JuruXML – an XML retrieval system at INEX'02

Yosi Mass, Matan Mandelbrod, Einat Amitay, David Carmel, Yoelle Maarek, Aya Soffer
IBM Research Lab
Haifa 31905, Israel
+972-3-6401627
{yosimass, matan, einat, carmel, yoelle, ayas}@il.ibm.com

ABSTRACT

XML documents represent a middle range between unstructured data such as textual documents and fully structured data encoded in databases. Typically, information retrieval techniques are used to support search on the "unstructured" end of this scale, while database techniques are used for the other end. To date, most of the work on XML query and search has stemmed from the structured side and is strongly inspired by database techniques**.** We describe here an approach that originates from the "unstructured" end and is based on augmentation of information retrieval techniques. It is specifically targeted to support the information needs of end-users, more specifically a generic querying mechanism, and ranking of results for approximate needs. We describe our query format and ranking mechanism and demonstrate how it was used to run the INEX topics.

Keywords

XML Search, Information Retrieval, Vector Space Model.

1. INTRODUCTION

To date, most of the work on XML query and search has stemmed from the document management and database communities and from the information needs of business applications, as evidenced by existing XML query languages such as W3C's XPath[9] or XQuery [10], which are strongly inspired by SQL. We propose here to extend the realm of XML by supporting the information needs of users wishing to query XML collections in a flexible way without knowing much about the documents structure. Rather than inventing a new query language, we suggest to query XML documents via pieces of XML documents or "XML fragments" of the same nature as the documents that are queried. We then present an extension of the vector space model for ranking XML results by relevance.

We have extended Juru [3], a full-text information retrieval system developed at the IBM Research Lab in Haifa, to handle XML documents. INEX provided a useful framework to evaluate the capabilities of our query format and ranking methods. The rest of the paper is organized as follows, Section 2 introduces our query format and mechanism. Section 3 shows how the INEX topics were translated to this format. Section 4 proposes various ranking approaches and Section 5 provides some implementation details of our system. We conclude in Section 6 by describing our three INEX runs.

2. THE QUERY FORMAT

As stated above, we propose to tackle the XML search issue from an information retrieval (IR) perspective, and thus support the information needs of users wishing to query XML collections in a flexible way. In a classical IR system, the document collection consists of "free-text' documents and the query is expressed in free text. We claim that the same can hold for XML collections and we suggest to query XML documents via pieces of XML documents or "XML fragments" of the same nature as the documents that are queried. Returned results should be not only perfect matches but also "close enough" ones ranked according to some measure of relevance.

One key element of this work is to avoid defining yet another sophisticated XML query language but rather to allow users to express their needs as fragments of XML documents, or XML fragments for short. Users should not need to reformulate their queries as they may become too specific. The ranking mechanism should be responsible for giving priority to the closest form. This approach of using a very simple "fragment-based" language rather than SQL-like query languages (e.g., XQuery [10]) is somewhat analogous to using free-text rather than Boolean queries in IR: less control is given to the user, and most of the logic is put in the ranking mechanism so as to best match the user's needs.

2.1 Query syntax

XML fragments are portions of XML, possibly combined with free text, which can be viewed as a tree[1]. Documents that contain the query or part of it as a subtree are returned as results. XML attributes are queried using the same syntax used in the XML documents[2].

[1] We add an artificial root node that encloses the whole query so as to make it a valid XML data

[2] As an alternative, attributes can be queried as if they were children node of their containing node.

The default semantic of a query is that a document/component is considered a valid result if it contains at least one path of the query tree from the root to a leaf (see examples below), or to follow the vector space model, if it has a non-null similarity with the query profile.
In order to allow for more control on the XML fragments and yet still keep their simple intuitive syntax, we augment the XML fragments with the following symbols:

