Answering Content and Structure-based queries on XML documents using relevance propagation

Karen Sauvagnat, Mohand Boughanem, Claude Chrisment

IRIT/SIG, 118 route de Narbonne,
31062 Toulouse Cedex 4, France
{*sauvagna,bougha,chrisment*}*@irit.fr*

Abstract

As XML documents contain both content and structure information, taking advantage of the document structure in the retrieval process can lead to better identify relevant information units. In this paper, we describe an Information Retrieval (IR) approach dealing 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. This paper aims at evaluating functions used in the propagation process, and particularly the use of distance between nodes as a parameter. The proposed method is evaluated thanks to the INEX evaluation initiative. Results show a relative high precision of our proposal.

Key words: XML, Information retrieval, relevance propagation method, content and structure queries

1 Introduction

Long documents usually contain many subjects. In that case and in an IR context, selecting a whole document as answer unit is not necessary useful for the user. The user may actually require document parts, which are of higher precision and finer granularity. Today, with the availability of XML documents, there is a growing demand of developing techniques that take into account structured and unstructured (i.e. text) data. XML documents allow the processing of information at another granularity level than the whole document. The challenge in IR context is to identify and retrieve relevant parts of the document (or documents passages [13]). In other words, the aim is to retrieve

the most exhaustive [1] and specific [2] information units [12] answering a given query.
Many approaches dealing with this challenge can be found in the literature. They can be divided into two main sub-groups [5]: the data-oriented and the document-oriented approaches.
The database community was the first to investigate the XML retrieval issue, using the data-oriented approaches. In these approaches, XML documents are considered as collections of homogeneous and typed data. Approaches proposed in the literature focus on indexing schemes and query languages. For exemple, the XQuery language proposed by the W3C [26] extends SQL functionalities on tables (collection of tuples) 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. An extension of XQuery with full-text search features is however expected [27].
The document-oriented approaches, proposed by the IR community, consider that tags are used to describe the logical structure of documents. Traditional IR approaches are adapted to address the problem of retrieving document parts that answer the user information needs.
This paper describes a document-centric approach, that can also perform 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 describes the results of the experiments carried out on the INEX collection.

2 Related work: Information Retrieval Approaches for XML Retrieval

XML documents can be viewed as text documents containing tags and relations between tags. Thus, traditional information retrieval models should be extended to take into account the structure and semantic of these documents.
One of the first 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 di-

[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

rectly retrieved, without first processing the whole documents. Let us describe some of them.
The *extended boolean model* uses a new non-commutative operator called "contains". The first term is an Xpath expression and the second a boolean expression. This model allows queries to be specified completely in terms of content and structure [11].
A natural extension of the *vector space model* separates structural information from content information. However, this idea is not really used. In fact, 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 nodes. The model can be generalized with the aggregation of relevance scores in the documents hierarchy [7]. Schlieder and Meuss [23] proposed the ApproXQL model, which integrates the document structure in the vector space model similarity measure. 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 [12,25,5,9]. The XIRQL query language [5] extends the Xpath operators with operators for relevance-oriented search. Other operators allow 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 [1,16] and *bayesian networks* [18] are also adapted for XML retrieval.
Our proposal is more concerned with the approach described in [9]. In this approach, Gövert, Fuhr and al. proposed an augmentation method for dealing with XML documents. 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 containing 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 toward 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.
Our approach 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 (it can be a title or sub-title node for example). The index process can be performed automatically, without any human intervention. Moreover, the way the relevance values are propagated in the document tree is function of the distance that separates nodes in the tree, whereas in [9] the augmentation factor is a simple constant parameter. Finally, our approach is able to process structure conditions in a vague way, by doing

some relevance propagations in the document tree. Documents that do not exactly match the query structure conditions can have a non-zero relevance value, but are lower ranked in the results list. 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_{ij}, leaf nodes ln_{ij} and attributes a_{ij}.
Structured document: $sd_i = (tree_i) = (\{n_{ij}\}, \{ln_{ij}\}, \{a_{ij}\})$
This representation is a simplification of Xpath and Xquery data model [28], in which a node can be a document, an element, text, a namespace, an instruction or a comment.
In order to easily 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 : $n_{ij} = (pre, post, parent, attribute)$
Leaf node : $ln_{ij} = (pre, post, parent, \{(t_1, w_1^{ln_{ij}}), (t_2, w_2^{ln_{ij}}), \ldots, (t_n, w_n^{ln_{ij}})\})$
Attribute: $a_{ij} = (pre, val)$

```
<article>
<fm>
<title> Search engines : how to find a needle in a haystack</title>
<author num= " 1 " > J. Dupont </author>
<year> 1998 </year>
</fm>
<body>
        <sec >
                <st> Introduction </st>
                <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>
```

Table 1
Example of XML document

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 ($\{(t_1, w_1^{ln_{ij}}), (t_2, w_2^{ln_{ij}}), \ldots, (t_n, w_n^{ln_{ij}})\}$) and their associated weight in the node. A simple node can either contains other simple nodes, leaf nodes, or both. This last case is not really a problem, because as each node owns a *pre* and *post* order value independently of its type (simple node or leaf node), the reconstruction of the tree structure can be easily done. 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 pre-fixed and post-fixed traversal of the document tree, as illustrated in figure 1.

