available candidate classes. For each class, all possible candidates are computed for the results with known relevance. Out of all those candidates, the best b candidates are selected and used to build the expanded query that is sent back to the engine which evaluates it and presents the results to the user. The user may now again submit feedback for some of the new results, triggering another feedback cycle. To facilitate an automatic assessment of the feedback approach, the system can additionally import queries and results from INEX (see Section 5.1) and automatically generate feedback for the top-k results, using the existing INEX assessments.

We have implemented the framework in Java with the candidate classes shown in Section 3.2 and the TopX search engine. Our implementation requires that important information about elements is precomputed: unique identifiers for the element (`eid`) and its document (`did`), its pre and post order to facilitate the evaluation of structural query conditions like the XPath axes [7] or any other similar information, its tag, and its terms (after stemming and stopword removal), together with their score. This information is stored in a database table with schema (`did,eid,pre,post, tag,term,score`) that contains one tuple for each distinct term of an element. We can reuse an existing inverted file of an XML search engine that typically captures similar information, possibly after some transformation. On the database side, we provide indexes on (`eid,did`)

to efficiently find $d(e)$ for an element e and on `(did)` to efficiently collect all elements of a document. Inverse element and document frequencies of the different candidate classes are precomputed (e.g., while initially parsing the collection) and stored in database tables, too.

For each element with known relevance, the implementation of the feedback engine first loads the complete content of the element's document and computes all possible expansions in memory with the ief for the candidate loaded from the database. The additional constraint on the candidate's document frequency is then only checked for the best candidates that are considered for expanding the query.

5 Experimental Results

5.1 Settings

We use the *INEX* [12] benchmark for XML IR that provides a set of 12,107 XML documents (scientific articles from IEEE CS), a set of NEXI queries together with a manually assessed set of results for each query, and an evaluation environment to assess the effectiveness of XML search engines. INEX provides a Relevance Feedback Track [11, 4] that aims at assessing the quality of different feedback approaches. As this paper concentrates on keyword-based queries (*content-only topics* or *CO* for short in INEX), we used the set of 52 CO queries from the 2003 and 2004 evaluation rounds with relevant results together with the *strict* quantization mode, i.e., an element was considered as relevant if it exactly answers the query.

A *run* is the result of the evaluation of all topics with a search engine, it consists of 1500 results for each topic that are ranked by expected relevance. The measure of effectiveness is the *mean average precision* (MAP) of a run. Here, we first compute for each topic the average precision over 100 recall points (0.01 to 1.00) and then take the macro average over these topic-wise averages. Note that absolute MAP values are quite low for INEX (with 0.152 being the best MAP value of any participating engine in 2004). This reflects the fact that XML retrieval is inherently more complex than document retrieval as not only relevant documents, but also relevant elements within these documents have to be identified. In addition to MAP values, we also measured precision at different positions for each run.

To assess the quality of feedback algorithms, we use the residual collection technique [23] that is also used in the INEX 2004 Relevance Feedback Track. In this technique, all XML elements that are used by the feedback algorithm, i.e., those whose relevance is known to the algorithm, must be removed from the collection before evaluation of the results with feedback takes place. This includes all k elements "seen" or used in the feedback process regardless of their relevance. Under INEX guidelines, this means not only each element used or observed in the RF process but also all descendants of that element must be removed from the collection (i.e., the residual collection, against which the feedback query is evaluated, must contain no descendant of that element). All ancestors of that element are retained in the residual collection.

Table 1. Precision at k for the baseline run

k	5	10	15	20
prec@k	0.231	0.204	0.191	0.174

For all experiments we used the TopX engine that fully supports the evaluation of weighted content-and-structure queries. The baseline for all experiments is a run for all 52 INEX topics, with 1500 results for each topic. Table 1 shows the macro-averaged precision for this run for the top-k ranked elements per topic, for different k; this corresponds to the average fraction of relevant results among the elements used for top-k feedback.

To select the candidates for query expansion, we tried using the plain weight, the number of relevant elements with the feature, and RSV. In our experiments, all of these gave similar results with a small advantage for RSV, so we report only the results for experiments with RSV for feature selection.

