A database approach to content-based XML retrieval

Djoerd Hiemstra
University of Twente, Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands
d.hiemstra@utwente.nl

ABSTRACT

This paper describes a first prototype system for content-based retrieval from XML data. The system's design supports both XPath queries and complex information retrieval queries based on a language modelling approach to information retrieval. Evaluation using the INEX benchmark shows that it is beneficial if the system is biased to retrieve large XML fragments over small fragments.

Keywords: XML Databases, Information Retrieval, Language Models

1. INTRODUCTION

This paper describes a number of fundamental ideas and starting points for building a system that seamlessly integrates data retrieval and information retrieval (IR) functionality into a database system. We describe a first prototype system that is developed according to these ideas and starting points and report on experimental results of the system on the INEX collection. The current prototype system only support a small part of the functionality that we envision for future systems. In the upcoming years we will build a number of such prototype systems in the CIRQUID (Complex Information Retrieval Queries in a Database) project that is funded by the Netherlands Organisation for Scientific Research (NWO).

The CIRQUID project bridges the gap between structured query capabilities of XML query languages and relevance-oriented querying. Current techniques for XML querying, originating from the database field, do not support relevance-oriented querying. On the other hand, techniques for ranking documents, originating from the information retrieval field, typically do not take document structure into account. Ranking is of the utmost importance if large collections are queried, to assist the user in finding the most relevant documents in a retrieved set.

The paper is organised as follows: Section 2 describes our database approach to relevance-oriented querying from XML documents. Section 3 reports the experimental results of our first prototype system. Finally, Section 4 concludes this paper.

2. A MULTI-MODEL DATABASE APPROACH

A three level design of DBMSs – distinguishing a conceptual, a logical, and a physical level – provides the best opportunity for balancing flexibility and efficiency. In our approach, we take the three level architecture to its extreme. Not only do we guarantee logical and physical data independence between the three levels, we also map the conceptual data model used by the end users to a physical implementation *using different data models at different levels of the database architecture*: the so-called "multi-model" database approach [26].

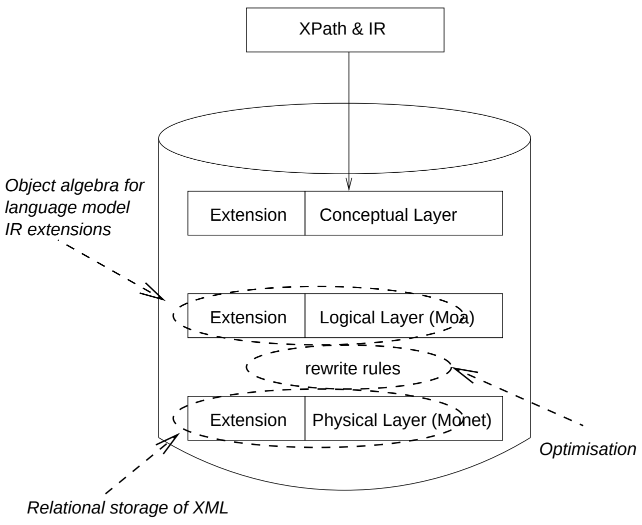

Figure 1: Database internals

Figure 1 shows a graphical representation of the approach. At the logical level, language models will be used to develop information retrieval primitives as a logical algebra. The physical level provides a relational storage of the XML data, including fast index structures. A new approach to query optimisation deals with the complex queries that combine structure and content at the logical level. In the following three subsections we will present some of the ideas and starting points for developing the three levels of the multi-model database approach.

2.1 All of XPath and modern IR queries

The conceptual level should support XML and IR queries. Our objective is to build a system that supports "all of XML and all of IR".

For XML, standards are currently emerging, and it seems reasonable to support the XPath standard for our "traditional database queries". Practically, this means that our system should contain a complete representation of the XML data, and that the system is able to reproduce (parts of) the data as the result of the query. For XPath we refer to [2].

Unlike the database and XML communities, which have developed some well-accepted standards in the past 30 years, the information retrieval community does not have any comparable standard query language or retrieval model. If we look at some practical systems however, e.g. internet search engines like Google and AltaVista, or online search services as provided by e.g. Dialog and LexisNexis, it turns out that there is much overlap in the kind of functionality they provide.

```
1.  IT magazines
2.  +IT magazine* -MSDOS
3.  "IT magazines"
4.  IT NEAR magazines
5.  (IT OR computer) (books OR magazines OR journals)
6.  XML[0.9] IR[0.1] title:INEX site:utwente.nl
```

Figure 2: Examples of complex IR queries

Figure 2 gives some example queries from these systems. The first query is a simple "query by example": retrieve a ranked list of documents about IT magazines. The second query shows the use of a mandatory term operator '`+`', stating that the retrieved document *must* contain the word IT,[1] a wild card operator '`*`' stating that the document might match "magazine", but also "magazines" or "magazined" and the '`-`' operator stating that we do not prefer IT magazines about MSDOS. The third and fourth query searches for documents in which "IT" and "magazines" occur respectively adjacent or near to each other. The fifth query shows the use of the 'OR' operator, stating that the system might retrieve documents about "IT magazines", "computer magazines", "IT journals", "IT books", etc. The sixth and last query shows the use of structural information, very much like the kind of functionality that is provided by XPath; so "`title:INEX`" means that the title of the document should contain the word INEX. The last query also shows additional term weighting, stating that the user finds XML much more important than IR.

