Retrieving the most relevant XML Components

Yosi Mass, Matan Mandelbrod
IBM Haifa Research Lab
Haifa 31905, Israel
+972-3-6401627
{yosimass, matan}@il.ibm.com

ABSTRACT
XML enables to encode semantics in full text documents through XML tags. While query results on corpora of full text documents is typically a sorted list of ranked documents, this granularity can be refined to return sub components when searching over XML documents. In this paper we describe an approach for finding the most relevant XML components for a given query.

Keywords
XML Search, Information Retrieval, Vector Space Model

1. INTRODUCTION
XML documents represent a family of semi-structured documents in which data has some structure but is not fully structured as in databases. It is thus not surprising that approaches for searching in collections of XML documents are either extension of Information Retrieval (IR) techniques or of database query languages. The main difference between the two approaches is that while the results of Information Retrieval techniques is a list of documents sorted by their relevance to the query, the results of a database query are strict matches with no relevance values. In this paper we focus on Information Retrieval approaches and explore a technique whereby we rank individual XML components rather than full documents.

The Initiative for the evaluation of XML Retrieval (INEX) [7] coined two types of queries over XML documents: In **Content Only (CO)** queries the user has no knowledge of the document structure and the search engine is supposed to return the best components that match the query concepts. In **Content and Structure (CAS)** queries the user has some knowledge of the document structure and can use it to constrain content to a specific structure and also to specify the XML components to be returned.

It should be noted that techniques that are suited for returning a specific XML component that matches a CAS query may be orthogonal to the task at hand, which requires that the best matching component be retrieved. Indeed the version of JuruXML[11] that we used in INEX'02[7] could retrieve XML components as specified by CAS topics yet it could score only full documents. Consequently, all relevant components in a retrieved document where assigned identical scores – the score of their enclosing document, and individual component ranking was not supported.

In modern Information Retrieval engines document ranking is done based on the vector space model[13]. The idea is to treat both the documents and the query as a vector of terms (typically words). Each term is given a weight proportional to its Term Frequency (TF) in a document/query and inversely proportional to its Document Frequency (DF), which is the number of documents in which the term appears. The similarity between a document and a query is defined as the distance between the two vectors usually measured as the cosine between the two.

In order to rank components rather than entire documents, this classic model must be expanded to take into account component level statistics. The problem is that components in XML documents are nested and this hierarchy needs to be taken into account when counting term occurrences. More specifically, a specific term should not be counted more than once. For example consider a term inside a paragraph, which is itself nested in a section. What is the component frequency of this term? If it is counted as belonging to two components, it may distort ranking since the term actually appears only once in the document. On the other hand, if it is counted only once, with which component should this count be associated?

In this paper we describe an extension to the classic vector space model that can correctly handle retrieval at the component level. We demonstrate the use of this method on the INEX topics. This method can be implemented as an extension of any vector space based text search engine with no need to modify its basic structures and algorithms, making it highly applicable for any search engine wishing to rank components.

The remainder of the paper is organized as follows: In section 2 we outline some related work. In section 3 we describe our novel approach for selecting the most relevant XML component and how it was used for the INEX CO topics. In section 4 we show how this method was extended to handle the CAS topics. Our method for result clustering and for filtering redundant components is described in Section 5. We conclude with summary and future work.

2. RELATED WORK
The idea of ranking document subcomponents has been explored in the context of *passage retrieval* [10],[14]. The goal there is to identify the sentences that best match the user's query and assemble them into passages that are then returned to the user. The returned unit can be any combination of sentences even if they are inconsecutive. This technique is not suitable for XML components retrieval where the returned unit must be a fixed XML component.

The work described in [12] tries to identify subject boundaries in a text document based on the assumption that words that are related to a certain subject will be repeated whenever that subject is mentioned. Again this work assumes a flat text document with freedom to pick a portion of the text as an answer. This is not

suitable for XML retrieval where the retrieval unit must be a predefined XML component.