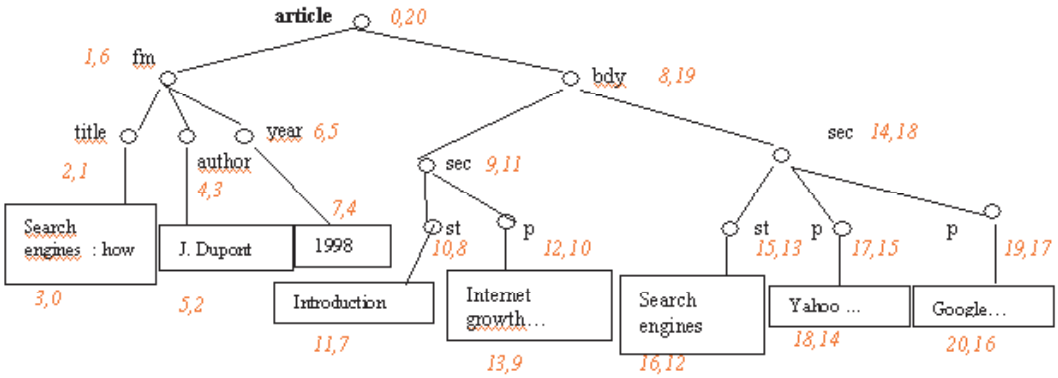

Fig. 1. Tree representation of the XML document in Table 1. Each node is assigned a pre-order and post-order value.

If we represent nodes in a two-dimensions space based on the *pre* and *post* order coordinates (see figure 2), 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)

Xpath Accelerator is well-suited for the navigation in XML documents with Xpath expressions. In contrast to others path index structures for XML, it also efficiently supports path expressions that do not start at the document root. Moreover, this data representation allows the processing of all XML documents, without necessarily knowing their DTD. This implies the ability of the model to process *heterogeneous* collections.
As explained in our previous work [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): for each node, its type, and its pre and post-order values are stored. The *Term Index* (TI) is a traditional

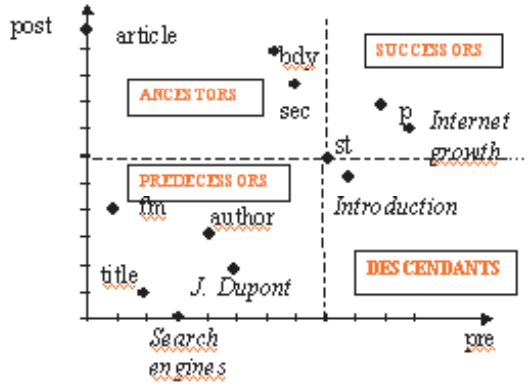

Fig. 2. Representation of the XML document in a two-dimension space based on the *pre* and *post* order coordinates

inverted file, i.e. for each term, the index stores the nodes containing it and its positions in the different nodes. The *Element Index* (IE) describes the content of each leaf node, i.e. the total number of terms and also the number of different terms it contains, and the *Attribute Index* (AI) gives the values of attributes. Finally, the *Dictionary* (DICT) provides for a given tag the tags that are considered as equivalent. It is useful in case of heterogeneous collections (i.e. XML documents with different DTD) or in case of documents containing similar tags, like for example, *title* and *sub-title*. This index is built manually.

3.2 The XFIRM Query language

A query language is associated to the model, allowing the expression of queries with simple keywords terms and/or with structural conditions [21].

When expressing complex structural conditions, users can use boolean and/or hierarchical operators to link what we called *elementary queries*. Elementary queries are atomic structural conditions, composed of an element name and a content condition on this element name. Users can also choose the element they want to be returned, thanks to the *te:* (target element) operator.
Examples of XFIRM queries:
(i) // p [+weather +forecasting systems] : elementary query which means that the user wants paragraphs about weather forecasting systems,
(ii) // sec[privacy law] AND sec[copyright] : boolean query indicating that the user wants an information unit containing a section about privacy law and a section about copyright,
(iii) // article[security] // te: sec ["facial recognition"]: hierarchical query meaning that the user wants sections about facial recognition in articles about security,

(iv) // te: article [Petri net] //sec [formal definition] AND sec [algorithm efficiency]: hierarchical query which indicates that the user wants articles about Petri net containing a section giving a formal definition and another section talking about algorithm efficiency,
(v) // te: article [] // sec [search engines]: hierarchical query which means that the user wants articles containing a section about search engines.
When no target element is specified, the information unit to be returned is decided by the system, as explained in next section. Queries containing structural conditions can be considered as trees, as XML documents.
When expressing the eventual content conditions, the user can enter simple keywords terms, optionally preceded by + or - (which means that the term should or should not be in the results), and connected with boolean operators. Phrases are also processed by the XFIRM system.
Concerning the structure, the query syntax allows the formulation of vague path expressions. For example, the user can ask for *"article[]//sec[]"* (he/she so knows that *article* nodes have *section* nodes as descendants), without necessarily asking for a precise path, i.e. *article[]/bdy[]/sec[]*. Moreover, thanks to the Dictionary index, the user does not need to express in his/her query all the equivalent tags of the tag he/she's looking for. He/she can ask for example for a *section* node, without saying he/she is also interested in *sec* nodes.
User can also express conditions on attribute values, as explained in [21].

The formal syntax of the XFIRM language is described in table 2.

3.3 Query processing

The approach we propose for dealing with queries containing content and structure conditions is based on relevance weights propagation. As seen in the previous section, content and structure queries can be of 4 types : elementary sub-queries, boolean queries, hierarchical queries with or without target element. The processing of these 4 types of queries is linked, as we will see in next section. Hierarchical queries are composed of boolean queries, which are composed of elementary queries. Consequently, the processing of a hierarchical query consists in evaluating elementary queries that compose it and to re-form the query tree thanks to these results.
More precisely, the evaluation iof hierarchical queries in XFIRM is carried out as follows:

(1) Queries are decomposed into boolean queries and elementary queries. Let us consider the following hierarchical query Q:

$$Q = //BQ_1//BQ_2//\ldots//te : BQ_j//\ldots//BQ_n, \quad (1)$$

query ::= <keywordsQuery> | <elementaryQuery> | <booleanQuery>
|<hierarcicalQuery>— <hierarcicalQueryWithTargetElement>

keywordsQuery ::= <reducedExpression> <expressionRemainder>
reducedExpression ::= <terms> <reducedExpressionRemainder>
| "(" <terms> ")" <reducedExpressionRemainder>
reducedExpressionRemainder ::= <reducedExpression> | empty
expressionRemainder ::= <booleanOperator> <P1> | empty
terms ::=<additiveOperator> <keywords>
keywords ::= term<termRemainder> | "" " term<termRemainder>" " "
<termRemainder>
termRemainder ::=empty | <terms>
booleanOperator ::= " OR " | " AND " | " NOT " | empty

elementaryQuery ::=elementName "[" <condition> "]"
condition ::= "@" attributeName "=" term | P1 | vide

booleanQuery ::= <elementaryQuery><elementaryQueryRemainder>
elementaryQueryRemainder ::= <booleanOperator> <subQuery> | empty

hierarchicalQuery ::= "//" <subQuery><hierarchicalQueryRemainder>
hierarchicalQueryRemainder ::=<hierarchicalQuery> | empty

hierarcicalQueryWithTargetElement ::= <hierarchicalQueryRemainder>
<targetElement><hierarchicalQueryRemainder >
targetElement ::= "// te: " <subQuery>

Key : empty : terminal expression for the empty set
term : terminal expression for a keyword
elementName : terminal expression representing a tag name
attributeName : terminal expression representing an attribute name
te : terminal expression indicating the presence of a tagart element

Table 2
Grammar of the XFIRM query language

where BQ_i are boolean queries and te : indicates the target elements. Each boolean query BQ_i can then be re-decomposed into elementary queries $EQ_{i,j}$, eventually linked with boolean operators and of the form:

$$EQ_{i,j} = \begin{cases} tg\_n[q] \\ tg\_n[] \\ tg\_n[a\_n = v] \end{cases} \tag{2}$$

Where tg_n is a tag name and $q = \{(t_1, w_1^q), \ldots, (t_n, w_1^n)\}$ is a set of keywords with their associated weight in the query, i.e. a content condition, a_n is the attribute name of attribute a, and v is the desired value of a. An example of query decomposition can be found in paragraph 3.4.

(2) Relevance values $RSV(q, ln)$ are then evaluated between leaf nodes ln and the content conditions q of elementary queries.
(3) Relevance values are propagated through the document tree to answer to the structure conditions of elementary queries. The relevance value of a node n having tg_n as tag name is evaluated thanks to the F_k *(RSV_m(q, ln_k), dist(n,ln_k))* function, which takes into account the distance that separates node n from leaf nodes ln_k;
(4) Boolean queries are processed using the results sets of elementary queries and the $\oplus_{OR}$ and $\oplus_{AND}$ operators.
(5) Original queries are evaluated thanks to upwards and downwards propagation of the relevance scores of nodes relevant to boolean queries to nodes belonging to the target elements set. These propagations are done using the $prop\_agg(r_n, r_m, dist(m, n))$ function, which takes into account the relevances values of nodes and the distance that separates them in the document tree.

3.3.1 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 of each elementary query (if they exist). Let $q = \{(t_1, w_1^q), \ldots, (t_n, w_1^n)\}$ be a content condition. Relevance values are computed thanks to a similarity function called $RSV_m(q, ln)$ (Retrieval Status Value), where m is an IR model. The XFIRM system authorizes the implementation of many IR models, which will be used to assign a relevance value to leaf nodes. As shown in [22], a simple adaptation of the $tf - idf$ measure to XML documents seems to perform better in case of content and structure queries. So:

$$RSV_m(q, ln) = \sum_{i=1}^{n} w_i^q * w_i^{ln} \tag{3}$$

with $w_i^q = tf_i^q * ief_i$ and $w_i^{ln} = tf_i^{ln} * ief_i$
And 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*, where n is the number of leaf nodes containing i and N is the total number of leaf nodes. If q is a phrase, n is the number of leaf nodes containing the phrase. This way, nodes containing phrases are preferred to nodes containing simple terms (if the query is composed of a phrase and of a list of keywords).

3.3.2 Elementary queries $EQ_{i,j}$ processing

The result set $R_{i,j}$ of elementary query $EQ_{i,j}$ is a set of pairs *(node, relevance)* defined as follows, depending on the form of $EQ_{i,j}$:
(1) If $EQ_{i,j} = tg\_n[q]$,
the relevance values assigned to leaf nodes are propagated upwards in the document tree until nodes having the asked tag name are found. The result set of an elementary query $tg\_n[q]$ is so composed of nodes having tg_n as tag name and their associated relevance values, which are obtained thanks to the propagation.
Formally,

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

Where :
- r_n is the relevance weight of node n
- the $construct(tg\_n)$ function allows the creation of the set of all nodes having tg_n 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).
We evaluated several propagation functions, that are described in paragraph 4.2.

(2) If $EQ_{i,j} = tg\_n[]$,

$$R_{i,j} = \{(n, 0)/n \in construct(tg\_n)\} \quad (5)$$

i.e. the set of nodes having tg_n as tag name.

(3) If $EQ_{i,j} = tg\_n[a\_n = v]$,

$$R_{i,j} = \{(n, 1) \,/\, n \in construct(tg\_n), a \in construct(a\_n)\ isAttribute\ of\ n \\ and\ value(a) = v)\} \quad (6)$$

A relevance value of 1 (which is the maximal score that a node answering directly to content conditions can have) is associated to nodes that verify conditions on attributes values. We consider these conditions as conditions on

data and not on text, and we process attribute values by doing *exact* match (in a database sense).