- **"+/-" :** a +/- prefix can be added to elements, attributes or content. Prefixing an element with a "+" operator in the XML fragment means that the subtree below the node associated with this element should be fully contained in any retrieved document. Prefixing an element with "–" means that the sub tree below the node associated with the element, should not exist in any retrieved document. For example:
    - *<Book><Title>**-**Graph Theory</Title></Book>* as a query, will return all books whose title contains the word "theory" but not the word "graph".
    - *<Book><-Abstract></Abstract></Book>* will return all books that do not contain abstracts.
- **"…" (phrase) :** Users can enclose any free text part of the XML fragment between quotes (**""**) to support phrase match.
- **At least one:** An exception to the regular + operator behavior occurs when it is applied to two or more sibling elements of exactly the same type (i.e., having the same name). In this case, the semantics of + is that **at least one** of the subtrees below one of those sibling nodes must hold even if they have some internal + nodes (see example in Section 2.2.3)

2.1.1 Target elements
The user can accompany the query with an optional list of target elements *(te)* to be returned. If there are no defined *te*'s then the search engine is left the freedom to decide whether it should return the entire document and/or the most relevant components. The decision is based on the ranking requirements and depends on the granularity level at which statistics (e.g. term frequency) are stored. We discuss our implementation in section 5.1.1 below.

2.2 Query examples
2.2.1 Task: Find books written by John.

Users with no knowledge of the documents DTD or schema, may simply issue a query in pure free text of the form "books written by John". However, if they have some basic knowledge of the DTD, their query can become:

```
<book>
  <author>John</author>
</book>
```

One key contribution of our technique is that the structured query does not need to express a "perfect" need, rather we allow for approximate matching. Thus for the above query, the system would also assign a non-null score to documents containing a fragment of the form below.

```
<book>
   <fm><author><first>John</first></author></fm>
</book>
```

2.2.2 Task: Find books written by John Doe

```
<+book>
  <author>John Doe</author>
</book>
```

In this example, <+book> imposes the constraint that there be an instance of <author> that contains both John and Doe under the same <author> instance. Thus the + avoids results in which there are two different authors one with <fnm>John and the second with <snm>Doe. The above syntax is similar to

```
<book><+author>John Doe</author></book>
```
and to
```
<book><author>+John +Doe</author></book>
```

2.2.3 Task: Retrieve all articles from the years 1999-2000 that deal with works on nonmonotonic reasoning. Do not retrieve articles that are calendar/call for papers

```
<bdy> <sec>+"nonmonotonic reasoning"</sec> </bdy>
<hdr>
    <yr>+1999</yr>
    <yr>+2000</yr>
</hdr>
<tig> <atl>-calendar –"call for papers"</atl> </tig>
```

In this example, we have two sibling <yr> nodes labeled with +. This means that a valid result should contain at least one of the years 1999 or 2000.

3. INEX QUERY TRANSLATION
We describe below how we translated the INEX topics into our query format. Note that the translation rules specified here are systematically applied to all queries. Their purpose is to capture the semantics of the INEX topics format (See its DTD in Figure 1) so as to best express it in our formalism.

```
<!ELEMENT INEX-Topic
<Title,Description,Narrative,Keywords)>
<!ATTLIST INEX-Topic
       topic-id        CDATA  #REQUIRED
       query-type      CDATA  #REQUIRED
       ct-no           CDATA  #REQUIRED
>
<!ELEMENT Title (te?, (cw, ce?)+)>
<!ELEMENT te     (#PCDATA)>
<!ELEMENT cw     (#PCDATA)>
<!ELEMENT ce     (#PCDATA)>
<!ELEMENT Description  (#PCDATA)>
<!ELEMENT Narrative    (#PCDATA)>
<!ELEMENT Keywords     (#PCDATA)>
```

Figure 1: INEX topics format

We decided to consider only the <Title> and <Keywords> tags of the topic and ignore the <Description> and the <Narrative> ones.

3.1 CO topics translation

For CO topics we systematically applied the following translation rules:

- If there is only one word under the <cw> tag, we add it to the query with an implicit +, together with the words under the <Keywords> tag.
- If there are only two words under the <cw> tag, we add them to the query with an implicit phrase augmented with a + operator, together with the words under the <Keywords> tag.
- If there are more than 2 words under <cw> we simply add them to the query and ignore the <Keywords> part.

In the first two cases, we are guaranteed that result candidates will contain the words under <cw> (via the + operator) and adding the words under the <Keywords> part simply improves ranking. In the last case, we do not add the keywords, since the query is long enough to be expressive in itself and since we want to gurantee that the results contain at least some of the <cw> decorated words. The words under the <Keywords> tag may add noise, therefore we ignore them.