The idea of scoring XML components separately has been suggested in the context of XML retrieval [5] [6]. In both cases, the term and document frequency is accumulated at the basic component level. An augmentation factor is used to propagate statistics from child to parent components. The problem with this technique is that the augmentation factors are either set manually by the user or set empirically and thus cannot be proven to give the best results.

3. APPROACH FOR CO TOPICS

We start by describing our approach for Content Only (CO) tasks and then we show how this approach was extended to handle Context and Structure (CAS) topics as well.

As a reminder, in a CO task the query is specified in full text (with additions of +/- and phrases) and the search engine is expected to return the most relevant XML components that match the query concepts.

Based on a training set composed of the INEX'02 topics and assessments, we found that the majority of the highly ranked components for CO topics (1296 out of 1394) were taken from the set: {article, bdy, abs, sec, ss1, ss2, p and ip1}. This is quite intuitive since {sec, ss1, and ss2} stand for sections and sub sections and {p, ip1} represent meaningful paragraphs, all good reasonable results for a query. The entire article or its abstract {abs} are also good candidates for component retrieval. The only exception is the {bdy} component that constitutes the main part of the article so whenever a bdy is relevant so is its containing article and vice versa. In that case we rather return the article and not the body.

Realizing that we have a clear list of candidate components for retrieval, our goal was to modify JuruXML[11] so that it could rank each of these candidate components separately. The ranking method used in JuruXML is based on the Extended Vector Space Model[3] where both documents and queries are represented as vectors in a space where each dimension represents a distinct term. It is typically computed using a score of the *tf x idf* family that takes into account the following document and collection statistics -

- **N** - Total Number of documents in the collection
- **Term Frequency** $TF_D(t)$ – number of occurrences of a term *t* in a document *D*
- **Document Frequency** DF(t) – total number of documents containing a term *t*

The relevance of the document D to the query Q, denoted below as $\rho(Q,D)$, is then evaluated by using a measure of similarity between vectors such as the cosine measure (see Formula 1).

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

Formula 1

Where

$$w_{X \in \{Q,D\}}(t) = \log(TF_X(t)) * \log(\frac{N}{DF(t)})$$

Formula 2

It follows that the weight WD(t) is proportional to the number of occurrences of t in D ($TF_D(t)$) and inversely proportional to the number of documents in which t appears (DF(t)). The motivation is that a term t that appears in a few documents in the corpus, should contribute a relatively high weight to the score of a document in which it appears compared to terms that are frequent in many documents. The contribution to the document score is additionally proportional to the number of its occurrences in the document.

In order to rank components instead of entire documents, these statistics should be tallied at the component level. That is, it is necessary to keep track of the following component and collection statistics:

- **N** - Total Number of components in the collection
- **Term Frequency** $TF_C(t)$ – number of occurrences of a term *t* in a component *C*
- **Component Frequency** CF(t) - total number of components containing a term *t*

The problem is that XML components are nested. For example consider a collection consisting of a single document (see Figure 1).

```
<article>
      t1
      <sec>
          <p>t2</p>
      </sec>
</article>
```

Figure 1

The document contains three components {C_1=article, C_2=sec, C_3=p} and two terms {t_1, t_2}. Term t_1 appears only in the article while t_2 appears in all 3 components. Therefore we get

- N = 3
- $CF(t_1) = 1$, $CF(t_2) = 3$
- $TF_{C1}(t_1) = 1$, $TF_{C1}(t_2) = 1$
- $TF_{C2}(t_1) = 0$, $TF_{C2}(t_2) = 1$
- $TF_{C3}(t_1) = 0$, $TF_{C3}(t_2) = 1$

By Formula 2 applied to component level statistics, we would get that $W_{c1}(t_1) > W_{c1}(t_2)$ which is not necessarily true since both t_1 and t_2 appear an equal number of times in the document.