3.3.3 Boolean queries BQ_i processing

Once each $EQ_{i,j}$ has been processed, boolean queries BQ_i are then evaluated as explained below. Let R_i be the result set of BQ_i.
- If boolean query BQ_i is composed of one elementary query $EQ_{i,j}$ then the result set of BQ_i is the same than the one of $EQ_{i,j}$

$$If\ BQ_i = EQ_{i,j},\ then\ R_i = R_{i,j} \tag{7}$$

- If boolean query BQ_i is composed of elementary queries $EQ_{i,j}$ linked by the Boolean operator AND, the result set of BQ_i is composed of nodes being the nearest common ancestors of nodes belonging to the result sets of elementary sub-queries $EQ_{i,j}$. The associated relevance values are obtained thanks to propagation functions. Formally,

$$If\ BQ_i = EQ_{i,j}\ AND\ EQ_{i,k},\ then\ R_i = R_{i,j} \oplus_{AND} R_{i,k} \tag{8}$$

with $\oplus_{AND}$ defined as follows:

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

$$\begin{aligned} N \oplus_{AND} M = \{&(l, r_l)/\ l\ is\ the\ nearest\ common\ ancestor\ of\ m\ and\ n,\\ &or\ l = m\ (respectively\ n)\ if\ m\ (resp\ .n)\ is\ ancestor\ of\ n\\ &(resp.\ m),\ \forall m, n\ being\ in\ the\ same\ document\\ &and\ r_l = aggreg_{AND}(r_n, r_m, , dist(l, n), dist(l, m))\} \end{aligned} \tag{9}$$

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.

An example of such boolean query processing can be found in paragraph 3.4. We evaluated several functions for $aggreg_{AND}(r_n, r_m, dist(l, n), dist(l, m))$, which are described in paragraph 4.2.
- If boolean query BQ_i is composed of elementary queries $EQ_{i,j}$ linked by the Boolean operator OR, the result set of BQ_i is an union of the result sets of elementary queries $EQ_{i,j}$.

$$If\ SQ_i = ESQ_{i,j}\ OR\ ESQ_{i,k},\ then\ R_i = R_{i,j} \oplus_{OR} R_{i,k} \tag{10}$$

with $\oplus_{OR}$ defined as follows:

Definition 2 *Let $N = \{(n, r_n)\}$ and $M = \{(m, r_m)\}$ be two sets of pairs (node, relevance).*

$$N \oplus_{OR} M = \{(l, r_l) \,/\, l = n \in N \text{ and } r_l = r_n \\ \text{or } l = m \in M \text{ and } r_l = r_m \\ \text{or } l = n = m \text{ and } r_l = aggreg_{OR}(r_n, r_m)\} \quad (11)$$

3.3.4 Hierarchical query processing

The processing of hierarchical queries $//BQ_1//BQ_2// \ldots //BQ_n$ consists in evaluating hierarchical conditions. For this purpose, results sets of boolean queries are used, and are combined thanks to the Δ operator defined below:

Definition 3 *Let $R_i = \{(n, r_n)\}$ and $R_{i+1} = \{(m, r_m)\}$ be two sets of pairs (node, relevance)*

$$R_i \Delta R_{i+1} = \{(n, r'_n)\} \quad (12)$$

$$with\ r'_n = \begin{cases} r_n + prop\_agg(r_n, r_m, dist(m,n)) \\ \qquad if\ n \in R_i\ is\ ancestor\ of\ m \in R_{i+1} \\ r_n\ else \end{cases} \quad (13)$$

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 . Relevance weights of nodes belonging to R_i are not necessarily augmented, but nodes are still considered as relevant. This way, document having a structure that do not match exactly the query structure can have a non-zero relevance value.

The result set R of a hierarchical query is thus defined as follows:

$$R = R_1 \Delta (R_2 \Delta (R_3 \Delta \ldots)) \quad (14)$$

which is equivalent to propagate the scores of the results nodes of boolean queries BQ_2 to BQ_n upwards in the document tree to results nodes of BQ_1, which constitute the result set sent back to the user.
If the result set of BQ_i is empty (no relevant node are found), the Δ operator is used between R_{i-1} and R_{i+1}. This way, we prevent having an empty result set for hierarchical queries.

3.3.5 Hierarchical query with target element processing

Whereas in case of simple hierarchical queries, nodes scores are propagated upwards in the document tree, in case of hierarchical query with target element, scores can be propagate backwards, due to the presence of the target element operator, which indicates the type of nodes to send back to users. We need to define the non-commutative operator ∇ as follows:

Definition 4 *Let* $R_i = \{(n, r_n)\}$ *and* $R_{i+1} = \{(m, r_m)\}$ *be two sets of pairs (node, relevance)*

$$R_i \nabla R_{i+1} = \{(m, r'_m)\} \tag{15}$$

$$with\ r'_m = \begin{cases} r_m + prop\_agg(r_n, r_m, dist(m,n)) \\ \qquad if\ m \in R_{i+1}\ is\ descendant\ of\ n \in R_i \\ r_n\ else \end{cases} \tag{16}$$

Thus, the ∇ operator is used to propagate scores of results nodes of boolean queries BQ_1 to BQ_{i-1} backwards in the document tree to results nodes of BQ_i, which constitute the target elements asked by the user.

The result set R of a hierarchical query with target element Q= // BQ_1 // BQ_2 //. . . //te : BQ_j //. . . //BQ_n is thus defined in 3 steps:

(1) Upwards propagation of scores of nodes belonging to results sets $R_{i+1}...R_n$ to nodes belonging to the set of target elements R_i:

$$SR_1 = R_i \Delta (R_{i+1} \Delta (R_{i+2} \Delta \ldots)) \tag{17}$$

(2) Backwards propagation of scores of nodes belonging to results sets $R_1...R_{i-1}$ to nodes belonging to the set of target elements R_i:

$$SR_2 = (((R1 \bigtriangledown R2) \bigtriangledown R3) \bigtriangledown \ldots) \bigtriangledown R_i \tag{18}$$

As for upwards propagation, if R_i is empty (no relevant node for BQ_i), the $\bigtriangledown$ operator is used between R_{i-1} and R_{i+1}, in order to prevent having an empty result set SR_2.