These examples suggest that at the logical level, our system should support algebraic constructs for proximity of terms, mandatory terms, a logical OR, term weighting, etc. To support proximity operators the system should at least store term position information somehow at the physical level.

2.2 Moa and Language Models

Parts of a prototype multi-model database system have already been developed with the extensible object algebra Moa [14] as the logical layer. An open question in this setup is how Moa, which provides a highly structured nested object model with sets and tuples, can be adapted to managing semi-structured data. In this paper we will not get into Moa, but direct our attention to the language modelling approach to information retrieval as proposed in [9, 18] to guide the definition of the logical layer of our system.

[1]Note that most retrieval systems do not distinguish upper case from lower case, and confuse the acronym "IT" with the very common word "it".

The basic idea behind the language modelling approach to information retrieval is that we assign to each XML element X the probability that the element is relevant, given the query $Q = q_1, \cdots, q_n$. Using Bayes' rule we can rewrite that as follows.

$$P(X|q_1, q_2, \cdots, q_n) \;=\; \frac{P(q_1, q_2, \cdots, q_n|X)P(X)}{P(q_1, q_2, \cdots, q_n)} \qquad (1)$$

Note that the denominator on the right hand side does not depend on the XML element X. It might therefore be ignored when a ranking is needed. The prior $P(X)$ however, should only be ignored if we assume a uniform prior, that is, if we assume that all elements are equally likely to be relevant in absence of a query. Some non-content information, e.g. the number of accesses by other users to an XML element, or e.g. the length of an XML element, might be used to determine $P(X)$.

Let's turn our attention to $P(q_1, q_2, \cdots, q_n|X)$. The use of probability theory might here be justified by modelling the process of generating a query Q given an XML element as a random process. If we assume that this page in the INEX proceedings is an XML element in the data, one might imagine picking a word at random from the page by pointing at the page with closed eyes. Such a process would define a probability $P(q|X)$ for each term q, which might simply be calculated by the number of times a word occurs on this page, divided by the total number of words on the page. Similar generative probabilistic models have been used successfully in speech recognition systems [21], for which they are called "language models".

The mechanism above suggests that terms that do not occur in an XML element are assigned zero probability. However the fact that a term is never observed does not mean that this term is never entered in a query for which the XML element is relevant. The problem that events which are not observed in the data might still be reasonable in a new setting, is called the *sparse data problem* in the world of language models [16]. Zero probabilities should therefore be avoided. A standard solution to the sparse data problem is to interpolate the model $P(q|X)$ with a background model $P(q)$ which assigns a non-zero probability to each query term. If we additionally assume that query terms are independent given X, then:

$$P(q_1, q_2, \cdots, q_n|X) \;=\; \prod_{i=1}^{n} \Big((1-\lambda)P(q_i) + \lambda P(q_i|X) \Big) \quad (2)$$

Equation 2 defines our basic language model if we assume that each term is generated independently from previous terms given the relevant document. Here, λ is an unknown mixture parameter, which might be set using e.g. relevance feedback of the user. Ideally, we would like to train the probability of an unimportant term $P(q_i)$ on a large corpus of queries. In practice however, we will use the document collection to define these probabilities. By some simple rewriting, it can be shown that Equation 2 can be implemented as a vector space weighting algorithm [10].

Why would we prefer the use of language models over the use of e.g. a vector space model with some *tf.idf* weighting algorithm as in [22]? The reason is the following: our generative query language model gives a nice intuitive explanation of *tf.idf* weighting algorithms by means of calculating the probability of picking at random, one at a time, the query terms from an XML element. We might extend this by any other generating process to model complex information retrieval queries in a theoretically sound way that is not provided by a vector space approach. For instance, we might calculate the probability of sampling either "magazines" or "books" or "journals" from the XML document by summing the probabilities $P(\text{magazines}|X)$, $P(\text{journals}|X)$, and $P(\text{books}|X)$. So, Query 5 from Figure 2 would assign the following probability to each XML element (ignoring for a moment the prior $P(X)$ and the linear interpolation with the background model $P(q_i)$ for simplification of the example).

$$P(\text{Query 5}) = (P(\text{IT}|X) + P(\text{computer}|X)) \cdot (P(\text{books}|X) + P(\text{journals}|X) + P(\text{magazines}|X))$$

Interestingly, a similar approach was proposed in 1960 by Maron and Kuhns [17]. In a time when manual indexing was still guiding the field, they suggested that an indexer, which runs through the various possible index terms q that possibly apply to a document, might assign a probability $P(q|X)$ to a term given a document instead of making a yes/no decision. The language modelling equivalent of 'disjunction' and 'conjunction' (i.e. 'AND' and 'OR' operators) is motivated by adding a so-called translation model to the basic model [1, 13, 27].