3.2 CAS topics translation

For CAS topics we applied similar rules as for the CO topics as follows:

- For each <cw><ce> pair:
    - If there is only one word under <cw>, we add it to the query with an implicit + under all nodes that appear in the <ce> tag
    - If there are only two words under <cw>, we add them to the query with an implicit phrase augmented with a + operator under all nodes that appear in the <ce> tag
    - If there are more than two words under <cw> we add them to the query under all nodes that appear in the <ce> tag
- For <cw> without a <ce> tag we apply the CO rules as described above.
- We add the words under the <Keywords> part to the query as free text

For example, lets consider the INEX **topic 5,** as expressed in Figure 2 below:

```
<Title>
    <te>tig</te>
    <cw>QBIC</cw><ce>bibl</ce>
    <cw>image retrieval</cw>
</Title>
<Keywords>
QBIC, IBM, image, video, content query, retrieval system
</Keywords>
```

Figure 2: INEX topic 5

According to the above rules, it is translated into:

```
<bibl>+QBIC</bibl>
+"image retrieval"
QBIC. IBM. image. video. "content query" . "retrieval system"
```

We assume some knowledge of the semantics of the INEX documents DTD and systematically apply the "at least one" rule for "years" and "authors" elements, as illustrated in topic 15 (see Figure 3).

```
<Title>
   <te>article/bm/bib/bibl/bb</te>
   <cw>
    hypercube, mesh, torus, toroidal,
    non-numerical, database
   </cw>
   <ce>article/bm/bib/bibl/bb</ce>
   <cw>1996 or 1997</cw>
   <ce>article/fm/hdr/hdr2/pdt</ce>
</Title>
<Keywords>
   1996 1997 hypercube mesh torus toridal
   non-numerical database
</Keywords>
```

Figure 3: INEX topic 15

This topic is translated into the following fragment form:

```
<article>
   <bm><bib><bibl><bb>
      hypercube. mesh. torus. toroidal. non-numerical.
      database.
   </bb></bibl></bib></bm>
  <fm><hdr><hdr2>
    <pdt>+1996</pdt>
    <pdt>+1997</pdt>
  </hdr2></hdr> </fm>
</article>
1996 1997 hypercube mesh torus toridal non-numerical
database
```

Note that according to our syntax, result candidates need to contain at least one of the years 1996 or 1997.

3.3 Limitations of our format

The proposed XML Fragments format is clearly not as expressive as a full-fledged SQL-like query language. However, our conjecture is that it covers most of users needs in querying XML collections and reduces significantly the complexity of the language. This is similar to free-text queries that provide less expressive power than complex Boolean queries, but provide sufficient expressiveness for most users' needs. We verified this hypothesis in the INEX evaluation, as we could easily express 58 out of the total 60 INEX topics.

We could not express Topic 14, which states "*Find figures that describe the Corba architecture and the paragraphs that refer to those figures*". This type of query requires a kind of "join" operation between two elements (or tables in database terms) "figures" and "paragraphs" which should be joined through a common "figure-id" field.

Another Topic that we could not express using our XML fragments was Topic 28, which states "*Retrieve the title of articles published in the Special Feature section of the journal 'IEEE Micro'"*. This topic depends on the order of sibling nodes (journals are built from <sec1> nodes followed by <article> nodes that belong to that section). Our query format is expressed as an XML tree and thus cannot express relations that depend on node ordering. We could express topic 28 if the <journal> was organized such that <article> nodes are children of <sec1> nodes, as specified below:

```
<journal>
    <title>…</title>
    <sec1>
        <title>…</title>
        <article>…</article>
        <article>…</article>
    </sec1>
</journal>
```

4. RANKING APPROACHES

In this section we discuss two possible approaches for combining the structured and unstructured portions of the query in terms of ranking Let us remind here that a typical ranking model for IR is the vector space model where documents and queries are both represented as vectors in a space where each dimension represents a distinct indexing unit t_i. The coordinate of a given document D on dimension t_i, is denoted as $w_D(t_i)$ and stands for the "weight" of t_i in document *D* within a given collection. It is typically computed using a score of the *tf x idf* family that takes into account both document and collection statistics. The relevance of the document D to the query Q, denoted below as $\rho(Q,D)$, is then usually evaluated by using a measure of similarity between vectors such as the cosine measure (Formula 1).