One can try to fix this by only counting the Term Frequency $TF_C(t)$ at the component level and still computing N & DF(t) at the document level. However, this imposes another problem that is illustrated in the following example (see Figure 2).

```
<article>
    <sec>t1</sec>
    <sec>t1</sec>
    <sec>t2</sec>
</article>
```

Figure 2

As before, the collection consists of a single document and we have

- $N = 1$
- $DF(t_1) = 1$, $DF(t_2) = 1$

If we mark the 3 sections by C_1, C_2, C_3 we get

- $TF_{C1}(t_1) = 1$
- $TF_{C2}(t_1) = 1$
- $TF_{C3}(t_2) = 1$

By Formula 2 it follows that $W_{c1}(t_1) = W_{c2}(t_1) = W_{c3}(t_2)$. However if we regard each section as a standalone component then since t_2 appears only in one section while t_1 appears in 2 sections we expect to get $W_{c1}(t_1) < W_{c3}(t_2)$ (which is what would have happened if the sections were in different documents, since we would then have then $DF(t_1) = 2$). With this approach to counting statistics it is thus impossible to differentiate between the rankings of the three sections.

In view of the above problems, we selected a strategy whereby we create a different index for each component type. Statistics can thus be tallied at the precise level of granularity for each component. In particular, we created six indices corresponding to the following tags: {article, abs, sec, ss1, ss2, p, and ip1[1]}. The *article* index contains the full data of all documents. The *sec* index contains each sec from each article as a separate document and so on for each of the six tags above. For example the document in Figure 2 above will result in 3 separate documents in the *sec* level index.

For each index, the entities are determined according to the topmost XML tag of the corresponding type. That is, nested components of the same type do not yield a new partition of the document. For example consider a document as in Figure 3

```
<Article>
    <sec>
        <p>some text
            <p>some internal text</p>
        </p>
    </sec>
    <p>some higher level text</p>
</article>
```

Figure 3

This document will add two "documents" to the paragraph level index (See Figure 4 & Figure 5)

```
<p>some text
    <p>some internal text</p>
</p>
```

Figure 4

[1] P and IP_1 were indexed into one Index

And

```
<p>some higher level text</p>
```

Figure 5

The search engine's regular ranking formula can now be used to accurately score and rank individual components among themselves. In other words, given a query, the system can return the best matching articles, sections, sub-sections, etc. Our goal however is to return one ranked list of the best matching components regardless of granularity and thus need to compare scores from the individual indices. To achieve this, the query is submitted in parallel to each index, resulting in six sorted lists of components – one from each index.

The scores in each index are normalized into the range (0,1) using a formula that ensures that this normalization yields absolute numbers and is index independent. This is achieved by each index computing *P(Q,Q)* (see Formula 1) which is the score of the query itself as if it was a document in the collection. Since the score measures the cosine between vectors, then the max value is achieved between two identical vectors. Each index therefore normalizes all its scores to its computed *P(Q,Q)*. The normalized results are then merged into a one ranked list consisting of components of all granularities.

It should be noted that this approach can be implemented on top of almost any full text ranking engine resulting in a system than is able to rank XML components without modifying the core search engine code. It simply requires an XML parser that can parse documents and feed the components into separate indexes. At run time, queries are submitted in parallel to each index and the results are merged as described above.

3.1 The CO runs

We now describe the implementation of this method on the INEX collection. The size of the collection is ~500Mb. Six indices were created as described above, resulting in the following index sizes:

- Article – 290Mb
- Sec – 270Mb
- Ss1 – 158Mb
- Ss2 – 38Mb
- P, ip1 – 280Mb
- Abs – 14Mb

Overall we get an index size that is about twice as large as the original collection. While this can be an inhibiting factor, our goal was to prove the viability of this method from a quality standpoint. We believe there is room for optimisation in terms of index sizes.