In CIRQUID we will explore language modelling approaches that model all structured queries in Figure 2. The interested reader is referred to [18, 25] for so-called bigram models for proximity queries, and [12] for mandatory terms. A similar approach to querying XML data is proposed by List and De Vries [15], and Ogilvie and Callan [19].

2.3 Relational storage

At the physical level, we will use the 'good-old' relational model for storage of the data. In order to combine XPath and information retrieval functionality, we somehow have to combine relational data representations of XML as described in e.g. [4, 24], and relational representations of information retrieval indexing structures as described by e.g. [3, 7, 26]. Our starting point for the relational storage of the XML data is that it should not critically depend on the existence of a schema or DTD, and that it should be possible to reconstruct the XML data completely. Our starting point for the storage of information retrieval indexing structures is that it should provide the 'traditional information retrieval' functionality as well as term position information to support proximity queries.

Related work on XML storage

A standard approach to storing hierarchical or nested data, with or without a schema, is to store each "instance node" separately in a relational table. This is illustrated in Figure 3, 4 and 5. Figure 4 shows a tree representation of the XML instance of Figure 3. Each node in the tree is assigned a node identifier "id".

Now for each node, we might store its id and the id of its parent as shown in Figure 5. One can think of numerous alternative ways to assign the ids to the instance nodes (in this example they were assigned in pre-order). Similarly, one can think of many relational schemas that support this basic idea, by fragmenting the tables of Figure 5 in various ways. In previous work, we used a full fragmentation in binary relational tables [14] which provides efficient support for XML querying [24].

```
<article>
  <au><fnm>Boudewijn</fnm><snm>Büch</snm></au>
  <atl>Kleine blonde dood</atl>
  <bdy>
    <p>Een schrijver ontmoet een oude bekende.</p>
    <p>Er ontstaat een liefdesrelatie.</p>
  </bdy>
</article>
```

Figure 3: Example XML data

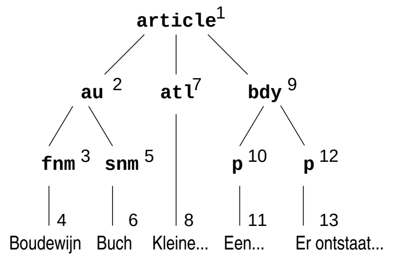

Figure 4: Tree representation of the data

tags

id	parent	tag_name
1	0	article
2	1	au
3	2	fnm
5	2	snm
7	1	atl
9	1	bdy
10	9	p
12	9	p

pcdata

id	parent	string
4	3	Boudewijn
6	5	Büch
8	7	Kleine blonde ...
11	10	Een schrijver ...
13	12	Er ontstaat ...

Figure 5: Example relational storage of XML data

Related work on the storage of IR indexes

A standard approach to the relational storage of information retrieval indexes uses two tables. One table stores the document term statistics, i.e. for each document-term pair some statistics related to the number of times the term occurs in the document. A second table stores the global term statistics, i.e. for each term some statistics related to the total number of times that a term occurs in the entire collection. In traditional systems that use a *tf.idf* term weighting algorithm, the first table contains the *tf*'s (term frequencies) and the second table contains the *df*'s (document frequencies). In the language modelling approach we might store $P(q|X)$ in the first table and $P(q)$ in the second.

In [3, 7, 26], `id` refers to a document identifier. For XML data it should refer to the node id of the XML element as shown in Figure 4 and 5. A fundamental problem with this approach is the following. If we include all word-id pairs in the table `local_stats` of Figure 6, then each word in the data will occur as often as the average depth of the XML data. For INEX, the average depth is about 7, so our information retrieval index would be 7 times as big as local_stats

word	id	P(word|id)
aardvark	43	0.007
after	3	0.09
after	42	0.11
after	78	0.015
after	980	0.067
affect	321	0.2
ambient	761	0.0001
:		
bekende	1	0.031
blonde	1	0.031
boudewijn	1	0.031
:		

global_stats

word	P(word)
aardvark	0.00001
after	0.0345
affect	0.00055
ambient	0.0000001
an	0.107
:	
:	
:	
:	
:	
:	
:	

Figure 6: Example relational storage of an IR index

the "regular" index that only indexes traditional documents (e.g. web pages).