(3) Union of the 2 results sets created previously:

$$R = SR_1\ U\ SR_2 \tag{19}$$

The model we presented for processing content and structure queries allows the evaluation of the similarity between the query tree and the document tree, thanks to many propagation in the document tree. The way the different propagation functions are ajusted allow the processing of queries in a more or less strict way.
For example, let us consider documents of figure 3 and the query *//a[content1] // i[content2] // te: c[content3].*

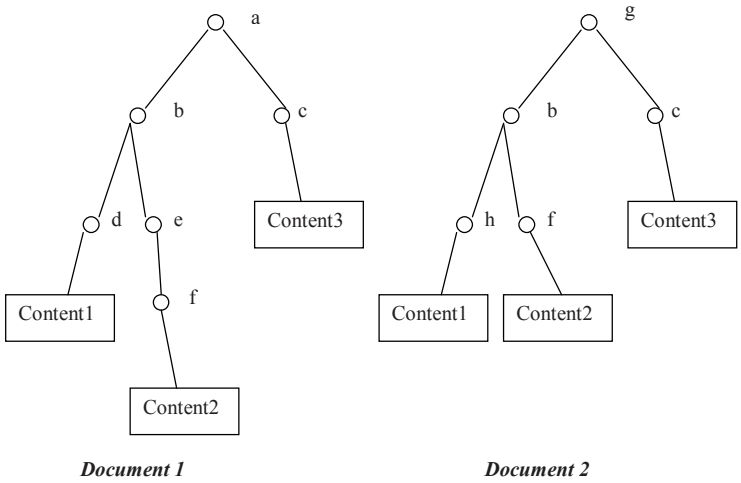

Fig. 3. Example of vague processing of document structure

Nodes $/a/c$ of document 1 and $/g/c$ of document 2 will have a non-zero relevance value, even if neither of the two respects all the structure conditions of the query. Node $/a/c$ of document 1 will be however better ranked in the result list.
If we now consider the query $//b[content1]//te : f[content2]$; node $g/b/f$ of document 2 will be better ranked than node $/a/b/e/f$ of document 1, due to the distance that separates b and f in document trees.
In the same way, a node having a path that does not respect the order of the structure conditions of a hierarchical query would have a non-zero relevance value, but will be ranked lower in the result list than a node having a XPath respecting this order. For example, node $/a/b/d$ of document 1 will have a non-zero relevance value for query *//a[]//d[]//b[content2].*

3.4 Example

Let us take the XFIRM query:
// te: article [search engines] // sec [Internet growth] AND sec [Google],
which means that the user is looking for articles about *search engines* containing a section about *Google* and a section about *Internet growth.*

This query can be decomposed in boolean queries and elementary queries as follows:

$$
\begin{aligned}
BQ_1 &= article[search\ engines] \\
&\Rightarrow \quad EQ_{1,1} = article[search\ engines] \\
BQ_2 &= //sec[Internet\ growth]\ AND\ sec[Google] \\
&\Rightarrow \quad EQ_{2,1} = sec[Internet\ growth] \\
&\qquad\ EQ_{2,2} = sec[Google]
\end{aligned}
$$

Each elementary query is processed independently. Leaf nodes compute a relevance value and this value is propagated along the document tree until a node having the asked tag name is found. This process is illustrated in Figure 4.

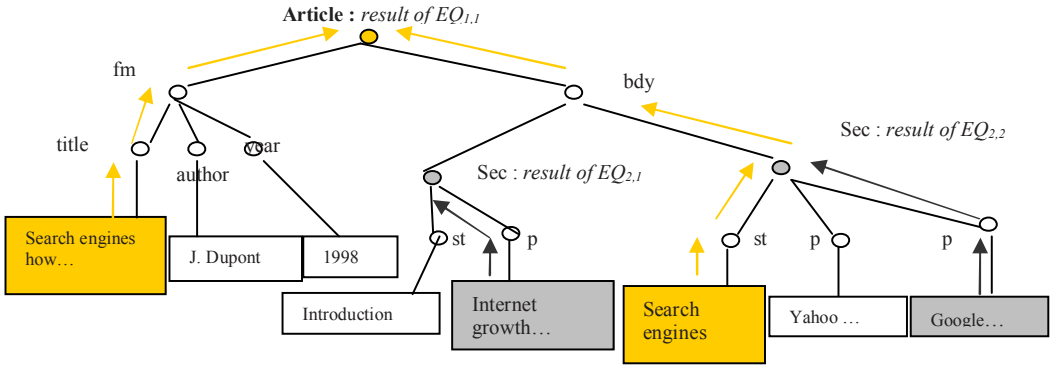

Fig. 4. Processing of elementary queries $EQ_{1,1}$, $EQ_{2,1}$ and $EQ_{2,2}$

As BQ_1 is composed of one elementary query $EQ_{1,1}$, the result of BQ_1 is the result of $EQ_{1,1}$, i.e. the *article* node. Boolean query BQ_2 is processed thanks to the $\oplus_{AND}$ operator : its result is the nearest common ancestor of the result node of $EQ_{2,1}$ and of the result nodes of $EQ_{2,2}$, i.e. the *bdy* node. The initial query is finally evaluated. For this purpose, relevance value of nodes in BQ_2 result set are propagated in the document tree until a node belonging to the BQ_1 result set (i.e. the set of target elements) is found. In our example, the relevance value of the *bdy* node is propagated upwards to the *article* node. Figure 5 illustrates the final propagation.

4 Experiments and results

The aim of the experiments presented here was to evaluate the functions used during the propagation process, and particularly the use of distance between nodes as parameter. For this purpose, we conducted some runs for the SCAS task of the INEX evaluation initiative.