5.2 Results

Table 2 shows the MAP values of our experiments with different combinations of candidate classes for query expansion, providing relevance feedback for a different number of top elements of the baseline run and selecting the best 10 candidates for expansion. Note that all values, including those for the baseline, are computed for the residual collection. It is evident that using only candidates of class D consistently outperforms the established content-based feedback (candidates of class C), with an increase over the baseline run of almost 150% for top-15 feedback. Candidates of classes A and AD did not perform too well if used alone, but this could be expected as using these candidates potentially adds a lot of nonrelevant elements to the result of the expanded query (see Section 3.3). However, if they are combined with the other classes and relevance for at least 10 elements is provided, the combination outperforms all other combinations.

We also measured the precision of the feedback runs at different positions; the results are depicted in Table 3. Again, using class D candidates outperformed traditional content-based feedback and gained up to 100% increase over the baselines' precision values. The combination of all candidate classes was again best if feedback for more than 10 results was available.

Table 2. MAP values for top-k feedback runs with different configurations and different values of k

k	baseline	C	D	C+D	A	AD	A+AD	A+C+D+AD
5	0.0493	0.0727	0.0757	**0.0772**	0.0520	0.0540	0.0497	0.0647
10	0.0544	0.0748	**0.0784**	0.0778	0.0584	0.0528	0.0547	0.0777
15	0.0290	0.0659	0.0717	0.0709	0.0574	0.0560	0.0575	**0.0742**
20	0.0498	0.0644	0.0725	0.0721	0.0555	0.0594	0.0579	**0.0759**

Table 3. Precision@p values for top-k feedback runs with different configurations and different values of k

k	baseline	C	D	C+D	A	AD	A+AD	A+C+D+AD
p=5								
5	0.1529	0.1804	0.2118	**0.2157**	0.1725	0.1686	0.1765	0.2000
10	0.1373	0.2078	**0.2471**	0.2392	0.1686	0.1752	0.1451	0.2391
15	0.0980	0.1804	**0.1922**	0.1843	0.1412	0.1529	0.1490	**0.1922**
20	0.1120	0.1800	0.1960	**0.2160**	0.1360	0.1600	0.1360	0.2000
p=10								
5	0.1411	0.1569	0.1588	**0.1667**	0.1549	0.1490	0.1412	0.1569
10	0.1275	0.1647	0.1765	0.1745	0.1373	0.1471	0.1451	**0.1843**
15	0.1039	0.1569	0.1627	0.1647	0.1235	0.1294	0.1314	**0.1667**
20	0.1040	0.1480	0.1720	0.1700	0.1140	0.1320	0.1200	**0.1820**
p=15								
5	0.1333	0.1412	0.1412	**0.1464**	0.1373	0.1399	0.1373	0.1346
10	0.1163	0.1438	**0.1634**	0.1621	0.1280	0.1307	0.1294	0.1608
15	0.1033	0.1333	**0.1529**	0.1490	0.1124	0.1216	0.1176	0.1490
20	0.0933	0.1307	0.1360	0.1360	0.0987	0.1147	0.1080	**0.1467**
p=20								
5	0.1255	0.1373	0.1363	**0.1412**	0.1255	0.1275	0.1245	0.1245
10	0.1108	0.1314	**0.1431**	0.1412	0.1186	0.1245	0.1225	0.1373
15	0.0902	0.1245	**0.1382**	0.1343	0.0980	0.1078	0.1029	0.1343
20	0.0830	0.1170	0.1150	0.1190	0.0890	0.1080	0.0990	**0.1260**

6 Conclusion and Future Work

This paper has made important steps from content-based to structural feedback in XML retrieval. It presented an integrated solution for expanding keyword queries with new weighted content and structure constraints as a part of an extensible framework and showed huge performance gains with the established INEX benchmark of up to 150% for MAP and up to 100% for precision under the evaluation method used in the INEX 2004 relevance feedback track.

Our future work will contentrate on adding new candidate classes (like twigs, tags and paths) and extending this work to queries with content and structural constraints. We also plan to evaluate the effectiveness of our approach with pseudo relevance feedback.

References

1. G. Amati, C. Carpineto, and G. Romano. Merging XML indices. In *INEX Workshop 2004*, pages 77–81, 2004.
2. A. Balmin et al. A system for keyword proximity search on XML databases. In *VLDB 2003*, pages 1069–1072, 2003.
3. H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors. *Intelligent Search on XML Data*, volume 2818 of *LNCS*. Springer, Sept. 2003.