A solution to this problem is to let the database administrator choose the nodes that need to be indexed, the so-called "indexing nodes" [5, 28], however, this will restrict the functionality such that queries like `//*[. =~ "computational biology"]` (pseudo "XPath+IR" for any element about "computational biology") would be impossible, or only possible by inefficient linear scans over all string fields in the `pcdata` table of Figure 5.

An alternative solution to this problem is to only store all leaf nodes of the XML data in `local_stats` as suggested in [6]. In this case, queries like `//article[. =~ "computational biology"]` (any *article* element about "computational biology") would need a number of repeated joins with the table `tags` of Figure 5 in order to determine the id of the article node that contains the query terms.

Instead of storing the tag name, one could store the complete path in Figure 5. This would solve only part of the problem, because it would require a special purpose implementation of regular path matches on attributes.

```
SELECT id, SUM(f(local_stats.p, global_stats.p)) AS s
FROM local_stats, global_stats
WHERE local_stats.word = global_stats.word
  AND (local_stats.word = 'computational'
    OR local_stats.word = 'biology')
GROUP BY id
ORDER BY s DESC
```

Figure 7: Traditional IR query in pseudo SQL

Figure 7 shows the typical information retrieval ranking algorithm expressed in SQL to give the reader a flavour of how the system uses the tables of Figure 6 at run time. In practice, we will not use SQL at the physical level. The function `f` in the algorithm might be any *tf.idf* formula. In case of the language modelling approach, `f` might be defined as $\log(1 + P(q|X)/P(q))$ [10].

A first prototype

For our first prototype we implemented the XML storage scheme proposed by Grust [8]. Grust suggests to assign two identifiers to each instance node: one id is assigned in pre-order, and the other in post-order. These ids replace the explicit parent-child relations as described in the previous paragraphs.[2] The pre and post order assignment of XML element ids provides elegant support for processing XPath queries.

`<article>`^{1}
  `<au>`^{2}`<fnm>`^{3}`Boudewijn`^{4}`</fnm>`^{5}`<snm>`^{6}`Büch`^{7}`</snm>`^{8}`</au>`^{9}
  `<atl>`^{10}`Kleine`^{11} `blonde`^{12} `dood`^{13}`</atl>`^{14}
  `<bdy>`^{15}
    `<p>`^{16}`Een`^{17} `schrijver ontmoet een oude bekende.</p>`
    `<p>Er ontstaat een liefdesrelatie.</p>`
  `</bdy>`
`</article>`

Figure 8: Example XML document: assigning ids

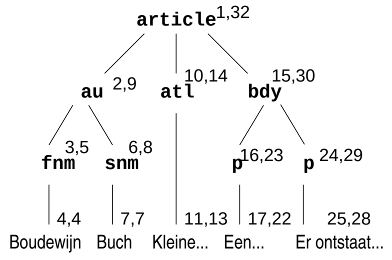

Figure 9: Tree representation: assigning ids

Note that pre and post order assignment can be done almost trivially in XML by keeping track of the order of respectively the opening and closing tags as shown in Figure 8 and 9. Both figures also show that position information is assigned to each word in the data. These positions will be used in our term position index. This leads to the relational storage of XML data as shown in Figure 10 and the relational storage of the information retrieval positional index as shown in Figure 11.

tags2

pre	post	tag_name
1	32	article
2	9	au
3	5	fnm
6	8	snm
10	14	atl
15	30	bdy
16	23	p
24	29	p

pcdata2

pre	post	string
4	4	Boudewijn
7	7	Büch
11	13	Kleine blonde...
17	22	Een schrijver ...
25	28	Er ontstaat ...

Figure 10: Relational storage of XML data

Note that exactly one 'join' (on the condition: `position > pre and position < post`, counting the positions) will give us a table that is similar to `local_stats` in Figure 6. Figure 12 expresses this in SQL.

Also note that, unlike the approaches in [6, 28], we are not interested in the total number of times a term occurs in a

[2]Actually, [8] store the id of the parent as well. Similarly, in [24] a field is added to keep track of the order of XML elements; here we emphasise different view points.

position_index

word	position
bekende	22
blonde	12
boudewijn	4
büch	7
een	17
een	20
een	27
er	25
kleine	11
:	

global_stats

word	P(word)
bekende	0.00321
blonde	0.00013
boudewijn	0.00004
büch	0.00001
een	0.0991
er	0.0145
:	

Figure 11: Relational storage of the IR positional index

```
CREATE VIEW local_stats2 AS
  SELECT word, pre
    CAST(COUNT(position) AS float) / (post - pre) AS p
  FROM position_index, tags2
  WHERE position > pre
    AND position < post
  GROUP BY word, pre
```

Figure 12: Combining term information and the structured information in pseudo SQL

certain XML element type (that is, the so-called 'document frequency' of the term). The language modelling approach suggests that $P(q)$ is the probability of a term in "general query English": It should be the same for all queries. Furthermore, to avoid the sparse data problem, it should be estimated on as much data as possible. In our case, $P(q)$ is defined by the total number of occurrences of q in the entire INEX collection, divided by the total number of term occurrences in INEX (i.e. the "collection length" measured in the number of words).