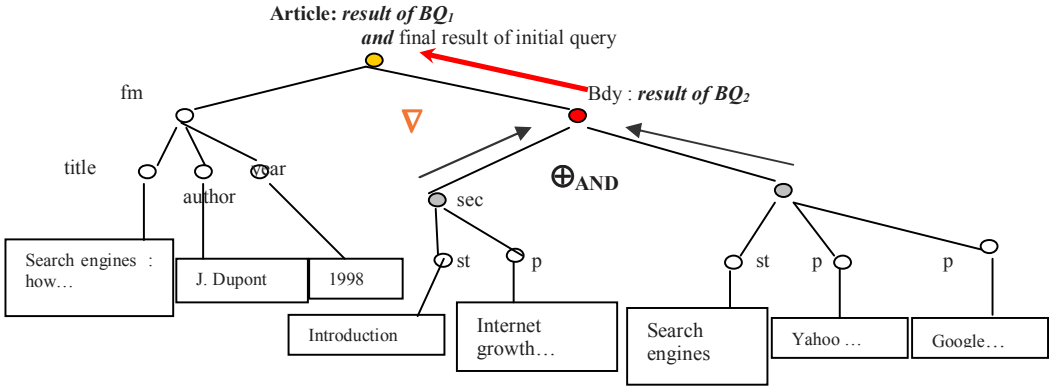

Fig. 5. Initial query reconstruction

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. The 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 (Deutsche Hollerith Maschinen Gesellschaft) in documents that discuss work or life of Herman Hollerith </narrative>
<keywords> Hollerith, DEHOMAG, Deutsche Hollerith Maschinen Gesellschaft </keywords>
</inex_topic>

Table 3
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 able 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 2002, the first INEX workshop classified the different approaches in three categories: extending well known full-text information retrieval models to handle XML retrieval; extending database management systems to deal with XML data; and XML- specific, which use native XML databases that usually incorporate existing XML standards (such as XPath, XSL or XQuery). Last year, most of the approaches used IR models to answer the INEX tasks. This shows the increased interest of the IR community to XML retrieval. Some approaches used a fetch and browse strategy [20,17], 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 [15], using 6 different index for terms (article index, section index, paragraph index, abtract index,...). Finally language models were used in [2,14,24]. Best performances were obtained in [24], 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

For each of the propagation functions, two or three functions are evaluated.
- $F_k(RSV_m(q, nf_k), dist(n, nf_k))$ (4) is either set to:

$$\hookrightarrow \; F_k(RSV_m(q, nf_k), dist(n, nf_k)) = \sum_k \beta * RSV(q, nf_k) \tag{20}$$

$$\hookrightarrow \; F_k(RSV_m(q, nf_k), dist(n, nf_k)) = \sum_k \alpha^{dist(n,nf_k)} * RSV(q, nf_k) \tag{21}$$

- $aggreg_{AND}(r_n, r_m, , dist(l, n), dist(l, m))$ (9) is either set to :

$$\hookrightarrow \; aggreg_{AND}(r_n, r_m, dist(l, n) \;,\; dist(l, m)) = \beta * (r_n + r_m) \tag{22}$$

$$\hookrightarrow \; aggreg_{AND}(r_n, r_m, dist(l, n) \;,\; dist(l, m)) = \frac{r_n}{dist(l, n)} + \frac{r_m}{dist(l, m)} \tag{23}$$

$$\hookrightarrow \; aggreg_{AND}(r_n, r_m, dist(l, n) \;,\; dist(l, m)) = \alpha^{dist(l,n)} * r_n + \alpha^{dist(l,m)} * r_m \tag{24}$$

And finally, $prop\_agg(dist(m, n), r_n, r_m)$ (13) is either set to:

$$\hookrightarrow \; prop\_agg(dist(m,n), r_n, r_m) = \beta * r_m + r_n \tag{25}$$

$$\hookrightarrow \; prop\_agg(dist(m,n), r_n, r_m) = \frac{r_n + r_m}{dist(n,m)} \tag{26}$$

$$\hookrightarrow \; prop\_agg(dist(m,n), r_n, r_m) = \alpha^{dist(m,n)} * r_m + r_n \tag{27}$$

Where β and $\alpha \in]0 \ldots 1]$ are constant parameters, and *dist(x,y)* is the distance which separates node x from node y in the document tree (i.e. the number of arcs that are necessary to join x and y). Functions 20, 22, 25 use a simple β parameter for down-weighting relevance weights, like the experiments conducted in Gövert et al. paper [9]. The importance of the distance parameter is evaluated in functions 21, 24, 27, thanks to the α constant.
At last, for $aggreg_{OR}$, we use a simple sum function:

$$aggreg_{OR}(r_n, r_m) = r_n + r_m \tag{28}$$

4.3 Implementation issues

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

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]

Table 4
Transformation of INEX topics into XFIRM queries

When a INEX topic contains a condition on the article publication date (as its the case in the last query of Table 4, 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 36 runs, combining the different propagation functions.

- A first set of runs uses functions (20) (22) (25) : these runs are performed in order to evaluate the use of a simple constant parameter β for down-weighting relevance weights. We evaluated several values for β, from 0.5 to 1. In the following, these runs are labeled with "$\beta = value$".
- A second set of runs uses functions (21) (24) (27) : they are performed in order to evaluate the impact of the distance parameter (modeled with α) in the propagation functions. We evaluated several values for α, from 0.5 to 1 [3] . These runs are labeled with "$\alpha = value$"
- A third set of runs uses functions (21) (23) (26): Runs also use the distance between nodes as a parameter in the propagation functions, but in a different way. Values of α go from 0.5 to 1 [4] . These runs are labeled with "$mix.\alpha = value$".

For all cases described behind, we take into account either the *title* and *keywords* fields of topics (runs labeled "TK") or the *title* only field (runs labeld with "T") or .

4.5 Analysis of the results

For all 36 runs, we processed 30 queries, expressed with the XFIRM language. The 36 runs resulted in 72 average precisions (for strict and generalized quantization) that are plotted in figures 6, 7.