We submitted three CO runs. Recall that a CO topic consists of full text with additions of +/- and Phrases. According to the topic development guide [8] the +/- "should be interpreted with a fuzzy flavour and not simply as must contain and must not contain conditions". We applied this vagueness to "+" terms but still we believe that if the user specify a "-" term then this term should not be returned. Therefore we treated the "-" strictly (namely results that contain such terms were never returned). The runs we submitted were -

- In the first run we considered all query parts: Title, Description and Keywords (CO-TDK)

- In the second run we applied post clustering on the first run (see section 5 below) (CO-TDK-with-clustering)
- In the third run we considered only the Title. In this run we ignored phrases and treated the phrase terms as regular words. We applied the clustering algorithm on this run as well (see Section 5 below) (CO-T-with-clustering)

The recall precision results achieved for the above runs based on assessments version 2.4 and using the "Strict with Overlap" metric are summarized in Table 1 below. Strict means that only highly assessed elements are considered and overlapping means that the metric removes overlapping results and penalizes submissions that return redundant fragments.

	CO-TDK	CO-TDK-with-clustering	CO-T-with-clustering
P@5	0.42	0.41	0.42
P@10	0.38	0.36	0.30
P@20	0.35	0.29	0.26
P@100	0.24	0.21	0.15
P@1500	0.162	0.142	0.129

Table 1

The table shows results at several precisions. It is quite clear that the first two runs which included all topic parts (title, keywords and description) were superior to the third run which used only the topic's title. Between the two TDK runs the one without the clustering performed better. This result is discussed in Section 5 below.

4. APPROACH FOR CAS TOPICS

The Content and Structure (CAS) topics differ in two aspects from the CO topics. First the query content can be limited to a given XML tag and second there is less freedom in selecting the component to be returned. The topic format is XPath[15] augmented with an 'about' predicate[8]. The last component in the path specifies the component that should be returned.

For example topic 66 (Figure 6) defines a constraint on the year <yr> and on section <sec>. <sec> is also the element to be returned.

```
/article[./fm//yr < '2000']
        //sec[about(.,'"search engines"')]
```

Figure 6

To enable fuzziness in the query constraints we introduce a *Synonyms* mechanism. We divide the XML tags into synonym groups such that all tags in the same group are regarded equivalent. Whenever there is a tag in the query that belongs to some synonym group, we substitute it by all tags in its group.
For example if we set {sec, ss1} to be in the same synonym group then in the query in Figure 6 above we substitute *sec* by {sec, ss1} and we get[2] the query in Figure 7. The synonyms mechanism is used at different granularity levels for the SCAS and VCAS runs (see below).

```
/article[./fm//yr < '2000']
        //{sec,ss1}[about(.,'"search engines"')]
```

Figure 7

In order to find documents in which all of the query constraints are met, we need to execute the modified query on the full documents. This will indeed return relevant components that match the query constraints, but as described above the components cannot be scored individually using only this one index.
Therefore we execute each query in two steps – in the first step we use the *article* index to locate candidates that fulfill the query constraints. In the second step, relevant parts of the query are extracted for each index (see example below) and the relevant query is submitted in parallel to the other five indexes of {abs, sec, ss1, ss2, p+ip1}. A relevance value is computed only for elements that were marked valid in the first step and a ranked list of results is returned. The separate lists are then merged similarly to what described for the CO case, resulting in one ranked list of results.
Note that although our indices do not cover all of the possible tags in the corpus, we can still resolve queries that request a tag that does not have a dedicated index. For example, topic 67 defines <fm> as the last component in the XPath expression, thus requesting a component for which we do not have a special index. In this case, we simply stop after the first step and use the article's score as the score of the component.

Example

In the following example we define one synonym group that consists of {sec, ss1, ss2} tags and we use it to run the query in Figure 6 above. We run the query first against the *article* level index and then we run the relevant query part on each of the synonyms ∈ {sec, ss1, ss2} so we run
//sec[about(.,'"search engines"')]
against the *sec* level index,
//ss1[about(.,'"search engines"')]
against the *ss1* level index and
//ss2[about(.,'"search engines"')]
against the *ss2* level index. We then merge the results based on their normalized scores as described above.

