

XFIRM at INEX 2005: ad-hoc and relevance feedback tracks.

Karen Sauvagnat, Lobna Hlaoua, and Mohand Boughanem

IRIT-SIG,
118 route de Narbonne, F-31 062 Toulouse Cedex 4, France

Abstract. This paper describes experiments carried out with the XFIRM system in the INEX 2005 framework. The XFIRM system uses a relevance propagation method to answer CO and CAS queries. Runs were submitted to the ad-hoc and relevance feedback tracks.

1 Introduction

The approach we used for our participation at INEX 2005 is based on the XFIRM system [6], and uses a relevance propagation method. The XFIRM system was adapted for submitting runs to the ad-hoc track (for CO, CO+S, and CAS tasks) and the relevance feedback track.

2 Experimental setup

2.1 XFIRM data model

The XFIRM system is based on a relevance propagation method. We use a generic data model that allows the implementation of many IR models and the processing of heterogeneous collection. We consider that a structured document sd_i is a tree, composed of leaf nodes ln_{ij} and attributes a_{ij} and simple nodes n_{ij} (all nodes that are not leaf nodes or attributes).

Structured document: $sd_i = (\{n_{ij}\}, \{ln_{ij}\}, \{a_{ij}\})$

In order to easily browse the document tree and to quickly find ancestors-descendants relationships, the model uses a representation of nodes and attributes based on the Xpath Accelerator approach [2].

All leaf nodes are indexed, because even the smallest leaf nodes can be relevant or can give information on the relevance of their ancestors. Intuitively, *title* or *subtitle* nodes are not informative, but if a query term occurs in those nodes, such information can be useful for evaluating the relevance of the ancestor node. Such an approach has other advantages: the index process can be done automatically, without any human intervention and the system will be so able to handle heterogeneous collections automatically; and secondly, even the most specific query concerning the document structure will be processed, since all the document structure is stored.

During query processing, relevance values are assigned to leaf nodes and relevance score of inner nodes are then computed dynamically, thanks to a propagation of leaf nodes score through the document tree. An ordered list of subtrees is then returned to the user.

2.2 Evaluation of leaf nodes scores

Whatever the considered type of queries, a first step in query processing is to evaluate the relevance value of leaf nodes ln according to the query. Let $q = t_1, \dots, t_n$ be this query. Relevance values are computed thanks to a similarity function $RSV_m(q, ln)$, where m is an IR model.

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

Where w_i^q and w_i^{ln} are respectively the weights of term i in query q and leaf node ln .

According to previous experiments [8], we choose to use the following term weighting scheme, which aims at reflecting the importance of terms in leaf nodes, but also in whole documents:

$$w_i^q = tf_i^q \quad w_i^{ln} = tf_i^{ln} * idf_i * ief_i \quad (2)$$

Where tf_i^q and tf_i^{ln} are respectively the frequency of term i in query q and leaf node ln , $idf_i = \log(|D|/(|di| + 1)) + 1$, with $|D|$ the total number of documents in the collection, and $|di|$ the number of documents containing i , and ief_i is the inverse element frequency of term i , i.e. $\log(|N|/|nf_i| + 1) + 1$, where $|nf_i|$ is the number of leaf nodes containing i and $|N|$ is the total number of leaf nodes in the collection.

Inner nodes relevance values are evaluated thanks to one or more propagation functions, which depend on the searching task. Such propagation functions are described in the following sections.

3 CO task

3.1 Inner nodes relevance values evaluation

In our model, each node in the document tree is assigned a relevance value which is function of the relevance values of the leaf nodes it contains. We believe that terms that occur close to the root of a given subtree are more significant for the root element than ones on deeper levels of the subtrees. It seems therefore that the larger the distance of a node from its ancestor is, the less it contributes to the relevance of its ancestor. This affirmation is modelled in our propagation formula by the use of the $dist(n, ln_k)$ parameter. $dist(n, ln_k)$ is the distance between node n and leaf node ln_k in the document tree, i.e. the number of arcs that are necessary to join n and ln_k .

It is also intuitive that the more a node contains relevant leaf nodes, the more it is relevant. We then introduce in the propagation function the $|L_n^r|$ parameter, which is the number of leaf nodes being descendant of n and having a non-zero relevance value (according to equation 1).

