

Searching XML documents using relevance propagation

Karen Sauvagnat, Mohand Boughanem, Claude Chrisment

IRIT- SIG, 118 route de Narbonne
31 062 Toulouse Cedex 4, France
{sauvagna, bougha, chrisment}@irit.fr

Abstract. The issue of information retrieval in XML documents was first investigated by the database community. Recently, the Information Retrieval (IR) community started to investigate the XML search issue. For this purpose, traditional information retrieval models were adapted to process XML documents and rank results by relevance. In this paper, we describe an IR approach to deal with queries composed of content and structure conditions. The XFIRM model we propose is designed to be as flexible as possible to process such queries. It is based on a complete query language, derived from Xpath and on a relevance values propagation method. The value of this proposed method is evaluated thanks to the INEX evaluation initiative. Results show a relative high precision of our system.

1 Introduction

Users looking for precise information do not want to be submerged by noisy subjects, as it can be in long documents. One of the main advantages of the XML format is its capacity to combine structured and un-structured (i.e. text) data. As a consequence, XML documents allow information to be processed at another granularity level than the whole document. The main challenge in XML retrieval is to retrieve the most exhaustive¹ and specific² information unit [12]. Approaches dealing with this challenge can be divided into two main sub-groups [5]. On the one hand, the data-oriented approaches use XML documents to exchange structured data. The database community was the first to propose solutions for the XML retrieval issue, using the data-oriented approaches. In the Xquery language proposed by the W3C [25], SQL functionalities on tables (collection of tuples) are extended to support similar operations on forests (collection of trees), as XML documents can be seen as trees. Unfortunately, most of the proposed approaches typically expect binary answers to very specific queries. However, an extension of XQuery with full-text search features is expected [26]. On the other hand, the document-oriented approaches consider that tags are used to describe the logical structure of documents. The IR community has

¹ An element is exhaustive to a query if it contains all the required information

² An element is specific to a query if all its content concerns the query

adapted traditional IR approaches to address the user information needs in XML collection.

The goal of this paper is to show that the approach we proposed, which belongs to the document-centric view, can also give good results for specific queries (regarding structure) containing content conditions. The following section gives a brief view of related work. Then, in section 3, we present the XFIRM (*XML Flexible Information Retrieval Model*) model and the associated query language. Section 4 presents the INEX initiative for XML retrieval evaluation and evaluates our approach via experiments carried out on the INEX collection.

2 Related work: Information Retrieval Approaches for XML Retrieval

One of the first IR approaches proposed for dealing with XML documents was the “fetch and browse” approach [3, 4], saying that *a system should always retrieve the most specific part of a document answering a query*. This definition assumes that the system first searches whole documents answering the query in an exhaustive way (the *fetch* phase) and then extracts the most specific information units (the *browse* phase). Most of the Information Retrieval Systems (IRS) dealing with XML documents allow information units to be directly searched, without first processing the whole documents. Let us describe some of them.

The *extended boolean model* uses a new non-commutative operator called “contains”, that allows queries to be specified completely in terms of content and structure [11].

Regarding the *vector space model*, the similarity measure is extended in order to evaluate relations between structure and content. In this case, each index term should be encapsulated by one or more elements. The model can be generalized with the aggregation of relevance scores in the documents hierarchy [7]. In [22], the query model is based on tree matching: it allows the expression of queries without perfectly knowing the data structure.

The *probabilistic model* is applied to XML documents in [12, 24, 5]. The XIRQL query language [5] extends the Xpath operators with operators for relevance-oriented search and vague searches on non-textual content. Documents are then sorted by decreasing probability that their content is the one specified by the user.

Language models are also adapted for XML retrieval [1, 15]. Finally, *bayesian networks* are used in [17].

In [9], Fuhr and al. proposed an augmentation method for dealing with XML documents. In this approach, standard term weighting formulas are used to index so called “index nodes” of the document. Index nodes are not necessarily leaf nodes, because this structure is considered to be too fine-grained. However, index nodes are disjoint. In order to allow nesting of nodes, in case of high-level index nodes comprising other index nodes, only the text that is not contained within the other index nodes is indexed. For computing the indexing weights of inner nodes, the weights from the most specific index-nodes are propagated towards the inner nodes. During

propagation, however, the weights are down-weighted by multiplying them with a so-called augmentation factor. In case a term at an inner node receives propagated weights from several leaves, the overall term weight is computed by assuming a probabilistic disjunction of the leaf term weights. This way, more specific elements are preferred during retrieval.

