INDEXING AND QUERYING CONTENT AND STRUCTURE OF XML DOCUMENTS ACCORDING TO THE VECTOR SPACE MODEL

Jacques Le Maitre
LSIS – Université du Sud Toulon-Var
BP 20132, F-83957 La Garde
lemaitre@univ-tln.fr

ABSTRACT

This paper presents a method to index and query content and structure of XML documents according to the vector space model. Indexing is performed in three steps: (i) choosing content elements i.e. those which refer to the semantic content of the documents, (ii) associating a vector to each terminal content element whose components are the weights associated to each indexing term, (iii) propagating these vectors bottom up along the ancestors of the terminal content elements. Querying is performed with XQuery extended by adding to it a *vscore* function which returns the similarity degree between a query vector and a content element vector and by integrating into it the NEXI language which is a subset of XPath, defined in the framework of the INEX initiative that we have equipped with a fuzzy semantics.

KEYWORDS

XML, Information Retrieval, Vector Space Model, XQuery, Fuzzy Queries

1. INTRODUCTION

The database and the information retrieval communities are more and more interested in information retrieval in XML documents. A working group of the W3C has proposed a language that extends XQuery with full-text search capabilities (Amer-Yahia, 2005). At the same time, a query language, XIRQL, based on XPath, for information retrieval in XML documents has been proposed (Fuhr, 2001). A bit later, the DELOS Network of Excellence on Digital Libraries has started the "Initiative for the Evaluation of XML Retrieval" (INEX) (Fuhr, 2002), the aim of which is: "to provide means, in the form of a large XML test collection and appropriate scoring methods, for the evaluation of XML retrieval systems". INEX has proposed an XPath-like and easy to use query language, NEXI, for specifying queries on XML documents (Trotman, 2004).

The three main models of information retrieval (Baeza-Yates, 1999) are the Boolean model, the vector space model and the probabilistic model. Among them, the vector space model is actually the most used because of its relative simplicity and efficiency. In this paper, we propose a method based on the vector space model to index the different fragments (chapters, sections, paragraphs, captions…) of a set of XML documents, combined with an extension of XQuery to retrieve information in these fragments. This extension consists of adding a *vscore* function which returns the similarity degree between a query and a content element and integrating the NEXI language which is a subset of XPath, defined in the framework of the INEX initiative, that we have equipped with a fuzzy semantic.

This paper is organized as follows. Section 2 give some reminders about the vector space model. Section 3 explains how XML documents are indexed using the vector space model and section 4 explains how they can be queried with the extension of XQuery that we propose. Section 5 concludes.

2. THE VECTOR SPACE MODEL

In the vector space model (Baeza-Yates, 1999), documents and queries are represented by vectors in a space where each dimension is associated with an indexation term. Let T be a set of M terms $\{t_1, \ldots, t_M\}$ and D be a set of N documents, a document d is represented by a M-dimensional vector $\vec{d}$: $[w_{1,d}, \ldots, w_{M,d}]$ where $w_{i,d}$ is the weight of the term t_i in the document d. Several formulas have been proposed to compute this weight. They combine three factors: local weighting, global weighting and normalization. The local weighting is an increasing function of the frequency of a term in a document. It is based on the supposition that a term is all the more relevant to a document as it appears frequently in this document. The global weighting is a decreasing function of the frequency of a term in the set of documents. It is based on the supposition that a term is all the less relevant to a document as it is more frequent in the set of documents. Finally, normalization is aimed to compensate the length differences between the documents: a term has more chance to be frequent in a long document, without being more significant for that.

The most classical formula to compute the weight $w_{i,d}$ of a term t_i in a document d is (1):

$$w_{i,d} = tf_{i,d} \times idf_i \tag{1}$$

In this formula, $tf_{i,d}$ is the normalized frequency of the term t_i in the document d and idf_i is the inverse of the frequency of the term t_i in D. They are calculated by formulas (2) and (3):

$$tf_{i,d} = \frac{freq_{i,d}}{\max_{k=1,M} freq_{k,d}} \tag{2}$$

$$idf_i = \log(\frac{N}{N_i}) \tag{3}$$

where $freq_{i,d}$ is the frequency of the term t_i in the document d and N_i is the number of documents indexed by the term t_i. The three factors mentioned above are used in these formulas: $tf_{i,d}$ is the local weighting, idf_i is the global weighting and normalization occurs at two levels: division by the maximum frequency in formula (2) and attenuation of the inverse document frequency by the *log* function in formula (3).