A relevance propagation function using these parameters has been tested in the INEX 2004 framework [6]. In the 2005 evaluation campaign, we propose to add two parameters:

- We propose to increase small nodes importance during propagation. Indeed, we think that authors of documents use small nodes to highlight important informations. These nodes can therefore give precious indications on the relevance of their ancestors. In our propagation function, this intuition corresponds to the $\beta(ln_k)$ parameter.
- We introduce the ρ parameter, inspired from work presented in [3], which allows the introduction of document relevance in inner nodes relevance evaluation. The idea behind context is: an element in a relevant document should be better ranked than an identical element in a non-relevant document.

The relevance value r_n of a node n is therefore computed according the following formula:

$$\begin{aligned}
r_n &= \rho * |L_n^r|. \sum_{ln_k \in L_n} \alpha^{dist(n,ln_k)-1} * \beta(ln_k) * RSV(q, ln_k) \\
&\quad + (1 - \rho) * |L^r|. \sum_{ln_k \in L} \alpha^{dist(root,ln_k)-1} * \beta(ln_k) * RSV(q, ln_k) \\
&= \rho * |L_n^r|. \sum_{ln_k \in L_n} \alpha^{dist(n,ln_k)-1} * \beta(ln_k) * RSV(q, ln_k) \\
&\quad + (1 - \rho) * r_{root}
\end{aligned} \tag{3}$$

where ln_k are leaf nodes being descendant of n and L_n is the set of leaf nodes being descendant of n . r_{root} is the relevance score of the *root* element, i.e. the relevance score of the whole document, evaluated with equation 3 with $\rho = 1$. $\beta(ln_k)$ is evaluated as follows:

$$\beta(ln_k) = \begin{cases} l_k/\Delta l & \text{if } dist(n, ln_k) = 1 \text{ and } l_k < \Delta l \\ \log(\Delta l/l_k) & \text{if } dist(n, ln_k) > 1 \text{ and } l_k < \Delta l \\ 1 & \text{else} \end{cases} \tag{4}$$

with l_k the length of node ln_k and Δl the average length of leaf nodes in the collection. If a node is smaller than the average length of leaf nodes, its role in the propagation function is emphasized.

3.2 Runs

CO.Thorough strategy. For the CO.Thorough task, all nodes having a non-zero relevance value are returned by the XFIRM system. We experimented using various values of $\rho \in [0..1]$.

CO.Focussed strategy. In order to reduce/remove nodes overlap, we use two different algorithms:

1. For each relevant path, we keep the most relevant node in the path (around 20% of nodes overlap still remains)
2. For each relevant path, we keep the most relevant node in the path. The results set is then parsed again one time, to eliminate any possible overlap among ideal components.

CO.FetchAndBrowse strategy. In this task, elements are first ranked by the relevance of the document they belong to, and then by their own relevance. We use the following algorithm:

1. relevance values are computed for each document in the collection;
2. relevance values are computed for each node of the collection;
3. documents are ranked by decreasing order of relevance;
4. for each document, elements they contain are ranked by decreasing order of relevance and are returned to users.

Document relevance is computed with the Mercure system [1]. Previous experiments [5] have shown that for a fetch and browse strategy, results are better when evaluating document relevance with the Mercure system (which was developed for this purpose) than with our relevance propagation method (which aims at find elements instead of documents).

3.3 Results

All results described in this paper use the inex1.8 version of the collection, which is the official 2005 collection. However, due to a misunderstanding, our official submissions were obtained with the inex1.6 version of the collection. For information, official submissions are mentioned in italic characters and are followed by the '*' symbol.

CO.THOROUGH strategy. Tables 1 and 2 show the results obtained with different values of ρ .

Table 1. CO.Thorough strategy. Quantisation: Generalised

	nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
$\rho = 1$	0,1684	0,168	0,1772	0,0562	0,1100	0,2231
$\rho = 0.9$ *	<i>0,15</i> *	<i>0,156</i> *	<i>0,174</i> *	<i>0,043</i> *	<i>0,085</i> *	<i>0,188</i> *
$\rho = 0.9$	0,1712	0,1696	0,1786	0,0577	0,1089	0,2179
$\rho = 0.8$	0,1634	0,1845	0,1859	0,0569	0,1057	0,2138
$\rho = 0.7$	0,1727	0,2006	0,1883	0,0569	0,1044	0,2110
$\rho = 0.6$	0,1713	0,2058	0,1928	0,0565	0,1031	0,2078
$\rho = 0.5$	0,1762	0,2036	0,1894	0,0561	0,1019	0,2051
$\rho = 0.4$	0,1802	0,2075	0,1897	0,0557	0,1009	0,2040
$\rho = 0.3$	0,1931	0,2116	0,188	0,0555	0,1001	0,2020
$\rho = 0.2$	0,2049	0,2126	0,1857	0,0553	0,0996	0,2010
$\rho = 0.1$	0,2083	0,2144	0,1868	0,0548	0,0986	0,2011
$\rho = 0$	0,2384	0,2126	0,1862	0,0542	0,0976	0,1981