The approach we describe in this paper is also based on an augmentation method. However, in our approach, all leaf nodes are indexed, because we think that even the smallest leaf node can also contain relevant information. Moreover, the way relevance values are propagated in the document tree is function of the distance that separates nodes in the tree. The following section describes our model.

3 The XFIRM model

3.1 Data representation

A structured document sd_i is a tree, composed of simple nodes n_j , leaf nodes ln_j and attributes a_j . Formally, this can be written as follows : $sd_i = (tree_i) = (\{n_j\}, \{ln_j\}, \{a_j\})$. This representation is a simplification of Xpath and Xquery data model [27], where a node can be a *document*, an *element*, *text*, a *namespace*, an *instruction* or a *comment*. In order to easy browse the document tree and to quickly find ancestors-descendants relationships, the XFIRM model uses the following representation of nodes and attributes, based on the Xpath Accelerator approach [10]:

Node : $n_j = (pre, post, parent, attribute)$

Leaf node : $ln_j = (pre, post, parent, \{t_1, t_2, \dots, t_n\})$

Attribute: $a_j = (pre, val)$

A node is defined thanks to its pre-order and post-order value (*pre* and *post*), the pre-order value of its parent node (*parent*), and depending on its type (simple node or leaf node), by a field indicating the presence or absence of attributes (*attribute*) or by the terms it contains ($\{t_1, t_2, \dots, t_n\}$). An attribute is defined by the pre-order value of the node containing it (*pre*) and by its value (*val*). Pre-order and post-order values are assigned to nodes thanks respectively to a prefixed and post-fixed traversal of the document tree, as illustrated in the following figure.

<pre> <article> <fm> <title> Search engines : how to find a need in a haystack</title> <author > J. Dupont </author> <year> 1998 </year> </fm> <body> <sec > <st> Introduction </st> </pre>	<pre> <p> Internet growth...</p> </sec> <sec > <st> Search engines </st> <p> Yahoo! is ...</p> <p> Google is a full-text search engine </p> </sec> </body> </article> </pre>
---	--

Fig. 1: Example of XML document

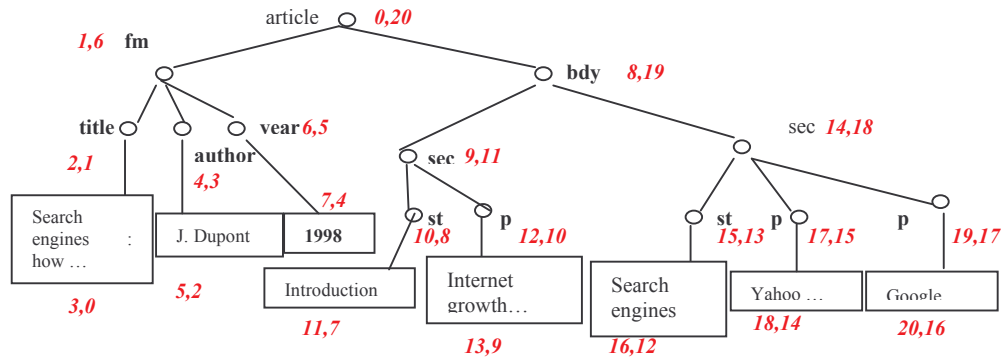


Fig. 2: Tree representation of the XML document in Figure 1. Each node is assigned a pre-order and post-order value.