2.4 Optimisation

As an example of a logical optimisation step, let's have a look at the fifth query of Figure 2 again. For the second part of Query 5, $P(\text{books OR journals OR magazines}|X)$ is defined in Section 2.2 as:

$$P(\text{books}|X) + P(\text{journals}|X) + P(\text{magazines}|X)$$

Remember that each $P(q|X)$ is defined by the 'join' of Figure 12. This suggests that we have to do the 'join' for each of the words *books*, *journals* and *magazines*, and then group them by the XML element id, adding the probabilities. In [11] it is shown that a more efficient approach would be to first determine the number of occurrences of either (*books* OR *journals* OR *magazines*) and then compute the probability by dividing by the length of the XML element. So, we could first do a selection of (*books* OR *journals* OR *magazines*) on the position index, and then do the 'join' with the tags table. This way we avoid two of the three joins. A similar optimisation step is in general not possible in extended Boolean models [23] and fuzzy set models [20].

To understand what is happening here, note that each occurrence of (*books* OR *journals* OR *magazines*) actually has its own position. At any place in the XML data where either *books*, or *journals*, or *magazines* occurs, we actually know its position. We cannot do a similar optimisation for 'AND' queries (Note that all queries of Figure 2, except for Query 5, are implicit 'AND' queries), that is, the words *books*, *journals*, and *magazines* occur nowhere in the data on exactly the same position, for the simple reason that each position contains exactly one word.

The above example shows a simple, almost trivial, optimisation step. A modern database query optimiser should be able to reason over queries that contain clauses over data structures that are typically implemented in different extensions of the DBMS. Current, state-of-the-art optimiser technology can deal with extensions in isolation. In future work, we will design an inter-object optimiser layer that is able to bridge the typical orthogonality of database extensions. At the logical level, the query optimiser will be extended to handle interacting extensions, including e.g. extensions for other media.

3. EXPERIMENTAL SETUP AND RESULTS

In this section we describe the experimental setup and the evaluation results of the system using the INEX testbed. We describe the tasks and evaluation procedure, the system setup and research questions, and finally the experimental results.

3.1 The INEX evaluation

INEX is the Initiative for the Evaluation of XML Retrieval. The initiative provides a large testbed, consisting of XML documents, retrieval tasks, and relevance judgements on the data. INEX identifies two tasks: the content-only task, and the content-and-structure task.

The content-only task provides queries of the form: `//*[. =~ "computational biology"]` ("XPath+IR" for: any element about "computational biology"). In this task, the system needs to identify the most appropriate XML element for retrieval. The task resembles users that want to search XML data without knowing the schema or DTD.

The content-and-structure task provides queries of the form: `//article[author =~ "Smith|Jones" and bdy =~ "software engineering"]` ("XPath+IR" for: retrieve articles written by either Smith or Jones about software engineering). This task resembles users or applications that *do* know the schema or DTD, and want to search some particular XML elements while formulating restrictions on some other elements.

For each query in both tasks, quality assessments are available. XML elements are assessed based on *relevance* and *coverage*. Relevance is judged on a four-point scale from 0 (irrelevant) to 3 (highly relevant). Coverage is judged by the following four categories: N (no coverage), E (exact coverage), L (the XML element is too large), and S (the XML element is too small).

In order to apply traditional evaluation metrics like precision and recall, the values for relevance and coverage must be quantised to a single quality value. INEX suggests the use of two quantisation functions: Strict and liberal quantisation. The strict quantisation function evaluates whether a given retrieval method is capable of retrieving highly relevant XML elements: it assigns 1 to elements that have a relevance value 3, and exact coverage. The liberal quantisation function assigns 1 to elements that have a relevance value of 2 and exact coverage, or, a relevance value of 3 and either exact, too small, or too big coverage.

3.2 System setup and research questions

We evaluate a system that only has limited functionality. First of all, we assume that $\lambda = 1$ in Equation 2, so we do not have to store the `global_stats` table of Figure 11. The system supports queries with a content restriction on only one XML element, so the example content-and-structure query in the previous section is not supported: Either the restriction on the `author` tag, or the restriction on the `bdy` tag has to be dropped. The system supports conjunction and disjunction operators, which are evaluated as defined in the example of Query 5 at the end of Section 2.2. All queries were manually formulated from the topic statements.