Best results are obtained with small values of ρ , especially for the strict quantisation function. This seems to show that root relevance (i.e. document relevance) has a high impact on elements relevance.

Table 2. CO.Thorough strategy. Quantisation: Strict

	nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
$\rho = 1$	0,012	0,0299	0,0464	0,0009	0,0012	0,0208
$\rho = 0.9$ *	0,011 *	0,025 *	0,047 *	0,001 *	0,001 *	0,021 *
$\rho = 0.9$	0,012	0,0258	0,0462	0,0012	0,0015	0,0216
$\rho = 0.8$	0,008	0,0329	0,0475	0,0014	0,0016	0,0213
$\rho = 0.7$	0,008	0,0425	0,0514	0,0015	0,0017	0,0216
$\rho = 0.6$	0,008	0,0505	0,0522	0,0016	0,0018	0,0218
$\rho = 0.5$	0,012	0,0569	0,0546	0,0017	0,0019	0,0209
$\rho = 0.4$	0,016	0,0585	0,0555	0,0017	0,0019	0,0206
$\rho = 0.3$	0,024	0,0617	0,0555	0,0017	0,0020	0,0199
$\rho = 0.2$	0,036	0,0633	0,0555	0,0018	0,0020	0,0199
$\rho = 0.1$	0,044	0,0633	0,0563	0,0019	0,0021	0,0199
$\rho = 0$	0,0684	0,0636	0,0579	0,0019	0,0021	0,0194

Table 3. CO.Focussed strategy. Quantisation: Generalised

		nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
Algorithm 1	$\rho = 1$	0,1202	0,1214	0,1279	0,0393	0,0798	0,1543
	$\rho = 0.9$	0,112	0,1073	0,0941	0,0260	0,0609	0,1249
	$\rho = 0.8$	0,1146	0,106	0,093	0,0256	0,06042	0,1208

	$\rho = 0.1$	0,1042	0,094	0,0852	0,0231	0,0577	0,1177
	$\rho = 0$	0,0951	0,0901	0,078	0,0235	0,0607	0,1172
Algorithm 2	$\rho = 1$ *	0,119 *	0,122 *	0,119 *	0,030 *	0,060 *	0,132 *
	$\rho = 1$	0,1364	0,1445	0,1453	0,0396	0,0748	0,1579
	$\rho = 0.9$ *	0,104 *	0,104 *	0,089 *	0,022 *	0,052 *	0,106 *
	$\rho = 0.9$	0,1299	0,1171	0,1021	0,0276	0,0624	0,1271
	$\rho = 0.8$	0,1235	0,1144	0,0988	0,0271	0,0622	0,1258

	$\rho = 0.1$	0,1131	0,1033	0,092	0,0256	0,0613	0,1197
	$\rho = 0$	0,0951	0,0901	0,078	0,0235	0,0607	0,1172

CO.FOCUSSED strategy. Tables 3 and 4 show the results obtained with different values of ρ .

Algorithm 2 (results without any nodes overlap) allows to obtain better results than algorithm 1 for all metrics. As opposed to results obtained for the CO.Thorough strategy, document relevance seems to have no impact on element relevance (best results were obtained with $\rho = 1$).

CO.FETCHBROWSE strategy. Results obtained with the CO.FetchBrowse strategy are described in table 5. Results are good compared to other participants, since we were ranked in the top 5 for both quantisation functions. Moreover results are substantially better for the MAP metric than those obtained for the CO.Thorough strategy (see tables 1 and 2). We observe for example up to 113% increase for the generalised quantisation function with $\rho = 0.9$.