$$\rho(Q,D) = \frac{\sum_{ti \in Q \cap D} w_Q(t_i) * w_D(t_i)}{\|Q\| * \|D\|}$$

Formula (1)

We describe now two ranking methods for XML documents: one that weights each individual context and one that merges all contexts that match a query term. We have tested the two ranking methods in two different INEX runs and will use the INEX assessment results to verify which method is better.

4.1 Assigning weights to individual contexts

The first approach, which extends the vector space model, is described in details in [4]. The idea is to use as indexing units not single terms but pairs of terms of the form *(t_i,c_i),* where t_i is the textual part or term and c_i is the path leading to it from the document root (the context). We allow "approximate matching" so that a term *(t_i,c_i)* in the query can match several actual terms of the form *(t_i,c_k)* in the documents. For example, a query term *(John, /author)* can match *(John, /fm/author/fnm)* and *(John, /bm/author/fnm).* For each query term *(t_i,c_i)*, we denote its weight in the query as $w_Q(t_{i,}c_i)$, the weight of each resembling context in the documents as $w_D(t_{i,}c_k)$, and the resemblance measure between the contexts as *$cr(c_i,c_k)$* (see an example *cr* function in Section 6.1).

Thus, in order to measure the similarity between XML fragments and XML documents we extend (Formula 1) to (Formula 2) below:

$$\rho(Q,D) = \frac{\sum_{(ti,ci) \in Q} \sum_{(ti,ck) \in D} w_Q(t_{i,}c_i) * w_D(t_i,c_k) * cr(c_i,c_k)}{\|Q\| * \|D\|}$$

Formula (2)

We impose that *cr()* values range between 0 and 1, where 1 is achieved only for a pair of perfectly identical contexts. Thus, we see that (2) is identical to (1), in the special case of free-text where there is only one unique default context.

4.2 Merging contexts

Recall that for each query term *(t_i,c_i)*, we can find a set of document terms *(t_i,c_k)* such that each c_k resembles the given context c_i.. As an alternative approach, instead of weighting the resemblance between c_i and all its *c_k's*, we consider merging all occurrences of t_i under all such *c_k's* and treating them as equally good from the user's perspective. The merged context is assigned a weight as a function of the details the user gave in her query, which is independent of the distance between the query context and the document contexts. Denoting $w(c_i)$ as the weight of the context c_i (see an example function in 6.2), our ranking formula becomes:

$$\rho(Q,D) = \frac{\sum_{(t_i,c_i)\in Q} w_Q(t_i) * w_D(t_i) * w(c_i)}{\|Q\| * \|D\|}$$

Formula (3)

5. IMPLEMENTATION – THE JuruXML SYSTEM

We have extended a full-text information retrieval system Juru [3], developed at the IBM Research Lab in Haifa so as to support the XML fragment query format and the above ranking mechanisms. We describe now the modifications we applied, for this purpose, to the indexing and to the retrieval processes.

5.1 Indexing stage

At indexing time, XML documents are parsed using an XML parser. A vector of *(t,c)* pairs is extracted to create the document profile where *t* is the textual part or term and *c* is the path leading to it from the document root (i.e., the context). In addition we store for each XML tag <*tag*> a pair *(_s_.tag, c)* for the tag start and *(_e_.tag, c)* for the tag end with *c* the path leading to the *tag*. By storing terms with their contexts, the posting-list of term *t* that encapsulates all occurrences of *t* in all documents, is split into separate posting lists, one posting list for each of the contexts in which *t* occurs. This splitting allows the system to efficiently handle retrieval of occurrences of a term *t* under a specific context *c*. For efficiency we map each context to a contextId, which can be stored as an integer.