The experiments are designed to answer the following research question: Can we use the prior probability $P(X)$ (see Equation 1) to improve the retrieval quality of the system? We present three experiments using the system described in this paper, for which only the prior probabilities $P(X)$ differ. The baseline experiment uses a uniform prior $P(X) = c$, where c is some constant value, so each XML element will have the same a priori probability of being retrieved. A second experiment uses a length prior $P(X) =$ *number of tokens in the XML element*, where a token is either a word or a tag. This means that the system will prefer bigger elements, i.e. elements higher up the XML tree, over smaller elements. A third experiment uses a prior that is somewhere in between the two extremes. The prior is defined by $P(X) = 100 +$ *number of tokens in the XML element.* Of course, the priors should be properly scaled, but the exact scaling does not matter for the purpose of ranking. We hypothesise that the system using the length prior will outperform the baseline system

3.3 Evaluation results

This section presents the evaluation results of three retrieval approaches (no prior, 'half' prior, and length prior) on two query sets (content-only, and content-and-structure), following two evaluation methods (strict and liberal). We will report for each combination the precision at respectively 5, 10, 15, 20, 30 and 100 documents retrieved.

Strict evaluation

Table 1 shows the results of the three experiments on the content-only queries following the strict evaluation. The precision values are averages over 22 queries. The results show an impressive improvement of the length prior on all cut-off values. Apparantly, if the elements that need to be retrieved are not specified in the query, users prefer larger elements over smaller elements.

precision	no prior	'half' prior	length prior
at 5	0.0455	0.0455	0.1909
at 10	0.0364	0.0455	0.1591
at 15	0.0303	0.0424	0.1394
at 20	0.0341	0.0364	0.1318
at 30	0.0364	0.0424	0.1318
at 100	0.0373	0.0559	0.1000

Table 1: Results of content-only (CO) runs with strict evaluation

Table 2 shows the results of the three experiments on the content-and-structure queries following the strict evaluation. The precision values are averages over 28 queries. The baseline system performs much better on the content-and-structure queries than on the content-only queries. Surprisingly, the length prior again leads to substantial improvement on all cut-off values in the ranked list.

precision	no prior	'half' prior	length prior
at 5	0.1929	0.2357	0.2857
at 10	0.1964	0.2321	0.2857
at 15	0.1976	0.2333	0.2714
at 20	0.1929	0.2232	0.2589
at 30	0.1786	0.2060	0.2607
at 100	0.0954	0.1107	0.1471

Table 2: Results of content-and-structure (CAS) runs with strict evaluation

Liberal evaluation

Table 3 shows the results of the three experiments on the content-only queries using the liberal quantisation function defined above for evaluation. The precision values are averages over 23 queries. Again, the results show a significant improvement of the length prior on all cut-off values.

precision	no prior	'half' prior	length prior
at 5	0.1130	0.1391	0.4261
at 10	0.0957	0.1304	0.3609
at 15	0.0957	0.1333	0.3304
at 20	0.1000	0.1152	0.3000
at 30	0.1087	0.1232	0.2812
at 100	0.0896	0.1222	0.2065

Table 3: Results of content-only (CO) runs with liberal evaluation

Table 4 shows the results of the three experiments on the content-and-structure queries following the liberal evaluation. The precision values are averages over 28 queries. The length prior again shows better performance on all cut-off values. Note that the content-only task and the content-and-structure task show practically equal performance if the liberal evaluation procedure is followed.

precision	no prior	'half' prior	length prior
at 5	0.2429	0.2929	0.4000
at 10	0.2286	0.2823	0.3750
at 15	0.2262	0.2881	0.3738
at 20	0.2268	0.2821	0.3607
at 30	0.2179	0.2583	0.3595
at 100	0.1279	0.1571	0.2054

Table 4: Results of content-and-structure (CAS) runs with liberal evaluation

4. DISCUSSION AND FUTURE WORK

We presented an initial design and implementation of a system that supports XPath and complex information retrieval queries. In the CIRQUID project we will develop an algebra that allows us to define complex queries using language modelling primitives, like bigrams (proximity) conditional independence, and mixture models.

From the INEX experiments we conclude that it is beneficial to assign a higher prior probability of relevance to bigger fragments of XML data than to smaller XML fragments, that is, to users, more information seems to be better information.

Acknowledgements

The research presented in this paper was funded in part by the Netherlands Organisation for Scientific Research (NWO).

5. REFERENCES

[1] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *Proceedings of the 22nd ACM Conference on Research and Development in Information Retrieval (SIGIR'99)*, pages 222–229, 1999.