Table 4. CO.Focussed strategy. Quantisation: Strict

		nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
Algorithm 1	$\rho = 1$	0,012	0,016	0,0336	0,0024	0,0034	0,0051
	$\rho = 0.9$	0,014	0,0156	0,0188	0,0030	0,0038	0,0015
	$\rho = 0.8$	0,014	0,0156	0,0196	0,0031	0,0039	0,0011

	$\rho = 0.1$	0,004	0,0056	0,0128	0,0011	0,0016	0,0008
	$\rho = 0$	0	0,004	0,012	0,0009	0,0015	0
Algorithm 2	$\rho = 1$ *	<i>0,011</i> *	<i>0,006</i> *	<i>0,025</i> *	<i>0,002</i> *	<i>0,003</i> *	<i>0,004</i> *
	$\rho = 1$	0,016	0,0112	0,0296	0,0034	0,0046	0,0066
	$\rho = 0.9$ *	<i>0,014</i> *	<i>0,020</i> *	<i>0,028</i> *	<i>0,004</i> *	<i>0,004</i> *	<i>0,002</i> *
	$\rho = 0.9$	0,014	0,0172	0,0204	0,0031	0,0039	0,0018
	$\rho = 0.8$	0,014	0,0172	0,0236	0,0031	0,0039	0,0011

	$\rho = 0.1$	0,004	0,0072	0,0176	0,0013	0,0019	0,0011
	$\rho = 0$	0	0,004	0,012	0,0009	0,0015	0

Table 5. CO.FetchBrowse strategy. ep/gr - MAP-Element metric

	Generalised	Strict
$\rho = 1$	0,1167	0,0063
$\rho = 0.9$ *	<i>0,108</i> *	<i>0,006</i> *
$\rho = 0.9$	0,1229	0,0068
...
$\rho = 0.1$	0,1183	0,0065
$\rho = 0$	0,0731	0,0042

4 CAS task

4.1 Inner nodes relevance value evaluation

The evaluation of a CAS query is carried out as follows:

1. INEX (NEXI) queries are translated into XFIRM queries
2. XFIRM queries are decomposed into sub-queries SQ and elementary sub-queries ESQ , which are of the form: $ESQ = tg[q]$, where tg is a tag name, i.e. a structure constraint, and $q = t_1, \dots, t_n$ is a content constraint composed of simple keywords terms.
3. Relevance values are then evaluated between leaf nodes and the content conditions of elementary sub-queries
4. Relevance values are propagated in the document tree to answer to the structure conditions of elementary sub-queries
5. Sub-queries are processed thanks to the results of elementary sub-queries
6. Original queries are evaluated thanks to upwards and downwards propagation of the relevance weights

Step 3 is processed thanks to formula 1. In step 4, the relevance value r_n of a node n to an elementary subquery $ESQ = tg[q]$ is computed according the

following formula:

$$r_n = \begin{cases} \sum_{ln_k \in L_n} \alpha^{dist(n,ln_k)-1} * RSV(q,ln_k) & \text{if } n \in construct(tg) \\ 0 & \text{else} \end{cases} \quad (5)$$

where the $construct(tg)$ function allows the creation of set composed of nodes having tg as tag name, and $RSV(q,ln_k)$ is evaluated during step 2 with formula 1. The $construct(tg)$ function uses a *Dictionnary* Index, which provides for a given tag tg the tags that are considered as equivalent. For example, a *title* node can be considered as equivalent to a *sub-title* node. This index is built manually. More details about CAS queries processing are can be found in [7].

4.2 Runs

In order to answer the different searching tasks, we used different Dictionary indexes:

- The DICT index is composed of equivalencies given in the INEX guidelines. For example, *ss1*, *ss2* and *ss3* nodes are considered as equivalent to *sec* nodes.
- The ExtendedDICT is composed of very extended equivalencies. For example, *sec*, *ss1*, *ss2* and *ss3* nodes are equivalent to both *p* and *bdy* nodes.

SSCAS strategy. We use the DICT index and results are filtered in order to answer strictly to constraints on the target element and support elements.

VVCAS strategy. We use the EXtendedDICT index, and no filter is applied on results.

SVCAS strategy. We use the DICT index. No filter is applied on results: they match the structure constraint on the target element in a strict way (since the DICT index is used), and their relevance score is eventually increased by the relevance score of results of subqueries on support elements.

VSCAS strategy. We use the DICT index on support elements and the ExtendedDICT on target elements.

4.3 Results

Results for all strategies are showed in tables 6, 7, 8 and 9. We are in the top 10 for almost all metrics.