We use a scheme first introduced in [1], for navigating XML collections and implemented in the XMLFS system that allows to store such pairs *(t,c)* in the lexicon of a regular full-text information retrieval system via only minor modifications: each pair *(t,c)* is presented to the indexer as a unique key *t#c*. At retrieval time, the system can identify the precise occurrences of the term *t* under a given context *c* in the collection, by fetching the posting list of the key *t#c*. Juru [3] stores all index terms (that form the lexicon of the system) in a *Trie* data structure (see for example [8]) and therefore all contexts under which the term *t* has been stored can easily be retrieved by suffix matching of "*t#*"

5.1.1 *Component statistics*

As described in the previous section, the terms we store in the index are of the form *t#c* where *t* is a word and *c* is the context leading to the term from the document root. This allows us to query for content under a specific context and to return a specific component as a result. However, Juru[3] tracks statistics (e.g., term frequency) at the document level, therefore relevance can be evaluated only at the document level. This means that all components in a retrieved document will be assigned the same relevance score and thus the same ranking (namely the document's ranking).

In order to allow ranking at a granularity level other than the full document level, it is possible to define at indexing time a list of elements whose associated fragments will be indexed as separate entities. This allows for statistics to be tracked at the indicated level of granularity, and to score results at the same granularity. While this approach works well for CO like queries, it does not perform as well for queries that specify a combination of contexts since these contexts may reside in different indexing entities.

In future work we investigate how to support various levels of granularity in one index based on ideas taken form [5, 6]. In the meantime, for the INEX collection, we used a fixed granularity of <sec> for CO topics.

5.2 Retrieval stage

As described above, the query is expressed as a combination of XML fragments and possibly free text. In order for queries to be expressed as valid XML, we encapsulate the query within a pair of <root></root> tags, which have no semantic meaning and are removed at a later stage. We parse queries with a standard XML parser in order to obtain a set of terms in context of the form *t#c*, in the same way as we parsed the original XML documents. The retrieval algorithm is described below:

1. Parse the query and create a list of terms of the form $t_i\#c_i$
2. Expand each term $(t_i\#c_i)$ to relevant terms $(t_i\#c_k)$ that resemble it from the index (see Section 5.2.1)
3. Issue a regular Juru query formed by the expanded terms
4. Rank results according to one of the methods described in Section 4.
5. Filter results based on the query tree structure (see section 5.2.2)

Figure 4: Retrieval algorithm

We detail each of the key steps of the algorithm in the following sections.

5.2.1 *Query expansion*

Let us illustrate the expansion with the example below. Consider the query:

<bibl>QBIC</bibl>

It is parsed into "qbic#/bibl". We execute suffix matching (thanks to the trie structure) on "qbic#" and get all the contexts under which the word qbic was indexed. An example of such a context is "/article/bm/bib/bibl/bb". We now have to check which of them is relevant to the query.

In our current implementation, we consider only the contexts for which the query context is a subsequence. Therefore, "/article/bm/bib/bibl/bb" is a relevant context since it includes "/article/bibl" as a subsequence. Note that we allow for gaps in the inclusion. At the end of this step we have a set of terms of the form *t#c*, which are now sent to Juru as a free text query.

5.2.2 *Result filtering*

The retrieval process could potentially assign a non-zero score to any document containing parts of the query based on the selected scoring function. While we want such matches to contribute to the score, we also wish to assure that the documents conform to the well-specified parts of the query. This is achieved by post-filtering

This filtering is handled as follows. A "tree" representing the XML fragments associated with the query is created to represent the logical structure of the query. Each node in the tree corresponds to a single query term (either a content or context term). For each document that was assigned a non-zero score by our scoring model, we extract the query term's instances together with their offsets in the document (as stored in the index). We then confirm that the constraints imposed by the query tree hold in the specific document. This includes constraints imposed by +/- operators as well as instance level constraints. (For example for the query <+author>John Doe</author> the filtering verifies that only documents that contain both John and Doe under the same <author> instance are returned).

The filtering process is also responsible for filtering the required target elements (*te*) as defined by the user (see section 2.1.1 above). If there are no target element defined then the whole document is returned. Otherwise we return all *te*'s instances that satisfy the query constraints (or all *te* instances if there are no query constraints on the *te* – e.g. return all <author> of articles with <title>databases</title> from <yr>2002</yr>)

6. INEX RUNS

We conducted three INEX runs. For the first two runs, we applied the automatic query translation rules specified in section 3 above, while in the 3rd run we performed some manual editing of the query attempting to better fit the topic's <Description>.