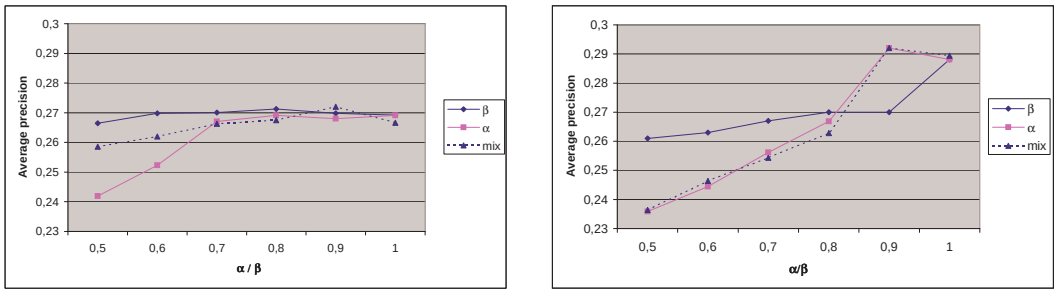

Fig. 6. Evolution of average precision against α and β for strict quantization, use of title field only of topics (left) or title and keywords fields (right)

Associated recall-precision curves of the most significative results for strict

[3] We also experimented with values from 0.1 to 0.4, but average precisions were lower

[4] Experiments with values from 0.1 to 0.4 did not give good results

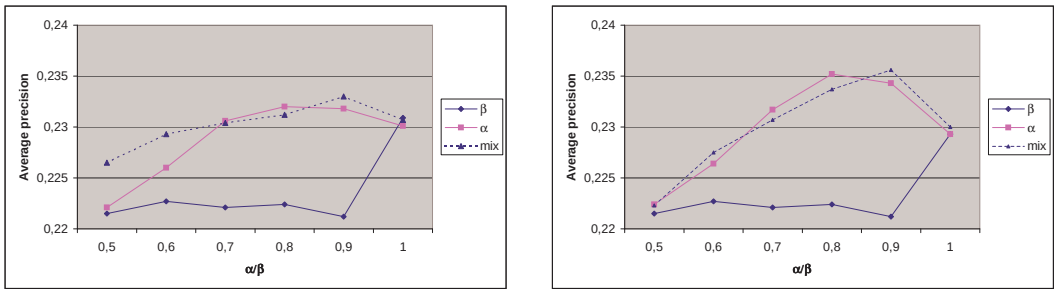

Fig. 7. Evolution of average precision against α and β for generalized quantization, use of title field only of topics (left) or title and keywords fields (right)

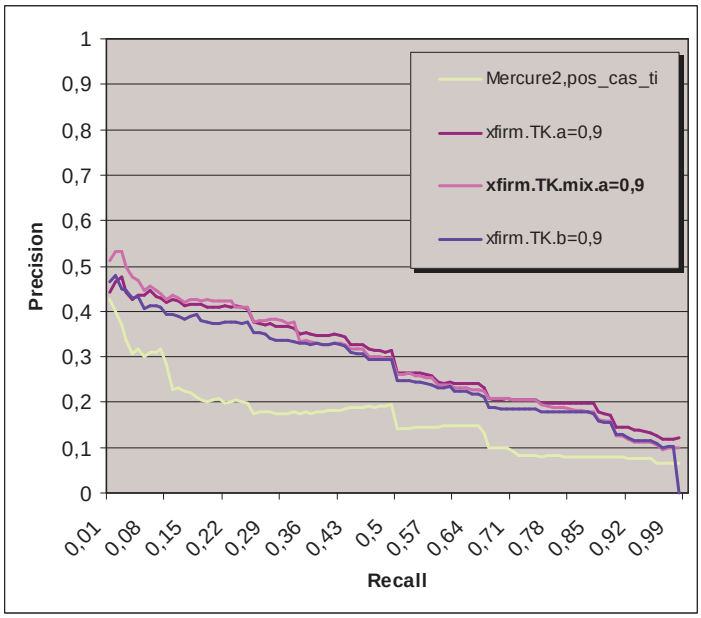

Fig. 8. Average/precision curves for strict quantization

quantization are plotted in figure 8. They are compared to the best run we performed last year in the Inex SCAS task with our fetch and browse method: *Mercure2.pos_cas_ti* [20] .

The first point to be noticed is the relatively high precision for all runs comparing to the official INEX results. Table 5 shows our best run if it was integrated in the official results for strict quantization. Best results were obtained by the University of Amsterdam, using language models [24]. Most of our runs would have been ranked between the second and third position, before the Queensland University of Technology [17], 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 checked that conditions on the target element were respected.

In a general manner, the use of title and keywords fields of INEX topics

rank	Avg precision	Organisation	Run ID
1	0.3182	U. of Amsterdam	UamsI03-SCAS-MixedScore
2	0.2987	U. of Amsterdam	UamsI03-SCAS-ElementScore
	0.2920		***Xfirm.TK.mix.α=0.9***
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	Universitt 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
...	...	...	...
24	***0.1641***	***IRIT***	***Mercure2.pos_cas_ti***

Table 5
Ranking of official INEX submissions and of our best run for strict quantization. Please note that most of our runs are also in the "top ten" for both strict and generalized quantization.

increases the average precision of almost all runs, even if it decreases the precision for some particular queries.
Concerning the impact of the distance between nodes in the propagation process, it can be noticed that the more weights are down-weighted by the β constant, the more the average precision decreases. Indeed, the best average precisions for "β runs" are obtained for β=1, which is equivalent to simply sum the leaf nodes weights (figure 6 and 7).
However, down-weighting leaf nodes weights during propagation seems to be useful when using the distance parameter: runs with $\alpha = 1$ (and with $\beta = 1$) obtain lower precision than runs with $\alpha = 0.9$ where distance between nodes is considered. However, runs evaluated with different values of α show that the distance parameter should be considered carefully. Indeed, when relevance values are too down-weighted by the distance, performances decrease.
The sets of functions (21) (23) (26) and (21) (24) (27) taking into account the distance between nodes during the propagation obtain similar results, with a slight preference for functions (21) (23) (26).
Finally, it can be noticed that all this runs were re-processed The relevance propagation method seems to give good results, using all leaf nodes as start

point to the propagation. The distance between nodes seems to be a useful parameter during the propagation, but should be considered carefully. Our methods have to be explored on other topics/ collections to confirm these performances and we are currently working on the INEX Heterogeneous Collection Track to achieve this goal. Moreover, the propagation functions used are now to be evaluated with content only queries. As in this case no structural constraint is expressed, the way the propagation functions are used is a crucial issue.