Table 6. SSCAS strategy

		nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
Generalised	<i>DICT</i> *	<i>0,329</i> *	<i>0,397</i> *	<i>0,374</i> *	<i>0,105</i> *	<i>0,157</i> *	<i>0,258</i> *
	DICT	0.2861	0.2722	0.338	0.109	0.178	0.246
Strict	<i>DICT</i> *	<i>0,325</i> *	<i>0,31</i> *	<i>0,32</i> *	<i>0,016</i> *	<i>0,02</i> *	<i>0,121</i> *
	DICT	0.35	0.329	0.338	0.0166	0.021	0.1169

Table 7. VVCAS strategy

		nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
Generalised	<i>ExtendedDICT</i> *	0,29 *	0,258 *	0,246 *	0,061 *	0,101 *	0,206 *
	ExtendedDICT	0.3047	0.2727	0.2487	0.0687	0.115	0.219
Strict	<i>ExtendedDICT</i> *	0,067 *	0,076 *	0,136 *	0,005 *	0,006 *	0,044 *
	ExtendedDICT	0.0885	0.0756	0.1244	0.0054	0.0059	0.0468

Table 8. SVCAS strategy

		nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
Generalised	<i>DICT</i> *	0,303 *	0,272 *	0,276 *	0,105 *	0,181 *	0,301 *
	DICT	0.2645	0.2758	0.2916	0.1378	0.2318	0.330
Strict	<i>DICT</i> *	0,42 *	0,408 *	0,416 *	0,017 *	0,02 *	0,1 *
	DICT	0.44	0.4571	0.4662	0.017	0.022	0.11

Table 9. VSCAS strategy

		nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
Generalised	<i>DICT+ExtendedDICT</i> *	0,194 *	0,21 *	0,207 *	0,07 *	0,119 *	0,218 *
	DICT+ExtendedDICT	0.237	0.2346	0.2292	0.047	0.069	0.159
Strict	<i>DICT+ExtendedDICT</i> *	0	0,007 *	0,023 *	0,007 *	0,008 *	0,07 *
	DICT+ExtendedDICT	0.1667	0.15	0.15	0.006	0.007	0.05

Results are especially good for the SVCAS strategy, which is not really surprising. Our original model processes CAS queries with a SVCAS strategy: relevance score of target elements are eventually increased by score of results of subqueries on support elements [7]. Relevance propagation seems consequently to be a very good solution for processing CAS queries with a SVCAS strategy.

5 CO+S task

5.1 Inner nodes relevance value evaluation

In the CO+S task, queries are processed as in the CAS task. Nodes relevance values are evaluated using equation 5.

5.2 Runs

+S.THOROUGH strategy and +S.FOCUSSED strategy. We either use the DICT or ExtendedDICT dictionary index, since the aim of the task is to investigate the usefulness of the structural hints.

+S.FETCHBROWSE strategy We follow the same algorithm as the one used for the CO.FETCHBROWSE strategy.

5.3 Results and comparison to the CO task

+S.THOROUGH strategy. Results are not as good as those obtained without structural hints (see table 1 and 2 for comparison).

Table 10. COS.Thorough strategy. Quantisation: Generalised

	nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
<i>DICT</i> *	0,172 *	0,147 *	0,123 *	0,016 *	0,03 *	0,089 *
DICT	0,1759	0,162	0,1422	0,0192	0,0369	0,1022
<i>ExtendedDICT</i> *	0,169 *	0,191 *	0,187 *	0,045 *	0,088 *	0,001 *
ExtendedDICT	0,1787	0,2037	0,206	0,0569	0,1086	0,2166

Table 11. COS.Thorough strategy. Quantisation: Strict

	nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
<i>DICT</i> *	0,024 *	0,037 *	0,047 *	0 *	0 *	0,008 *
DICT	0,0242	0,0321	0,0362	0,0003	0,0004	0,0062
<i>ExtendedDICT</i> *	0,027 *	0,042 *	0,056 *	0,001 *	0,001 *	0,019 *
ExtendedDICT	0,0308	0,0434	0,0532	0,0012	0,0014	0,0210

Table 12. COS.Focussed strategy. Quantisation: Generalised

	nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
DICT	0,1567	0,1362	0,121	0,0458	0,1046	0,1783
<i>ExtendedDICT</i> *	0,144 *	0,128 *	0,127 *	0,031 *	0,069 *	0,143 *
ExtendedDICT	0,1586	0,1497	0,1428	0,0410	0,0855	0,1674