If we transpose nodes in a two-dimensions space based on the pre and post order coordinates, we can exploit the following properties, given a node n :

- all ancestors of n are to the upper left of n 's position in the plane
- all its descendants are to the lower right,
- all preceding nodes in document order are to the lower left, and
- the upper right partition of the plane comprises all following nodes (regarding document order)

In contrast to other path index structures for XML, Xpath Accelerator efficiently supports path expressions that do not start at the document root. As explained in [19], all data are stored in a relational database. The *Path Index* (PI) allows the reconstruction of the document structure (thanks to the Xpath Accelerator model). The *Term Index* (TI) is a traditional inverted file. The *Element Index* (IE) describes the content of each leaf node, the *Attribute Index* (AI) gives the values of attributes, and the *Dictionary* (DICT) allows the grouping of tags having the same signification.

3.2. The XFIRM Query language

XFIRM is based on a complete query language, allowing the expression of queries with simple keywords terms and/or with structural conditions [20]. In its more complex form, the language allows the expression of hierarchical conditions on document structure and the element to be returned to the user can be specified (thanks to the *te*: (*target element*) operator). For example, the following XFIRM queries:

- (i) // *te*: p [weather forecasting systems]
- (ii) // article[security] // *te*: sec ["facial recognition"]
- (iii) // *te*: article [Petri net] //sec [formal definition] AND sec[algorithm efficiency]
- (iv) // *te*: article [] // sec [search engines]

respectively mean that (i) the user wants a paragraph about *weather forecasting systems*, (ii) a section about *facial recognition* in an article about *security*, (iii) an article about *Petri net* containing a section giving a *formal definition* and another section

talking about *algorithm efficiency*, and (iv) an article containing a section about *search engines*.

When expressing the eventual content conditions, the user can use simple keywords terms (or phrases), eventually preceded by + or - (which means that the term should or should not be in the results). Terms can also be connected with Boolean operators. Regarding the structure, the query syntax allows the user to formulate vague path expressions. For example, he/she can ask for “*article [] // sec []*” (he/she so knows that article nodes have sections nodes as descendants), without necessarily asking for a precise path, i.e. *article/bdy/sec*. Moreover, a tag dictionary is used in query processing. It is useful in case of heterogeneous collections (i.e. XML documents don't necessarily follow the same DTD) or in case of documents containing tags considered as equivalent, like for example, *title* and *sub-title*.

3.3. Query processing

The approach we propose for dealing with queries containing content and structure conditions is based on relevance weights propagation. The query evaluation is carried out as follows:

1. queries are decomposed in elementary sub-queries
2. relevance values are assigned to leaf nodes
3. relevance values are propagated through their document tree
4. original queries are evaluated thanks to elementary sub-queries

Query decomposition

Each XFIRM query can be decomposed in sub-queries SQ_i as follows:

$$Q = // SQ_1 // SQ_2 // \dots // te : SQ_j // \dots // SQ_n \quad (1)$$

Where *te*: indicates which element is the target element. Each sub-query SQ_i can then be re-decomposed in elementary sub-queries $ESQ_{i,j}$, eventually linked with boolean operators and of the form:

$$ESQ_{i,j} = tg [q] \quad (2)$$

Where *tg* is a tag name and $q = \{t_1, \dots, t_n\}$ is a set of keywords, i.e. a content condition.

Evaluating leaf nodes relevance values

The first step in query processing is to evaluate the relevance value of leaf nodes *ln* according to the content conditions (if they exist). Let $q = \{t_1, \dots, t_n\}$ be a content condition. Relevance values are evaluated thanks to a similarity function called $RSV_m(q, ln)$, where *m* is an IR model. XFIRM authorizes the implementation of many IR models. As the purpose of this article is to evaluate the interest of relevance values propagation, we choose to take the vector space model as reference. So:

$$RSV(q, ln) = \sum_{i=1}^n w_i^q * w_i^{ln}, \text{ with } w_i^q = tf_i^q * ief_i \quad \text{And } w_i^{ln} = tf_i^{ln} * ief_i \quad (3)$$

Where: tf_i is the term frequency in the query *q* or in the leaf node *ln*, ief_i is the inverse element frequency of term *i*, i.e. $\log(N/n+1)+1$, *n* is the number of leaf nodes containing *i* and *N* is the total number of leaf nodes.