4.1 SCAS and VCAS

This year there were 2 CAS variants - Strict CAS (SCAS) and Vague CAS (VCAS). The SCAS defines that "structural constraints of a query must be strictly matched" while the VCAS defines that "structural constraints of a query can be treated as vague conditions". The vague means that XML elements that are "structurally similar" to those specified in the query can be returned. We used our synonym groups in different configurations to support both SCAS and VCAS.

For the SCAS runs we used the equivalent tags that were defined in the INEX topic development guide[8]. The synonyms we used were:

- {sec, ss1, ss2} for sections.

[2] This is not the syntax we use, the actual substitution is done in the internal implementation.

- {p, ip1} for paragraphs

The other two tags {article} and {abs} were not synonyms to any other tags so in topics that requested article or abs as results, only those tags were returned.

For the VCAS topics we defined one large synonym group that included all the tags {sec, ss1, ss2, p, ip1, abs}, except for the {article} tag. Again in topics that requested the article tag as a result we returned only articles.

4.2 The CAS runs

We submitted 3 SCAS and 3 VCAS runs. In all runs we treated the "-" strictly and the "+" with a fuzzy flavour. In all runs we treated query constraints in a strict manner up to the synonym tags. So for example results for the query in Figure 6 will be only sections and all their synonym tags that discuss "search engines" but only from articles that were published before year 2000.

We ran the following 3 runs for both SCAS and VCAS

- In first run, we considered all query parts: Title, Description and Keywords.
- In the second run, we applied a post-clustering algorithm on the first run (see Section 5 below).
- In the third run, we considered only the Title and Keywords and applied a post-clustering algorithm (see Section 5 below).

At the time this report is written full CAS results are not yet available so we don't report these results here.

5. RESULT CLUSTERING

The approaches described above may result in redundant components that are returned to the user. For example consider a section with four paragraph children. We can identify two extreme scenarios -

In the first scenario assume all four paragraphs are highly relevant to the topic. In this case all four paragraphs as well as their parent section will be ranked in high positions.

In the second scenario assume that only the first paragraph is very relevant to the topic and therefore it is assigned a high score. As a result it may also contribute to its parent's section score even if it is the only relevant paragraph in that section. Again that paragraph and its parent section may be ranked in a high position when merging the results.

One expectation of a good search engine is that it should not return redundant results; therefore in the first scenario it should return the section and not the paragraphs, while in the second scenario it should return the first paragraph only.

To filter such redundancies we developed a clustering algorithm that maps related components to one of the scenarios above. The algorithm gets the result set of the original run and constructs a tree consistent with the parent-child relationship of the components in the XML document. Each node in the tree corresponds to a result component and has the following data -

- Its score as a number between 0 and 1
- Total number of descendant children in the original document. This number is extracted while parsing the document.

The algorithm processes the tree bottom up and at each level compare the score of a node to that of its children. When it manages to identify one of the two scenarios above it remove the redundant components from the result set.

Recall that a score is a number between 0 and 1 so we need some means to say when two scores are close to each other. Let a node's score be *s1* and a child's score be *s2*. We say that the two scores are *close* if $\frac{|s1 - s2|}{s1} < ScoreThresh$ for some configured *ScoreThresh* value. Otherwise we say that s1 is *higher* than s2 (if s1>s2) or *lower* (if s1<s2).