[2] A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Simeon. XML Path language (XPath) 2.0. Technical report, World Wide Web Consortium, 2002. http://www.w3.org/TR/xpath20/

[3] H.E. Blok. *Database Optimization Aspects for Information Retrieval*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, 2002.

[4] D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. In *Proceedings of the VLDB'99*, pages 105–110, 2001.

[5] N. Fuhr and K. Grossjohann. XIRQL: A query language for information retrieval in XML. In *Proceedings of the 24th ACM Conference on Research and Development in Information Retrieval (SIGIR'01)*, pages 172–180, 2001.

[6] T. Grabs. Generating vector spaces on-the-fly for flexible XML retrieval. In *Proceedings of the SIGIR workshop on XML and Information Retrieval*, pages 4–13, 2002.

[7] D.A. Grossman, O. Frieder, D.O. Holmes, and D.C. Roberts. Integrating Structured Data and Text: A Relational Approach. *Journal of the American Society of Information Science*, 48(2):122–132, 1997.

[8] T. Grust, Accelerating XPath location steps. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 109–120, 2002.

[9] D. Hiemstra. A linguistically motivated probabilistic model of information retrieval. In *Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 569–584, 1998.

[10] D. Hiemstra. A probabilistic justification for using tf.idf term weighting in information retrieval. *International Journal on Digital Libraries*, 3(2):131–139, 2000.

[11] D. Hiemstra. *Using language models for information retrieval*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, 2001.

[12] D. Hiemstra. Term-specific smoothing for the language modeling approach to information retrieval: The importance of a query term. In *Proceedings of the 25th ACM Conference on Research and Development in Information Retrieval (SIGIR'02)*, pages 35–41, 2002.

[13] D. Hiemstra and F.M.G. de Jong. Disambiguation strategies for cross-language information retrieval. In *Proceedings of the third European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 274–293, 1999.

[14] M. van Keulen, J. Vonk, A.P. de Vries, J. Flokstra, and H.E. Blok. Moa: extensibility and efficiency in querying nested data. Technical report 02-19, Centre for Telematics and Information Technology, 2002.

[15] J. List and A.P. de Vries. CWI at INEX. In *Proceedings of the first INEX workshop*, 2003. (in this volume)

[16] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[17] M.E. Maron and J.L. Kuhns. On relevance, probabilistic indexing and information retrieval. *Journal of the Association for Computing Machinery*, 7:216–244, 1960.

[18] D.R.H. Miller, T. Leek, and R.M. Schwartz. A hidden Markov model information retrieval system. In *Proceedings of the 22nd ACM Conference on Research and Development in Information Retrieval (SIGIR'99)*, pages 214–221, 1999.

[19] P. Ogilvie and J. Callan. Language Models and Structured Document Retrieval. In *Proceedings of the first INEX workshop*, 2003. (in this volume)

[20] C.P. Paice. Soft evaluation of Boolean search queries in information retrieval systems. *Information Technology: Research and Development*, 3(1):33–42, 1984.

[21] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In A. Waibel and K.F. Lee, editors, *Readings in speech recognition*, pages 267–296. Morgan Kaufmann, 1990.

[22] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[23] G. Salton, E.A. Fox, and H. Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

[24] A. R. Schmidt, M. L. Kersten, M. A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents (Extended Version). In *The World Wide Web and Databases - Selected Papers of WebDB 2000*, Springer-Verlag, pages 137–150, 2000.

[25] F. Song and W.B. Croft. A general language model for information retrieval. In *Proceedings of the 8th International Conference on Information and Knowledge Management, CIKM'99*, pages 316–321, 1999.

[26] A.P. de Vries. *Content and Multimedia Database Management Systems*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, 1999.

[27] J. Xu, R. Weischedel, and C. Nguyen. Evaluating a probabilistic model for cross-lingual information retrieval. In *Proceedings of the 24th ACM Conference on Research and Development in Information Retrieval (SIGIR'01)*, pages 105–110, 2001.

[28] R. van Zwol. *Modelling and searching web-based document collections*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, 2002.

A. APPENDIX: THE QUERIES

We formulated the following "XPath+IR" queries from the INEX topics 1–60. The first 30 queries are the content-and-structure queries, whereas the queries 31–60 are the content-only queries.