A query q is also represented by a M-dimensional vector $\vec{q} = [w_{1,q}, \ldots, w_{M,q}]$ where $w_{i,q}$ is the weight of the term t_i in the query q. Usually, $w_{i,q} = idf_i$ if documents described by the term t_i are to be retrieved, and $w_{i,q} = 0$ otherwise. Several formulas to calculate the degree of similarity between a document d and a query q have also been proposed. The most classical one consists in measuring this similarity by the cosine of the angle of the vectors $\vec{d}$ and $\vec{q}$ (4):

$$sim(d, q) = \frac{\vec{d} \bullet \vec{q}}{|\vec{d}| \times |\vec{q}|} = \frac{\sum_{i=1}^{M} w_{i,d} \times w_{i,q}}{\sqrt{\sum_{i=1}^{M} w_{i,d}^2} \times \sqrt{\sum_{i=1}^{M} w_{i,q}^2}} \tag{4}$$

The motivation behind this formula is that a document answers even better to a query since the angle of their respective vectors is small. Let us remark that using this formula, the normalization of the term frequency by the maximum frequency ($\max_{k=1,M} freq_{k,d}$) in formula (2) is useless because this maximum frequency disappears in the division.

3. VECTORIAL INDEXATION OF XML DOCUMENTS

The problem of indexing the content of a set of XML documents taking in account their structures actually receives particular attention, especially in the framework of the INEX initiative (Fuhr, 2002, 2003 and 2004). All classical information retrieval models are being experimented and among others, the vector space model.

The first problem is to choose the unit of indexation, which no longer can be the document because its structure must be taken in account, and therefore its division into logical units. In general, not all the components of an XML document are important for information retrieval. It is the case, for example, for

```
<? xml version=1.0 ?>
<anthology>
<article year="2005"/>
<title>IR in XML documents</title>
<author>J. Le Maitre</author>
<section number="1">
<para>This paper deals with…</para>
…
<section>
…
</article>
…
</anthology>
```

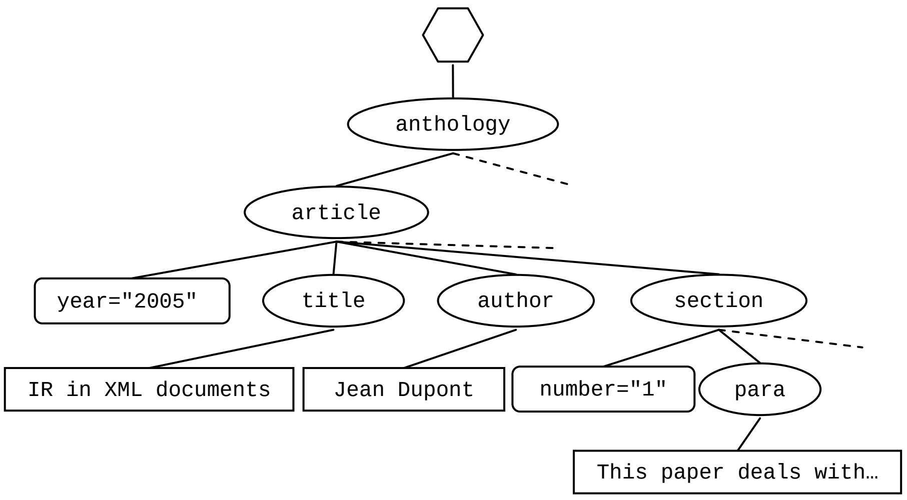

Figure 1. An XML document and its tree

elements specifying author names, publication year, number of pages, etc. It is also the case for attributes. To take in account this observation, we classify the elements of an XML document into two categories: those which relate to the semantic content of this document that we call *content element*, and the others. To be or not to be a content element is a property which is associated to an element type and is held by every instance of this type. Among the content elements, we distinguish the most specific ones (the lowest in the document tree) that we call *terminal content elements* from the others that we call *non terminal content elements*. Moreover, we assume that every element which is an ancestor of a content element is itself a content element. The knowledge of element types whose instances are terminal content elements is enough to determine all content elements of a set of documents. Finally, let us remark that the choice of content elements depends on each set of documents to index and is under the responsibility of the user in charge of indexing these documents. For example, in the document of Figure 1, which can be seen as a sample of an anthology of articles presented at a scientific conference since its creation, we can consider that *title* and *para* elements are terminal content elements what implies that *section*, *article* and *anthology* elements are non terminal content elements.

We propose to associate a vector to each content element of an XML document. Let D be the set of XML documents to be indexed. The vectors associated to the content elements of E are constructed as follows:

1. The text subjacent to each terminal content element e is tokenized, for example by filtering the stop words and by stemming the remaining words. Let T be the set of terms indexing the terminal content elements of D and M be the cardinality of T.
2. A vector $\vec{e} = [w_{1,e}, \ldots, w_{M,e}]$ is associated to each terminal content element e in D. The weight $w_{i,e}$ is calculated by formula (1) where $tf_{i,e}$ is the frequency of the term t_i in the content element e and idf_i is the inverse of the frequency of the term t_i in E calculated by formula (3) where N is the number of terminal content elements in D and N_i is the number of these ones which are indexed by the term t_i. We decided to

not normalize the term frequency because we use the cosine formula to calculate the similarity coefficient between a content element and a query.