The algorithm clusters each node into the following cases –

- **HighParent** - If the node's score is *higher* than all its direct children, then we remove the children from the tree.
- **HighChild** - If some child's score is *higher* than the current node's score, then we remove the node from the tree.
- **ManyDescendants** – Let N_e be the number of *close* descendants and N_t the number of all descendants of our node. If $\frac{N_e}{N_t} > ManyDescendantsThresh$ for some configured *ManyDescendantThresh* then we say that there are many good children and we remove the direct children from the tree (corresponding to the first scenario above)
- **SingleChild –** For each direct child C_i let N_i be the number of *close* descendants of C_i and N the total number of *close* descendants of the current node. If there is a child C_i with $\frac{Ni}{N} > SingleChildThresh$ for some configured *SingleChildThresh* value then we say that most good results are concentrated in that child so we remove the parent from the tree (corresponding to the second scenario)

For all other cases no filtering takes place, and all components are returned

5.1 Clustering runs

We used the following values for the clustering runs:

- *ScoreThresh*=0.45
- *ManyDescendantThresh* = 0.2
- *SingleChildThresh*=0.42.

According to the INEX evaluations received thus far, it seems that the runs that applied clustering received a lower overall score than runs that did not apply clustering. It thus seems that there was no penalty for runs returning redundant results. This topic should be discussed in order to devise metrics that evaluate a good overall result set, rather than individual results.

6. CONCLUSIONS AND FUTURE WORK

We presented a novel approach and implementation for scoring and ranking individual components of XML documents. At the time this report is written, Recall Precision graphs for the CO topics were published and one of our runs was ranked first indicating that this approach indeed computes more accurate component scores. The approach presented here can be implemented on top of almost any full text search engine without modifying its code to return ranked components for Content Only queries. Similarly the approach can be used by XML search engines to compute more accurate scores for target components specified in CAS topics. One limitation of our approach is that the set of potential components to be returned must be known in advance. We believe however, that this is a reasonable requirement for any given collection. Additionally, some space as well as runtime overhead is incurred by multi-indexing. Improving the efficiency is left for future research.

7. ACKNOLEDGMENT

We would like to thank Aya Soffer for reviewing the paper and for her useful comments.

8. REFERENCES

[1] 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

[2] D. Carmel, E. Amitay, M. Herscovici, Y. Maarek, Y. Petruschka and A. Soffer, "Juru at TREC 10 - Experiments with Index Pruning", In [1]

[3] 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.

[4] D. Carmel, Y. Maarek, M. Mandelbrod, Y. Mass, A. Soffer, Searching XML Documents via XML Fragments, SIGIR 2003, Toronto, Canada, Aug. 2003

[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] INEX'02 , Initiative for the Evaluation of XML Retrieval, http://qmir.dcs.qmul.ac.uk/inex/

[8] INEX'03 Topic development guide, http://inex.is.informatik.uni-duisburg.de:2003/internal/#topics

[9] Juru, A generic full text search engine, http://w3.haifa.ibm.com/afs/haifa/proj/docs/www/projects/km/ir/juru/index.html

[10] M. Kaszkiel and J. Zobel, "Effective ranking with arbitrary passages", Journal of the American Society of Information Science, volume 52, number 4, pg 344-364, 2001.

[11] Y. Mass, M. Mandelbrod, E. Amitay, D. Carmel, Y. Maarek, A. Soffer, JuruXML - an XML Retrieval System at INEX'02, Proceedings of the First Workshop of the Initiative for The Evaluation of XML Retrieval (INEX), 9-11 December 2002, Schloss Dagstuhl, Germany

[12] K. Richmond, A. Smith and E. Amitay, "Detecting Subject Boundaries within Text: A Language Independent Statistical Approach", Proceedings of the Second Conference on Empirical Methods in Natural Language Processing, pg 47-54, 1997

[13] G. Salton, Automatic Text Processing – The Transformation, Analysis and Retrieval of Information by Computer, Addison Wesley Publishing Company, Reading, MA, 1989.

[14] G. Salton, J. Allan, and C. Buckley. "Approaches to passage retrieval in full text information systems." In Proceedings of SIGIR'93, Pittsburgh, PA, 1993.

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

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