5 Conclusion

We have presented in this paper an approach for XML content and structure-oriented search that addresses the search issue from 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 computed for each leaf node and then on propagation functions using the distance between nodes to aggregate the relevance values. The XFIRM model decomposes each query into 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 for the ad-hoc track. Further experiments should be achieved to confirm results on other topics/ collections and to adapt this method to content only queries.

References

[1] M. Abolhassani, N. Fuhr, Applying the Divergence From Randomness Approach for Content-Only Search in XML Documents. In: ECIR 04. April 2004, pp. 409-419.

[2] M. Abolhassani , N. Fuhr, S. Malik, HyREX at INEX 03. In Proceedings of INEX 2003 Workshop, December 2003.

[3] Afrati, N. Foto , Koutras, D. Constantinos : A Hypertext Model Supporting Query Mechanisms. Proceedings of the European Conference on Hypertext, INRIA, France, November 1990. pp. 52-66

[4] Y. Chiaramella, P. Mulhem, F. Fourel, A model for multimedia search information retrieval. Technical report, Basic Research Action FERMI 8134, University of Glasgow, April 1996.

[5] N. Fuhr, K. Grossjohann, XIRQL: A query Language for Information Retrieval in XML Documents. In Proc. of the 24th annual ACM SIGIR conference on research and development in Information Retrieval, New Orlans, USA, p. 172-180. ACM Press, 2001.

[6] N. Fuhr, S. Malik, M. Lalmas, Overview of the Initiative for the Evaluation of XML Retrieval (INEX) 2003. In Proceedings of INEX 2003 Workshop, December 2003.

[7] M. Fuller, E. Mackie, R. Sacks-Davis, R. Wilkinson : Structural answers for a large structured document collection. In Proc. ACM SIGIR, pp. 204-213. Pittsburgh, 1993.

[8] S. Geva, L-S. Murray, Xpath inverted file for information retrieval. In Proceedings of INEX 2003 Workshop, December 2003.

[9] N. Gövert, M. Abolhassani, N. Fuhr, K. Grossjohann, Content-oriented XML retrieval with HyREX. In Proceedings of the first INEX Workshop, December 2002.

[10] T. Grust, 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, Madison, Wisconsin, USA, ACM Press, 2002.

[11] Y. Hayashi, J. Tomita , G. Kikoi, Searching text-rich XML documents with relevance ranking. In Proc ACM SIGIR 2000 Workshop on XML and IR (pp. 27-35). Athens 2000.

[12] M. Lalmas, Dempster-Shafer Theory of evidence applied to structured documents: modeling uncertainty. In Proc. ACM-SIGIR, pp. 110-118, Philadelphia, 1997.

[13] P.-Y. Lambollez , J.-P. Queille , J-F. Voidrot ,C. Chrisment , EXREP : un outil générique de réécriture pour l´extraction d´informations textuelles , Revue ISI, Vol. 3(4), 1995, pp 471-487.

[14] J. List, V. Mihazjlovic , A.P. de Vries, G. Ramirez, D. Hiemstra, The TIJAH XML-IR system at INEX 03. . In Proceedings of INEX 2003 Workshop, December 2003.

[15] Y. Mass, M. Mandelbrod, Retrieving the most relevant XML component. . In Proceed-ings of INEX 2003 Workshop, December 2003.

[16] P. Ogilvie, J. Callan, Using Language Models for Flat Text Queries in XML Retrieval. In Proceedings of INEX 2003 Workshop, pp. 12-18, December 2003.

[17] J. Pehcevski, J. Thom, A-M. Vercoustre, RMIT experiments : XML retrieval using Lucy/eXist. . In Proceedings of INEX 2003 Workshop, December 2003.

[18] B. Piwowarski, G-E. Faure, P. Gallinari, Bayesian networks and INEX. In Proceedings in the First Annual Workshop for the Evaluation of XML Retrieval (INEX), Dec. 2002.

[19] K. Sauvagnat , XFIRM, un modèle flexible de Recherche d´Information pour le stockage et l`interrogation de documents XML, CORIA'04, Toulouse, France.

[20] K. Sauvagnat, G. Hubert, M. Boughanem, J. Mothe, IRIT at INEX 03. In Proceedings of INEX 2003 Workshop, December 2003.

[21] K. Sauvagnat, M. Boughanem, 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.

[22] K. Sauvagnat, M. Boughanem, The impact of leaf nodes relevance values evaluation in a propagation method for XML retrieval. In: 3rd XML and Information Retrieval Workshop, SIGIR 2004, Sheffield, England, July 2004. Ricardo Baeza-Tates, Yoelle Marek, Thomas Roelleke, Arjen P. de Vries, p. 19-22.

[23] T. Schlieder, H. Meuss, Querying and ranking XML documents. Journal of the American Society for Information Science and Technology, 53(6) : 489-503, 2002.

[24] B. Sigurbjörnsson, J. Kaamps, M. de Rijke , An element-based approach to XML retrieval. In Proceedings of INEX 2003 Workshop, December 2003.

[25] J.E. Wolff, H. Flrke, A.B. Cremers , Searching and browsing collections of structural information. In Proc of IEEE advances in digital libraries, Washington, 2000.

[26] World Wide Web Consortium (W3C), XQuery 1.0 : an XML query language. W3C Working Draft, November 2003. http://www.w3.org/TR/xquery/

[27] World Wide Web Consortium (W3C), Xquery and Xpath Full-Text Use Cases. W3C Working draft, February 2003. http://www.w3.org/TR/2003/WD-xmlquery-full-text-use-cases-20030214/.

[28] World Wide Web Consortium (W3C, M Fernandez et al.) , XQuery 1.0 and XPath 2.0 Data Model. W3C Working Draft, May 2003. http://www.w3c.org/TR/xpath-datamodel/