4. C. Crouch. Relevance feedback at the INEX 2004 workshop. In *SIGIR 2005 Forum*, 2005.
5. C. J. Crouch, A. Mahajan, and A. Bellamkonda. Flexible XML retrieval based on the extended vector model. In *INEX 2004 Workshop*, pages 149–153, 2004.
6. E. N. Efthimiadis. A user-centred evaluation of ranking algorithms for interactive query expansion. *SIGIR Forum (USA), special issue*, pages 146–159, 1993.
7. T. Grust. Accelerating XPath location steps. In *SIGMOD 2002*, pages 109–120, 2002.
8. L. Guo et al. XRANK: ranked keyword search over XML documents. In *SIGMOD 2003*, pages 16–27, 2003.
9. L. Hlaoua and M. Boughanem. Towards context and structural relevance feedback in XML retrieval. In *Workshop on Open Source Web Information Retrieval (OSWIR)*, 2005. `http://www.emse.fr/OSWIR05/`.
10. W. Hsu, M. L. Lee, and X. Wu. Path-augmented keyword search for XML documents. In *ICTAI 2004*, pages 526–530, 2004.
11. INEX relevance feedback track. `http://inex.is.informatik.uni-duisburg.de:2004/tracks/rel/`.
12. G. Kazai et al. The INEX evaluation initiative. In Blanken et al. [3], pages 279–293.
13. S. Liu, Q. Zou, and W. Chu. Configurable indexing and ranking for XML information retrieval. In *SIGIR 2004*, pages 88–95, 2004.
14. Y. Mass and M. Mandelbrod. Relevance feedback for XML retrieval. In *INEX 2004 Workshop*, pages 154–157, 2004.
15. V. Mihajlovič et al. TIJAH at INEX 2004 modeling phrases and relevance feedback. In *INEX 2004 Workshop*, pages 141–148, 2004.
16. H. Pan, A. Theobald, and R. Schenkel. Query refinement by relevance feedback in an XML retrieval system. In *ER 2004*, pages 854–855, 2004.
17. M. Porter and V. Galpin. Relevance fedback in a public access catalogue for a research library: Muscat at the Scott Polar Research Institute. *Program*, 22(1): 1–20, 1988.
18. G. Ramirez, T. Westerveld, and A. de Vries. Structural features in content oriented xml retrieval. Technical Report INS-E0508, CWI,Centre for Mathematics and Computer Science, 2005.
19. G. Ramírez, T. Westerveld, and A. P. de Vries. Structural features in content oriented XML retrieval. In *CIKM 2005*, 2005.
20. S. Robertson. On term selection for query expansion. *Journal of Documentation*, 46:359–364, Dec. 1990.
21. S. Robertson and K. Sparck-Jones. Relevance weighting of search terms. *Journal of the American Society of Information Science*, 27:129–146, May–June 1976.
22. J. Rocchio Jr. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 14, pages 313–323. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1971.
23. I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *Knowledge Engineering Review*, 18(1), 2003.
24. B. Sigurbjörnsson, J. Kamps, and M. de Rijke. The University of Amsterdam at INEX 2004. In *INEX 2004 Workshop*, pages 104–109, 2004.
25. M. Theobald, R. Schenkel, and G. Weikum. An efficient and versatile query engine for TopX search. In *VLDB 2005*, pages 625–636, 2005.
26. M. Theobald, R. Schenkel, and G. Weikum. TopX & XXL at INEX 2005. In *Preproceedings of the 2005 INEX Workshop*, Dagstuhl Castle, Germany, 2005.
27. A. Trotman and B. Sigurbjörnsson. Narrowed Extended XPath I (NEXI). available at `http://www.cs.otago.ac.nz/postgrads/andrew/2004-4.pdf`, 2004.

28. C. van Rijsbergen, D. Harper, and M. Porter. The selection of good search terms. *Information Processing and Management*, 17(2):77–91, 1981.
29. J.-N. Vittaut, B. Piwowarski, and P. Gallinari. An algebra for structured queries in bayesian networks. In *INEX Workshop 2004*, pages 58–64, 2004.
30. R. Weber. Using relevance feedback in XML retrieval. In Blanken et al. [3], pages 133–143.
31. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD 2005*, pages 537–538, 2005.