Elementary sub-queries ESQ_{i,j} processing

The result set $R_{i,j}$ of $ESQ_{i,j}$ is a set of pairs (node, relevance) defined as follows:

$$R_{i,j} = \{ (n, r_n) / n \in \{construct(tg)\} \text{ and } r_n = F_k(RSV_m(q, nf_k), dist(n, nf_k)) \} \quad (4)$$

Where : r_n is the relevance weight of node n ; the $construct(tg)$ function allows the creation of the set of all nodes having tg as tag name ; the $F_k(RSV_m(q, nf_k), dist(n, nf_k))$ function allows the propagation and aggregation of relevance values of leaf nodes nf_k , descendants of node n , in order to form the relevance value of node n . This propagation is function of distance $dist(n, nf_k)$ which separates node n from leaf node nf_k in the document tree (i.e. the number of arcs that are necessary to join n and nf_k).

Subqueries SQ_i processing

Once each $ESQ_{i,j}$ has been processed, subqueries SQ_i are evaluated thanks to the commutative operators \oplus_{AND} et \oplus_{OR} defined below :

Definition 1 : Let $N = \{ (n, r_n) \}$ and $M = \{ (m, r_m) \}$ be two sets of pairs (node, relevance)

$$N \oplus_{AND} M = \{ (l, r_l) / l \text{ is the nearest common ancestor of } m \text{ and } n \text{ or } l=m \text{ (respectively } n) \text{ if } m \text{ (resp. } n) \text{ is ancestor of } n \text{ (resp. } m) , \forall m, n \text{ being in the same document and } r_l = aggreg_{AND}(r_n, r_m, dist(l, n), dist(l, m)) \} \quad (5)$$

$$N \oplus_{OR} M = \{ (l, r_l) / l=n \in N \text{ or } l=m \in M \text{ and } r_l = r_n \text{ or } r_m \} \quad (6)$$

Where $aggreg_{AND}(r_n, r_m, dist(l, n), dist(l, m)) = r_l$ defines the way relevance values r_n and r_m of nodes n and m are aggregated in order to form a new relevance r_l .

Let R_i be the result set of SQ_i . Then :

$$\text{If } SQ_i = ESQ_{i,j}, \text{ then } R_i = R_{i,j} \quad (7)$$

$$\text{If } SQ_i = ESQ_{i,j} \text{ AND } ESQ_{i,k}, \text{ then } R_i = R_{i,j} \oplus_{AND} R_{i,k} \quad (8)$$

$$\text{If } SQ_i = ESQ_{i,j} \text{ OR } ESQ_{i,k}, \text{ then } R_i = R_{i,j} \oplus_{OR} R_{i,k} \quad (9)$$

Whole queries processing

The result set of sub-queries SQ_i are then used to process whole queries. In each query, a target element is specified, as defined above.

$$Q = // SQ_1 // SQ_2 // ... // te : SQ_j // ... // SQ_n$$

Thus, the aim in whole query processing will be to propagate the relevance values of sub-queries SQ_i to nodes belonging to the result set of the sub-query SQ_j which defines the target element. This is obtained thanks to the non-commutative operators ∇ and Δ defined below:

Definition 2 : Let $R_i = \{ (n, r_n) \}$ and $R_{i+1} = \{ (m, r_m) \}$ be two sets of pairs (node, relevance)

$$R_i \nabla R_{i+1} = \{ (n, r_n) / n \in R_i \text{ is ancestor of } m \in R_{i+1} \text{ and } r_n = prop_agg(r_n, r_m, dist(m, n)) \} \quad (10)$$

$$R_i \Delta R_{i+1} = \{ (n, r_n) / n \in R_i \text{ is descendant of } m \in R_{i+1} \text{ and } r_n = prop_agg(r_n, r_m, dist(m, n)) \} \quad (11)$$

Where $prop_agg(r_n, r_m, dist(m,n)) \rightarrow r_n$ allows the aggregation of relevance weights r_m of node m and r_n of node n according to the distance that separates the 2 nodes, in order to obtain the new relevance weight r_n of node n .

The result set R of a query Q is then defined as follows :

$$R = R_j \nabla (R_{j+1} \nabla (R_{j+2} \nabla \dots)) \quad (12)$$

$$R = R_j \Delta (R_{j-1} \Delta (R_{j-2} \Delta \dots))$$

In fact, this is equivalent to propagate relevance values of results set R_{j+1}, \dots, R_n and R_1, \dots, R_n respectively upwards and downwards in the document tree.