Table 13. COS.Focussed strategy. Quantisation: Strict

	nxCG[10]	nxCG[25]	nxCG[50]	ep/gr - MAP	Q	R
DICT	0,0231	0,0308	0,0345	0,0096	0,01183	0,0166
<i>ExtendedDICT</i> *	0,009 *	0,009 *	0,025 *	0,002 *	0,004 *	0,004 *
ExtendedDICT	0,0154	0,0163	0,0571	0,0032	0,0046	0,0086

+S.FOCUSSED strategy As opposed to results obtained for the +S.THOROUGH strategy, results here are better than those obtained without using structural hints (see tables 3 and 4 for comparison).

Table 14. +S.FetchBrowse strategy. ep/gr - MAP-Element metric

	Generalised	Strict
<i>DICT</i> *	0,111 *	0,015 *
DICT	0,0155	0,0008
<i>ExtendedDICT</i> *	0,0747 *	0,005 *
ExtendedDICT	0,072	0,0058

+S.FETCHBROWSE strategy. As for the +S.THOROUGH strategy, results obtained without using structural hints are better than those obtained for the CO.FETCHBROWSE strategy (see table 5 for comparison).

6 Relevance feedback track

For the RF track, we used three different algorithms that are described below.

6.1 Structure-Oriented Relevance Feedback

Our goal in what we call structure-oriented RF is to enrich the initial query by adding structural constraints. Our approach consists in refining the initial query by extracting from the set of judged elements the structure that could contain the information needed by the user. The idea behind structure-oriented RF is therefore to find for each query, the *appropriate generic structure* which is the generic structure shared by the greatest amount of relevant elements. The generative structure is extracted as follows. Let:

- E^r be the set of relevant elements,
- e_i be an element $\in E^r$,
- e_i be characterized by a path p_i and a score w_i initialized at the beginning of algorithm (and set to 1 for the experiments presented here). p_i is only composed of tag names. For example: */article/bdy/sec*.
- CS be a set of Common Structures, obtained as result.

For each $(e_i, e_j)_{i \neq j} \in E^r \times E^r$, we apply the *SCA* algorithm, which allows to retrieve the smallest common ancestor of e_i and e_j . The path of this smallest common ancestor is then added to the set of common structures CS . The *SCA* algorithm is processed for each pair of E^r elements. The *SCA* algorithm is described below:

```

SCA( $e_i, e_j$ )
Begin
If  $p_i.first = p_j.first$ , then
  if  $p_i.last = p_j.last$ , then  if  $\exists e_p(p_p, w_p) \in CS/p_p = p_i$ 
                                then  $w_p \leftarrow w_p + w_j$ 
                                else  $w_i \leftarrow w_i + w_j$ 
                                 $CS \leftarrow sp_i$ 
  else
    if  $head(p_j) \neq null$ , then   $p'_j \leftarrow head(p_j)$ 
                                 $w'_j \leftarrow w_j/2$ 
                                 $SCA(e_i(p_i, w_i), e'_j(p'_j, w'_j))$ 
                                else  $SCA(e_j, e_i)$ 
End

```

$p.last$ is the last tag of the path p and $head(p)$ is a function allowing to reduce the path p , i.e. to remove the last tag of the path. For example, $head(/article/bdy/section) = /article/bdy$. In our approach, we choose to only compare the $p.last$ tags of elements, since we are looking for component types (i.e. tags) instead of complete paths.

In order to express the new (CAS) query, we then extract the top ranked structure according w_i in the CS set. This structure will be either used as it is in the new query; this form is called **complex form= p** or simplified in a simple tag form called **simple form= $p.last$** . Original query terms are then added to the structural constraint.

Example for a given query, we consider three elements e_1 , e_2 and e_3 judged relevant having the following corresponding paths (we consider a path as a structure):

$p_1 = /article/bdy/sec/ss1$, $p_2 = /article/bdy/sec/ss1/ss2$ and $p_3 = /article/bdy$. Initial scores $w_1 = w_2 = w_3 = 1$ are assigned to structures p_1 , p_2 and p_3 (see table 15). Let CS be the set of Common Structures in which we add the generative

Table 15. The Input of extraction generative structure

Relevant element	Path	Score
$e_1(p_1, w_1)$	$p_1 = /article/bdy/sec/ss1$	$w_1=1$
$e_2(p_2, w_2)$	$p_2 = /article/bdy/sec/ss1/ss2$	$w_2=1$
$e_3(p_3, w_3)$	$p_3 = /article/bdy$	$w_3=1$

structure. At the beginning of the algorithm, CS is empty.