6.1 First run – assigning weights to individual contexts

In the first run we employed the ranking method of formula (2) using the following context resemblance function

$$cr(c_i, c_k) = \begin{cases} \dfrac{1+|c_i|}{1+|c_k|} & c_i \ subsequence \ of \, c_k \\ 0 & otherwise \end{cases}$$

where $|c_i|$ is the number of tags in the given query context and $|c_k|$ is number of tags in the expanded context. Thus, for example,

cr("/article/bibl", /article/bm/bib/bibl/bb") = 3/6 = 0.5

It is easy to see that $0 < \boldsymbol{cr} \leq 1$ and it is equal to 1 if and only if the query context is identical to the expanded context. For CAS topics this run was ranked 4th with Av. Precision 0.320 (see figure 5 below).

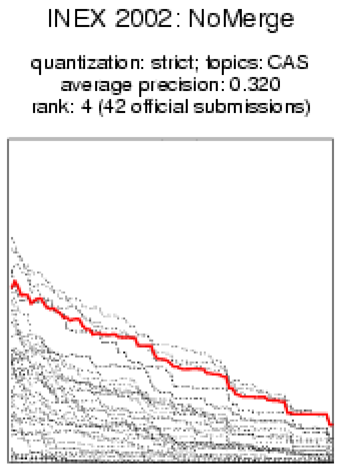

Figure 5 – individual weights

6.2 Second run – merging contexts

In the second run, we employed the ranking method of formula (3) where the weight function for context *c* was

$$w(c_i) = (|c_i| + 1)$$

For example, the weight of the context in the query term "qbic#/bibl" is 2. For CAS topics this run was ranked 2nd with Av. Precision 0.352 (see figure 6 below)

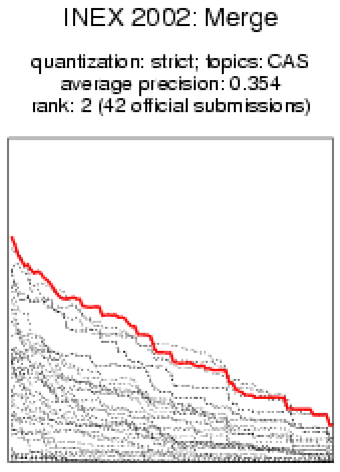

Figure 6 – CAS topics merge contexts

This result shows that merging contexts yields better results then the approach tested in the first run. In section 6.4 we analyze the reasons for this behavior.

For CO topics this run was ranked 10 with Av. Precision 0.053. As described above we didn't have dynamic component level statistics and for the CO topics we returned either the whole article or a sec. We expect that with dynamic component statistics we will achieve much better results.

6.3 Third run – manual editing

In this run we tried to exploit our query format capabilities by manual editing some of the queries based on their description. Let us consider for instance topic 18 as given in Figure 7.

```
<Title>
   <te>article</te>
   <cw>Hypertext Information Retrieval</cw>
   <ce>article</ce>
   <cw>Hypertext Information Retrieval</cw>
   <ce>bib/bibl/bb/atl</ce>
</Title>
<Description>
Retrieve   articles   on   hypertext   information
retrieval  where  the  bibliography  contains  works
with the words "hypertext",      "information" and
"retrieval" in at least one of the citations.
</Description>
```

Figure 7: INEX topic 18

This topic was translated for the first two runs into:

```
<article>
   Hypertext Information Retrieval
</article>
<bib><bibl><bb><atl>
   Hypertext Information Retrieval
</atl></bb></bibl></bib>
```

While it was expressed, in the third manual run as

```
<article>
    Hypertext Information Retrieval
</article>
<+bib><bibl><bb><atl>
   Hypertext Information Retrieval
</atl></bb></bibl></bib>
```

The only difference between these expressions is that in the latter form, a <+bib> is added in order to force all three words *Hypertext Information Retrieval* to appear under some same instance of a <bb> tag. The manual run returned only 5 such results, while the first 2 runs returned 100 results most of them containing only some of the required words under the same <bb> item. This run was ranked 3rd in the CAS topics.