4- Experiments and results

4.1. The SCAS task in the INEX initiative

Evaluating the effectiveness of XML retrieval systems requires a test collection (XML documents, task/queries, and relevance judgments) where the relevance assessments are provided according to a relevance criterion that takes into account the imposed structural aspects [6]. The Initiative for the Evaluation of XML Retrieval tends to reach this aim. INEX collection, 21 IEEE Computer Society journals from 1995-2002 consists of 12 135 documents with extensive XML-markup.

Participants to INEX SCAS task (Strict Content and Structure Task) have to perform CAS (Content and Structure) queries, which contain explicit references to the XML structure, and restrict the context of interest and/or the context of certain search concepts. One can found an example of INEX 2003 CAS query below.

```
<inex_topic topic_id="64" query_type="CAS">
<title> //article[about(/,'hollerith')] // sec[about(/, 'DEHOMAG')] </title>
<description> In articles discussing Herman Hollerith find sections that mention DEHOMAG
</description>
<narrative> Relevant sections deal with DEHOMAG in documents that discuss work or life of
Herman Hollerith </narrative>
<keywords> Hollerith, DEHOMAG, Deutsche Hollerith Maschinen Gesellschaft </keywords>
</inex_topic>
```

Fig. 5: Example of CAS query

The INEX metric for evaluation is based on the traditional recall and precision measures. To obtain recall/precision figures, the two dimensions of relevance (exhaustivity and specificity) need to be quantised onto a single relevance value. Quantisation functions for two user standpoints were used: (i) a “strict” quantisation to evaluate whether a given retrieval approach is capable of retrieving highly exhaustive and highly specific document components, (ii) a “generalised” quantisation has been used in order to credit document components according to their degree of relevance.

Some approaches

In INEX 2003, most of the approaches used IR models to answer the INEX tasks, which shows the increased interest of the IR community to XML retrieval.

Some approaches used a fetch and browse strategy [21, 16], which didn't give as good results as expected. The Queensland University of Technology used a filtering method to find the most specific information units [8]. The vector space model was adapted in [14], using 6 different index for terms (article, section, paragraph, abstract,...). Finally language models were used in [2, 13] and [23]. Last cited obtained the best of all performances, using one language model per element.

In the following, we present the results of the experiments we conducted in the INEX collection in order to evaluate several possible implementations of our model.

4.2. Various propagation functions

5 propagation functions have been evaluated.

$$F_k(RSV_m(q, nf_k), dist(n, nf_k)) \text{ (4) is set to:}$$

$$F_k(RSV(q, nf_k), dist(n, nf_k)) = \sum_{k=1..n} \alpha^{dist(n, nf_k)} * RSV(n, nf_k) \quad (13)$$

$agg_{AND}(r_n, r_m, dist(l, n), dist(l, m))$ (5) is either set to :

$$agg_{AND}(r_n, r_m, dist(l, n), dist(l, m)) = \frac{r_n}{dist(l, n)} + \frac{r_m}{dist(m, l)} \quad (14)$$

$$agg_{AND}(r_n, r_m, dist(l, n), dist(l, m)) = \alpha^{dist(l, n)} * r_n + \alpha^{dist(l, m)} * r_m \quad (15)$$

And finally, $prop_agg(dist(m, n), r_n, r_m)$ (10) is either set to:

$$prop_agg(dist(m, n), r_n, r_m) = \frac{r_n + r_m}{dist(m, n)} \quad (16)$$

$$prop_agg(dist(m, n), r_n, r_m) = \alpha^{dist(m, n)} * r_m + r_n \quad (17)$$

Where $\alpha \in]0..1]$ is a parameter used to adjust the importance of the distance between nodes in the different functions and $dist(x, y)$ is the distance which separates node x from node y in the document tree

4.3. Implementation issues

The transformation of INEX CAS queries to XFIRM queries was fairly easy. Table1 gives some correspondences:

Table 1: Transformation of INEX topics into XFIRM queries

INEX topic	XFIRM query
//article [about(., 'clustering + distributed') and about(./sec, 'java')]	// te: article [clustering + distributed] // sec [java]
//article[about(/sec, "e-commerce")] // abs[about(., 'trust authentication')]	//article [] AND sec["e-commerce"] // te: abs [trust authentication]
//article[(./yr='2000' OR ./yr='1999') AND about(., "intelligent transportation system")] // sec [about(., 'automation +vehicle)]	//article ["intelligent transportation system"] // te: sec [automation + vehicle]

During $ESQ_{i,j}$ processing, the most relevant leaf nodes are found, and for each of these leaf nodes, XFIRM looks for ancestors. In order to have a correct response time of the system, the propagation is stopped when 1500 “correct” ancestors are found (i.e. ancestors having a correct tag name).

When a INEX topic contains a condition on the article publication date (as its the case in the last query of Table 1), this condition is not translated in the XFIRM language, because propagation with a very common term (like a year) is too long. To solve this issue, queries are processed by XFIRM without this condition, and results are then filtered on the article publication date.

Finally, the Dictionary index is used to find equivalent tags. For example, according to INEX guidelines, *sec* (section) nodes are equivalent to *ss1*, *ss2* and *ss3*.

4.4. Runs

We evaluated 5 runs, combining the different functions:

Run name	Prop. functions	α	Topic Fields
xfirm.TK.alpha=0.7	(13) (15) (17)	0.7.	Title+Keywords
xfirm.TK.alpha=0.9	(13) (15) (17)	0.9.	Title+Keywords
xfirm.TK.alpha=1	(13) (15) (17)	$\alpha = 1$	Title+Keywords
xfirm.TK.mix	(13) (14) (16)	$\alpha = 0.9$ for (13).	Title+Keywords
xfirm.T.mix:	(13) (14) (16)	$\alpha = 0.9$ for (13)	Title

In addition, these runs were compared to the best run we performed last year in the Inex SCAS task with our fetch and browse method: **Mercure2.pos_cas_ti** [21].

4.5. Analysis of the results

Table 2 shows the average precision (for strict and generalized quantization) obtained by each run over 30 queries. The associated recall-precision curves for strict quantization are plotted in Figure 6.

Table 2: Average precision for our 6 runs

	Average precision (strict quantization)	Average precision (generalized quantization)
Xfirm.TK.alpha=0.7	0,2346	0,2253
Xfirm.TK.alpha=0.9	0,2766	0,2279
Xfirm.TK.alpha=1	0,2783	0,2257
Xfirm.TK.mix	0,2898	0,2300
Xfirm.T.mix	0,2675	0,2276
Mercure2.pos cas ti	0,1620	0,1637

The first point to be noticed is the relatively high precision for all runs. Table 3 shows our runs if they were integrated in the official INEX results for strict quantization. Best results were obtained by the University of Amsterdam, using language

models [23]. Most of our runs would have been ranked between the second and third position, before the Queensland University of Technology [16], who processed queries with a fetch and browse approach.

The propagation method we used increases in a very significant way the results we obtained with our “fetch and browse” method (run *Mercure2.pos_cas_ti*). This is not really surprising, because the XFIRM model is able to process all the content conditions, whereas the run performed with Mercure system only verify that conditions on the target element are respected. Moreover, the processing time for each query is of course lower (because thanks to the index structure, the XFIRM model has not to browse each exhaustive document to find the specific elements). The use of distance between nodes seems to be a useful parameter for the propagation functions. It can be noticed that the *Xfirm.TK.mix* run where distance is considered, obtains best average precision than the *Xfirm.TK.alpha=1* run, where the distance had no importance. However, the three runs evaluated with different values of α (*Xfirm.TK.alpha=0.7*, *Xfirm.TK.alpha=0.9*, *Xfirm.TK.alpha=1*) show that the distance should be considered carefully. Indeed, when relevance values are too down-weighted by the distance, the performances decrease.

Finally, the use of title and keywords fields of INEX topics increases the average precision of the *xfirm.TK.mix* run comparing to the *.xfirm.T.mix* run, even if it decreases the precision for some particular queries.