The first step of generative structure extraction consists in:

- checking if the two structures have the same root: (*article*).
- comparing the last tags of p_1 structure and p_2 structure: $p_1.last = ss1$ and $p_2.last = ss2$.
- As the tags are different ($ss1 \neq ss2$), we move to high level of p_2 structure p'_2 ($p'_2 = head(p_2) = /article/bdy/sec/ss1$). p'_2 's score is $w'_2 = w_2/2$.
- $p'_2.last = ss1 = p_1.last$. So, the p_1 structure will be added to CS with $w_1 + w_2/2 = 1 + 1/2$ score.

The second step corresponds to matching of the two structures $p_1 = /article/bdy/sec/ss1$ and $p_3 = /article/bdy$:

- matching in $p_1 \rightarrow p_3$ direction gives empty result.
- We match them in inverse direction ($p_3 \rightarrow p_1$). The Common Structure is retrieved after 2 iterations ($(head(head(p_1))).last = bdy = p_3.last$).
 \Rightarrow The score of p_1 is divided by 2^2 and the structure $p_3 = /article/bdy$ is added in the SC set with $w_3 + w_1/4 = 1 + 1/4$ score.

In the third step, we proceed in matching p_2 and p_3 . The same process as above is applied. The Common Structure is extracted when applying three iterations. The w_2 score is divided by $2^3 = 1/8$.

We notice that p_3 structure is already in the CS set. So, we increment its score by adding $w_2/8$.

The most generative structure is the one with the highest score in CS (see table 16).

Table 16. The Output of extraction generative structure: the CS set

Structure	Score
$/article/bdy/sec/ss1$	$1 + 1/2 = 1.5$
$/article/bdy$	$1 + 1/4 + 1/8 = 1.375$

We select in this example `/article/bdy/sec/ss1`.

Let Q be an initial CO query: Q= "Information Retrieval". The new CAS query is reformulated as follow:

- Structure-Oriented RF in its simple form: "**ss1**[about(., "Information Retrieval")]".
- Structure-Oriented RF in its complex form: "`//article/bdy/sec/ss1`[about(., "**Information Retrieval**")]".

Let Q' be a structured query (COS or VVCAS): Q'= "article[about(., "Information Retrieval")]". The new query is:

- Structure-Oriented RF: "`article`[about(., "**Information Retrieval**")] **OR** `ss1`[about(., "**Information Retrieval**")]".

6.2 Content-Oriented Relevance Feedback

Our Content-Oriented Relevance Feedback approach is based on the Rocchio' algorithm [4]. Our aim is to extract the most expressive terms from relevant elements. The content-oriented RF processes as follows :

- We consider the set of relevant elements (E^r) : $E^r = e_1^r, e_2^r, \dots, e_k^r, \dots, e_m^r$,
- A relevant element e_k^r is composed of a set of leaf nodes (ln_j) : $e_k^r = ln_1^k, \dots, ln_j^k, \dots, ln_n^k$
- A leaf node ln_j^k is a sequence of terms: $ln_j = \{t_{ij}\}$.

Each term is assigned a score according to the following formula:

$$score(t_{ij}, ln_j^k) = \frac{tf_i^j}{size(ln_j)} \quad (6)$$

Where tf_i^j is the frequency of term t_i in leaf node ln_j^k and $size(ln_j)$ is the number of terms in ln_j .

We then compute the score of terms for each relevant element. For each term, we sum its scores in different leaf nodes.

$$score(t_i, e_k^r) = \sum_{ln_j \in e_k^r} score(t_i, ln_j) \quad (7)$$

As a result, we obtain a set of expressive words for each element judged as relevant. Best terms are selected according to the following formula:

$$score(t_i) = \sum_{e_k^r \in E^r} score(t_i, e_k^r) \quad (8)$$

The new query is finally composed of terms ranked in the top k according to formula 8, that are added to the original query terms.