query 01: `//article[. =~ "description (logic|logics)"]/fm/au`

query 02: `//ack[. =~ "(funds|funded|fund) (usa|america|darpa|arpa)"]`

query 03: `//*[. =~ "information (visualization|visualisation) (database|databases|dbms)"]`

query 04: `//*[. =~ "extreme programming (results|experiences|problems)"]`

query 05: `//article[. =~ "qbic"]/fm/tig`

query 06: `//article[. =~ "(tutorial|survery) (programming|software)"]/fm/tig`

query 07: `//sec[. =~ "(video|mpeg) (retrieval|database|databases|dbms)"]`

query 08: `//article[. =~ "certificate ibm"]`

query 09: `//article[. =~ "nonmonotonic reasoning"]`

query 10: `//p[. =~ "book review"]`

query 11: `//*[. =~ "(wireless|mobile) (secure|security)"]`

query 12: `//sec[. =~ "(internet|www|web) search"]`

query 13: `//article[. =~ "review (virtual|augmented) reality"]/fm/au`

query 14: `//p[. =~ "(corba|orb) figure"]`

query 15: `//bb[. =~ "hypercube|mesh|torus"]`

query 16: `//article[. =~ "concurrency control"]/fm/tig/atl`

query 17: `//bb[. =~ "croft"]`

query 18: `//article[. =~ "hypertext information retrieval"]`

query 19: `//p[. =~ "singular value decomposition formula"]`

query 20: `//article[. =~ "concurrency control"]//sec`

query 21: `//p[. =~ "recommender (system|systems|agent|agents)"]`

query 22: `//article[. =~ "manilla"]/fm/au`

query 23: `//article[. =~ "xml commerce"]`

query 24: `//article[fm/au =~ "smith|jones"]`

query 25: `//article[. =~ "(qos|quality) service"]`

query 26: `//article[. =~ "xml"]/fm/tig/atl`

query 27: `//article[. =~ "ieee transactions visualization computer graphics"]//reviewer`

query 28: `//article[. =~ "special feature ieee micro"]/fm/tig/atl`

query 29: `//*[. =~ "image retrieval (colour|shape|texture)"]`

query 30: `//article[. =~ "parallel||parallelism"]//au`

query 31: `//*[. =~ "(genome|genomics|biology|bioinformatics|protemics|protein) (computation|computer|informatics)"]`

query 32: `//*[. =~ "(semantic|rdf|ontology|ontologies|meta|services) (web|internet|www)"]`

query 33: `//*[. =~ "software (patent|patents)"]`

query 34: `//*[. =~ "(efficient|fast) (search|index|data|access)"]`

query 35: `//*[. =~ "parallel (query|sql) (optimization|optimization|optimizer|optimiser)"]`

query 36: `//*[. =~ "(heath|cooling|thermal) (circuit|circuits|chip|chips)"]`

query 37: `//*[. =~ "temporal (database|databases|query|queries|dbms)"]`

query 38: `//*[. =~ "(multidemensional|feature|features|vector) (index|indices|access)"]`

query 39: `//*[. =~ "video demand"]`

query 40: `//*[. =~ "(content|multimedia|audio|image|images|video) (dbms|database|databases|search|retrieval)"]`

query 41: `//*[. =~ "(millenium|year) (bug|problems|problem|compliance) (money|spent|spend|budget|financial)"]`

query 42: `//*[. =~ "enigma"]`

query 43: `//*[. =~ "(aproximate|partial) (string|strings) (match|matching|algorithm|shift)"]`

query 44: `//*[. =~ "(social|sociology|sociological|society|culture) (internet|www|web|mail|usenet)"]`

query 45: `//*[. =~ "(augmented|virtual) (reality|world) (medicine|surgery|health)"]`

query 46: `//*[. =~ "(firewalls|firewall) (internet|www|web|mail|usenet) (secure|security|authentication)"]`

query 47: `//*[. =~ "semantic (transaction|transactions) (manager|management|techniques) (simulation|evaluation)"]`

query 48: `//*[. =~ "(adb|active|trigger|event) (database|databases|dbms) (rule|rules|syntax|speficication|semantics)"]`

query 49: `//*[. =~ "(query|search|queries|querying) (relaxation|approximate|fuzzy|intelligent)"]`

query 50: `//*[. =~ "xml (parsing|parser|parsers|edit|editing|editor|editors)"]`

query 51: `//*[. =~ "(knowlegde|text|data) (mining|textmining|datamining)"]`

query 52: `//*[. =~ "(ussr|soviet|lebedev|glushkov|besm) history"]`

query 53: `//*[. =~ "(semistructured|xml) (retrieval|ranking|search)"]`

query 54: `//*[. =~ "knowledge (building|acquisition|sharing)"]`

query 55: `//*[. =~ "digital (divide|planning) (city|cities|neighbourhood|community|communities)"]`

query 56: `//*[. =~ "open (agents|hypermedia)"]`

query 57: `//*[. =~ "public (key|rsa|dsa|cryptography)"]`

query 58: `//*[. =~ "(location|locate|find|finding) (mobile|wireless)"]`

query 59: `//*[. =~ "(schema|schemas) (integration|integrate) (database|databases|dbms)"]`

query 60: `//*[. =~ "(speed|fast|efficient|performance) (web|www|internet)"]`