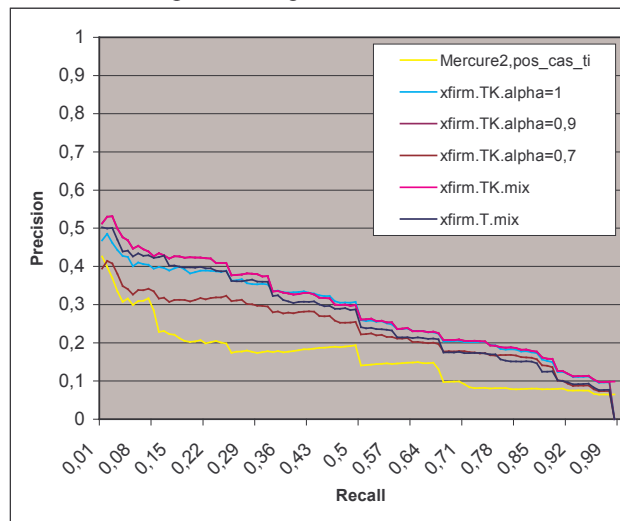


Fig.6 : Average/precision curves for strict quantization

So, the relevance propagation method seems to give good results, using all leaf nodes as start point to the propagation. Our methods have to be explored on other topics/collections to confirm these performances. Moreover, the IR model (i.e. the vector space model) used for relevance value calculation needs more investigations, the formula used for these experiments being not normalized. Further experiments will be necessary, for example with the *bm25* formula [18].

Table 3 : Ranking of official INEX submissions and of our runs for strict quantization. Please note that most of them are too in the “top ten” for generalized quantization.

rank	Avg precision	Organisation	Run ID
1	0.3182	U. of Amsterdam	UamsI03-SCAS-MixedScore
2	0.2987	U. of Amsterdam	UamsI03-SCAS-ElementScore
	0.2898		<i>Xfirm.TK.mix</i>
	0.2783		<i>Xfirm.TK.alpha=1</i>
	0.2766		<i>Xfirm.TK.alpha=0.9</i>
	0.2675		<i>Xfirm.T.mix</i>
3	0.2601	Queensland Univ. of Technology	CASQuery_1
4	0.2476	University of Twente and CWI	LMM-ComponentRetrieval-SCAS
5	0.2458	IBM, Haifa Research lab	SCAS-TK-With-Clustering
6	0.2448	Universität Duisburg-Essen	Scas03-way1-alias
7	0.2437	RMIT University	RMIT_SCAS_1
8	0.2419	RMIT University	RMIT_SCAS_2
9	0.2405	IBM, Haifa Research lab	SCAS-TK-With-No- Clustering
10	0.2352	RMIT University	RMIT_SCAS_3
	0.2346		<i>Xfirm.TK.alpha=0.7</i>
...
24	0.1641	IRIT	<i>Mercure2.pos_cas_ti</i>

5. Conclusion

We have presented here an approach for XML content and structure-oriented search that addresses the search issue from an IR viewpoint. We have described the XFIRM model and a relevance values propagation method that allows the ranking of information units according to their degree of relevance. This propagation method is based on relevance values calculation for each leaf node (thanks to the vector space model) and then on propagation functions using the distance between nodes to aggregate the relevance values. The XFIRM model decomposes each query in elementary sub-queries to process them and then recomposes the original query to respect the eventual hierarchical conditions.

This method achieves good results on the INEX topics. Further experiments should be achieved to evaluate the impact of the IR model used for leaf nodes relevance values calculation and to confirm results on other topics/collections.

References

1. Abolhassani, M., Fuhr, N. : Applying the Divergence From Randomness Approach for Content-Only Search in XML Documents. In: ECIR 04. 2004.
2. Abolhassani M., Fuhr, N., Malik, S. : HyREX at INEX 03. . In Proceedings of INEX 2003 Workshop, 2003.