3. A vector $\vec{e} = [w_{1,e}, \ldots, w_{M,e}]$ is associated to each non terminal content element *e* of *D* where each weight $w_{i,e}$ is the sum of the weights of the term t_i in the child content elements of *e*. Therefore, the vector associated to a non terminal content element is the sum of the vectors associated to the child content elements of this element. The justification behind this choice is that the vector associated to a content element *e* is the same that the vector associated to the text resulting of the concatenation of the texts contained in the child text nodes of *e* or in the child text nodes of the descendant content elements of *e*.

4. QUERYING VECTOR-INDEXED XML DOCUMENTS WITH XQUERY

In order to use XQuery (Boag, 2005) for information retrieval in a set of XML documents, we propose to extend this language:

- by adding to it a new function, called *vscore*, which returns the similarity degree between a query and a content element;
- by integrating into it Content and Structure (CAS) expressions of the NEXI language (Trotman, 2004). which is a subset of the XPath language defined in the framework of the INEX initiative as the simplest possible language for information retrieval in XML documents.

4.1 Adding the *vscore* function to XQuery

The function *vscore* is the counterpart, for the vector space model, of the *score* function of XQuery FullText (Amer-Yahia, 2005) which reflects the relevance of a Boolean query. The *vscore* function applies to a content element *e* and a textual query *q* and returns the degree of similarity between *e* and *q* computed by the cosine formula (4). For example, the following XQuery query:

```
vscore(doc("anthology.xml")/article[1]/title, "XML and XSLT")
```

returns the degree of similarity between the query `"XML and XSLT"` and the `title` element child of the first `article` element of the document of Figure 1.

The use of the *vscore* function can be illustrated through the following XQuery query addressed to the document of Figure 1 where the value of the `$threshold` variable is the similarity coefficient below which content elements are considered as non relevant:

Find the paragraphs about SGML and XSSL in the articles about XML and XSLT, ordered by decreasing similarity?

```
<answer>
 {
 for $art in doc(anthology)//article
 let $score_art := vscore($art, "XML and XSLT")
 for $para in $art//para
 let $score_para := vscore($para, "SGML and XSSL"),
     $score := min($score_art, $score_para)
 where $score > $threshold
 order by $score
 return <para score="{$score}">$para/text()</para>
 }
</answer>
```

This query can be interpreted as a fuzzy semi-join (`article//para`) of the fuzzy sequence of the paragraphs about SGML and XSSL with the fuzzy sequence of the articles about XML and XSLT. Each paragraph is weighted by the degree of similarity `$score_para` with the query `"SGML and XSSL"` and each article is weighted by the degree of similarity `$score_art` with the query `"XML and XSLT"`. The answer of query *Q* is a fuzzy sequence of paragraphs where each paragraph is weighted by the degree of similarity `$score` whose value is the minimum value of `$score_art` and `$score_para` according to the standard definition of the product of two fuzzy sets (Bosc, 2004).

This example:

- shows that XQuery extended with the *vscore* function can effectively be used as an information retrieval language for XML documents,
- emphasize the need to fuzzy evaluate predicates and operators on content elements.

4.2 Integrating NEXI Content and Structure Queries into XQuery

A NEXI content and structure (CAS) query has the following form (Trotman, 2004):

$$//nt_1[p_1]//\ldots\ nt_{k\text{-}1}[p_{k\text{-}1}]//nt_k[p_k]$$

It applies to a sequence of XML elements and returns elements of type nt_k about p_k which are descendants of elements of type nt_{k-1} about p_{k-1}, …, which are elements of type nt_1 about p_1. Prédicates p_1, …, p_{k-1}, p_k are built from textual or numerical predicates connected by disjunctive or conjunctive connectors. A textual predicate has the following form:

about(*relative_location_path*, *string_litteral*)

and a numerical predicate has the following form:

relative_location_path connector string_litteral

For example, the expression:

```
//article[about(.,"XML") and @year < 2000]//section[about(para, "SGML")]
```

is a NEXI CAS query to retrieve in the document of Figure 1, sections containing a paragraph about SGML contained in an article about XML, published before 2000.

The semantics of a NEXI CAS query has been left deliberately vague in order to let a complete freedom to the participants of the INEX initiative concerning the choice of the formalisms to implement to index or query the content of XML documents. We propose to integrate INEX CAS queries into XQuery as an argument of a new function, called *nexi*, provided with a semantic based on a fuzzy evaluation of predicates and sequence operators. This approach is similar to (Braga, 2002).