6.4 Comparing the Runs

We compare here the first 2 runs ignoring the manual run. We achieved quite good results for the CAS topics and average results for the CO topics. Since for the INEX runs we didn't have dynamic component level statistics we didn't expect good results for CO topics. Instead we focus on the CAS topics and by looking at the first 2 runs it turned out that the approach that merges context gave better results then the approach that weights contexts by their resemblance to the user query context. This can be explained by looking at formula 2 where $W_X(t,c)$ is defined as -

$$W_X(t,c) = tf_X(t,c) * idf(t,c)$$

where x stands for either D or Q and

- $tf_x(t,c)$ is a monotonic function of the number of occurrences of (t,c) in x.
- $Idf(t,c) = \log(|N|/|N_{(t,c)}|)$ with $|N|$ = total number of documents in the collection and $|N_{(t,c)}|$ = number of documents containing (t,c)

Since in formula 2 each term t is split into different contexts (t,c_k) it might happen that a given (t,c_k) would receive a very high *idf* value because (t,c_k) is very rare in spite of t being very common. In future work we investigate how to compensate for this behavior.

6.5 Generating the submission format

An INEX submission consists of a number of topics, each identified by a topic ID. A topic's result consist of a number of result elements as in the example below (we omit full format due to space limitation. It can be obtained from [7])

```
<result>
    <file>tc/2001/t0111</file>
    <path>/article[1]/bm[1]/ack[1]</path>
    <rsv>0.67</rsv>
</result>
```

In JuruXML a match is identified by its offset in the document. To generate the above format we parse again the XML document that contains the match and while counting offsets until the match's offset we build the requested <path> info.

7. CONCLUSION AND FUTURE WORK

The INEX framework allowed us to experiment with the expressiveness of the XML fragments query format. We showed that using, this rather simplistic query format, we could express 58 out of the 60 INEX topics. We then presented two ranking methods that combine IR ranking for free text with XML structure ranking. One approach assigns different weights to term occurrences under different contexts and the other merges all occurrences of document terms that match a query term. We achieved very good results on the CAS topics where the first run was ranked 4th and the second run was ranked 2nd among all INEX submissions.

In a following work we further investigate more models of structure ranking by introducing different *Context Resemblance* functions. We also investigate different levels of context merging that cover the scale between no context merging at all to the full context merging models that were presented in this paper. For CO type topics we investigate a dynamic component level statistics that should allow to select the most relevant component when target elements are not defined.

8. REFERENCES

[1] A. Azagury, M.Factor, Y. Maarek, B. Mandler "A Novel Navigation Paradigm for XML Repositories", pp 515-525 in ACM SIGIR'2000 workshop on XML and IR, SIGIR Forum, 2000.

[2] R. Baeza-Yates, N. Fuhr and Y. Maarek, *Second Edition of the XML and IR Workshop,* In SIGIR Forum, Volume 36 Number 2, Fall 2002

[3] D. Carmel, E. Amitay, M. Herscovici, Y. Maarek, Y. Petruschka and A. Soffer, "Juru at TREC 10 - Experiments with Index Pruning", In [2].

[4] D. Carmel, N. Efraty, G. Landau, Y. Maarek, Y. Mass, "An Extension of the Vector Space Model for Querying XML Documents via XML Fragments" In XML and Information Retrieval workshop of SIGIR 2002, Aug 2002, Tampere, Finland.

[5] N. Fuhr and K. GrossJohann, "XIRQL: A Query Language for Information Retrieval in XML Documents". In *Proceedings of SIGIR'2001*, New Orleans, LA, 2001

[6] T. Grabs and H. J. Schek, "Generating Vector Spaces On-the-fly for Flexible XML Retrieval", in [2].

[7] Initiative for the evaluation of XML retrieval http://qmir.dcs.qmul.ac.uk/INEX/

[8] Donald E Knuth, The art of computer programming: sorting and searching (vol 3), Addison Wesley, 1973.

[9] XPath – XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20/

[10] XQuery – The XML Query language, http://www.w3.org/TR/2002/WD-xquery-20020430