3. Afrati, Foto N., Koutras, Constantinos D.: A Hypertext Model Supporting Query Mechanisms. Proceedings of the European Conference on Hypertext, 1990.
4. Chiaramella, Y. , Mulhem, P. , Fourel, F. A model for multimedia search information retrieval. Technical report, Basic Research Action FERMI 8134, University of Glasgow, 1996.
5. Fuhr, N., Grossjohann, K. "XIRQL: A query Language for Information Retrieval in XML Documents". In Proc. of the 24th annual ACM SIGIR conference, 2001.
6. Fuhr, N., Malik, S., Lalmas, M : Overview of the Initiative for the Evaluation of XML Retrieval (INEX) 2003. In Proceedings of INEX 2003 Workshop, 2003.
7. M. Fuller, E. Mackie, R. Sacks-Davis, R. Wilkinson : *Structural answers for a large structured document collection*. In Proc. ACM SIGIR, 1993.
8. Geva, S., Murray L-S. : Xpath inverted file for information retrieval. . In Proceedings of INEX 2003 Workshop, 2003.
9. Gövert Norbert, Abolhassani, M., Fuhr, N., Grossjohann, K. : Content-oriented XML retrieval with HyREX. In Proceedings of the first INEX Workshop, 2002.
10. Grust, T, "Accelerating XPath Location Steps". In M. J. Franklin, B. Moon, and A. Ailamaki, editors, *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, USA, 2002*.
11. Hayashi, Y. , Tomita , J., Kikoi, G , "Searching text-rich XML documents with relevance ranking". In *Proc ACM SIGIR 2000 Workshop on XML and IR* . Athens 2000.
12. Lalmas, M., "Dempster-Shafer theory of evidence applied to structured documents: modeling uncertainty". In Proc. ACM-SIGIR, 1997.
13. List, J., Mihazjlovic , V., de Vries A.P., Ramirez, G., Hiemstra, D. : The TIJAH XML-IR system at INEX 03. . In Proceedings of INEX 2003 Workshop, 2003.
14. Mass, Y., Mandelbrod, M. : Retrieving the most relevant XML component. . In Proceedings of INEX 2003 Workshop, 2003.
15. Ogilvie, P., Callan, J. : Using Language Models for Flat Text Queries in XML Retrieval. In Proceedings of INEX 2003 Workshop, 2003.
16. Pehcevski, J., Thom, J., Vercoustre, A-M. : RMIT experiments : XML retrieval using Lucy/eXist. . In Proceedings of INEX 2003 Workshop, 2003.
17. Piwowarski, B., Faure, G-E., Gallinari, P. : Bayesian networks and INEX. In Proceedings in the First Annual Workshop for the Evaluation of XML Retrieval (INEX), 2002.
18. Robertson, S.E., Walker, S., Hancock-Beaulieu, M.M. : Okapi at TREC 3. In Proceedings TREC 3, 1994.
19. Sauvagnat, K. , "XFIRM, un modèle flexible de Recherche d'Information pour le stockage et l'interrogation de documents XML", *CORIA'04*, Toulouse, France, 2004.
20. Sauvagnat, K., Boughanem, M. : Le langage de requête XFIRM pour les documents XML: De la recherche par simples mots-clés à l'utilisation de la structure des documents. Inforsid 2004, Biarritz, France .
21. Sauvagnat, K., Hubert, G., Boughanem, M., Mothe, J. : IRIT at INEX 03. In Proceedings of INEX 2003 Workshop, 2003.
22. Schlieder, T., Meuss, H. , "Querying and ranking XML documents". *Journal of the American Society for Information Science and Technology*, 53(6) : 489-503, 2002.
23. Sigurbjörnsson, B., Kaamps, J., de Rijke, M. : An element-based approach to XML retrieval. . In Proceedings of INEX 2003 Workshop, 2003.
24. J.E. Wolff, H. Flörke, A.B. Cremers : Searching and browsing collections of structural information. In Proc of IEEE advances in digital libraries, Washington, 2000.
25. W3C. XQuery 1.0 : an XML query language. W3C Working Draft, 2003.
26. W3C. Xquery and Xpath Full-Text Use Cases. W3C Working draft, 2003 .
27. W3C. M Fernandez et al. : XQuery 1.0 and XPath 2.0 Data Model. Working Draft, 2003.