Let *e* be an XQuery expression returning a sequence of content elements, let *q* be a NEXI CAS query and let *t* be an XQuery expression returning a real number in [0, 1] and representing a relevance threshold. The function call *nexi*(*e*, *q*, *t*) returns the sequence of the elements of *e* having a similarity degree with *q* which is greater than *t*. Each element of the returned sequence has an attribute *score*, the value of which is the degree of similarity of this element with the query *q*. The returned sequence is ordered by decreasing values of the *score* attribute.

Calling the *nexi* function, the query of section 4.1 can be expressed much more easily, as follows:

```
<answer>
 {nexi(doc("anthology.xml"),
   //article[about(., "XML and XSLT")//para[about(., "SGML and XSSL")],
   $threshold)}
</answer>
```

The *nexi* function has a fuzzy semantics which we define by giving its definition in XQuery. The function call *nexi*(*e*, $//nt_1[p_1]//\ldots\ nt_{k\text{-}1}[p_{k\text{-}1}]//nt_k[p_k]$, *t*) is equivalent to the following XQuery expression:

```
for $e1 in e//nt1
let $score1 := translate[p1]
for $e2 in $e1//nt2
let $score2 := translate[p2], $score := fn:min($score1, $score2)
…
for $ek in $ek-1//ntk
```

```
let $score$k$ := $translate[e_k, p_k]$, $score := fn:min($score, $score$k$)
where $score > $t$
order by $score
return element {fn:name(e$k$)} {(attribute score {$score}), e$k$/@*, e$k$/*}
```

where *translate* is a meta-function which translates an XQuery expression by applying the following rules:

- $translate[e, p] = p$ if p is a numerical predicate,
- $translate[$`about(`e/p`, `q`)`$] =$ `vscore(`e/p`, `t`)`,
- $translate[p_1$ `and` $p_2] =$ `fn:min(`$translate[p_1]$`, `$translate[p_2]$`)` in accordance with the most classical definition of the fuzzy conjunction,
- $translate[p_1$ `or` $p_2] =$ `fn:max(`$translate[p_1]$`, `$translate[p_2]$`)` in accordance with the most classical definition of the fuzzy disjunction,
- $translate[$`(`p`)`$] = translate[p]$.

5. CONCLUSION

In this paper, we have presented a method to index and query content and structure of XML documents according to the vector space model. Indexation is performed in three steps: (i) choosing the names of content elements i.e. those which refer to the semantic content of the documents, (ii) associating a vector to each terminal content element whose components are the weight of each term indexing this element, (iii) propagating these vectors bottom up on the ancestors of the terminal content elements. Then we have shown that documents indexed in this way can be easily queried with XQuery extended by adding to it a *vscore* function which returns the similarity degree between a query vector and a content element vector and by integrating into it the NEXI language that we have equipped with a fuzzy semantic..

This work is still in progress. We are now validating the efficiency of our method by testing it on documents of the INEX test collection. In particular, we want to test various ways to determine the content elements of a set of XML documents and to weight them.

REFERENCES

Amer-Yahia, S. et al. (ed.), 2005. XQuery 1.0 and XPath 2.0 Full-Text. W3C Working Draft, http://www.w3.org/TR/2005/WD-xquery-full-text-20050404/

Baeza-Yates, R. and Ribeiro-Neto, B., 1999. *Modern Information Retrieval*. Addison-Wesley.

Bosc, P. et al, 2004. *Gradualité et imprécision dans les bases de données. Ensembles flous, requêtes flexibles et interrogation de données mal connues*. Ellipses, Paris, France.

Boag, S. et al (ed.), 2005. XQuery 1.0: An XML Query Language. W3C Working Draft, http://www.w3.org/TR/2005/WD-xquery-20050404/

Braga, D. et al., 2002. FXPath: Flexible Querying of XML Documents. *Proceedings of Eurofuse Workshop on Information Systems*. Varenna, Italy.

Fuhr, N. and Großjohann, K., 2001. XIRQL: A Query Language for Information Retrieval from XML Documents. Proceedings of SIGIR 2001, New Orleans, USA, pp. 151-158.

Fuhr, N. et al (eds), 2002. *Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX)*. http://qmir.dcs.qmul.ac.uk/inex/inex02_proceedings/INEX2002_ERCIM_proceedings.pdf.

Fuhr, N. et al (eds), 2003. *INEX 2003 Workshop Proceedings*. http://inex.is.informatik.uniduisburg.de:2003/proceedings.pdf.

Fuhr, N. et al (eds), 2004. *INEX 2004 Workshop Pre-Proceedings*. http://inex.is.informatik.uni-duisburg.de:2004/pdf/INEX2004PreProceedings.pdf.

Trotman, A. and Sigurbjörnsson, B., 2004. Narrowed Extended XPath I (NEXI). *Pre-Proceedings of the INEX 2004 Workshop*. Schloss Dagstuhl, Germany, pp. 219-227