6.3 Content-and-Structure-Oriented Relevance Feedback

In this approach, we propose to combine the structure-oriented Relevance Feedback method and the content-oriented Relevance Feedback described above. The new query (CAS) is composed of the most appropriate generic structure (complex or simple form) and of terms ranked in the top k according to formula 8, that are added to the original query terms.

6.4 Runs and Results

Relative Improvement (RI) and Absolute Improvement (AI) according $MANxCG[50]$, $MANxCG[1500]$ and $MAep$ measures using strict and generalized quantizations are used to evaluate our runs.

Due to a misunderstanding, official results use `inex1.6` version of the collection. They are presented in table 17, applying the Content and Structure-Oriented approach. The results differ w.r.t the query types (if we consider $MAep$ measure,

Table 17. Official Result of Content and Structure-Oriented RF

	MANxCG[50] strict	MANxCG[50] gen	MANxCG [1500] strict	MANxCG [1500] gen	MAep strict	MAep gen
AI-VVCAS-RF	-0.0397	-0.0813	0.0437	-0.0251	-0.0020	-0.0008
RI-VVCAS-RF	-0.5231	-0.3135	0.1975	-0.1007	-0.0795	-0.0142
AI-COS-RF	-0.0014	-0.0176	0.0168	0.078	0.0001	0.0202
RI-COS-RF	-0.1728	-0.1371	0.1502	0.6933	0.0421	1.8222
AI-CO-RF	-0.0169	-0.0597	-0.1679	-0.1494	-0.0072	-0.0324
RI-CO-RF	-0.6576	-0.3703	-0.723	-0.551	-0.7217	-0.7860

CO: very negative improvement, CO+S important improvement and VVCAS negative improvement). Such observations are not enough to conclude. We therefore tested the three approaches presented above for all 3 CO. Thorough, COS.Thorough and VVCAS task using the `inex1.8` version of the collection.

Impact of Structure-Oriented RF. According to table 18, we notice that the Structure-Oriented approach has a positive impact in case of COS queries (RI gen($MAep = 1,6708$, $MANxCG@1500=0.808$)). This confirms our hypothesis : *adding structural constraints leads to refining query*. This hypothesis is however not proved in case of CO and VVCAS queries and we notice that results obtained by adding a complex form are worse than those using a simple form. We can explain the negative impact in CO case by the fact that the user prefers more than one element type. In the case of VVCAS queries, the reformulation query using the *OR* operator may not be appropriate.

Impact of Content-Oriented RF. We have evaluated the adding of the 10 top terms to initial queries. According to table 19, we notice a positive improvement in case of CO+S queries (generalized quantization) and in case of VVCAS (strict

Table 18. Impact of Structure-Oriented RF

	MANxCG [1500] strict	MANxCG [1500] gen	MAep strict	MAep gen
AI-VVCAS-RF	-0.0929	-0.0748	0.0000	-0.0124
RI-VVCAS-RF	-0.4200	-0.2914	-0.0063	-0.1810
AI-COS-RF	0.0449	0.0941	0.0005	0.0208
RI-COS-RF	0.3630	0.8394	1.5790	1.0821
AI-CO-RF (simple form)	-0.0822	-0.0999	-0.0007	-0.0189
RI-CO-RF(simple form)	-0.3540	-0.3685	-0.5391	-0.3397
AI-CO-RF (complex form)	-0.1653	-0.1792	-0.0011	-0.0348
RI-CO-RF(complex form)	-0.7119	-0.6610	-0.9014	-0.6240

and generalized quantizations). These observations confirm the results obtained in classical IR. We do not observe improvement in case of CO queries. Therefore, it is necessary to look for other methods to extract and extend unstructured queries.

Table 19. Impact of Content-Oriented RF

	MANxCG [1500] strict	MANxCG [1500] gen	MAep strict	MAep gen
AI-VVCAS-RF	0.222	0.0095	0.0004	0.0022
RI-VVCAS-RF	0.1004	0.0370	0.0698	0.0321
AI-COS-RF	-0.0143	0.0515	0.0001	0.0103
RI-COS-RF	-0.1156	0.4594	0.01568	0.5343
AI-CO-RF	-0.1218	-0.1201	-0.0007	-0.0333
RI-CO-RF	-0.5245	-0.4430	-0.5787	-0.5977

Impact of Content and Structure-Oriented RF. According to above observations, the combined RF seems efficient in case of CO+S and VVCAS task. Indeed, table 20 shows that the relative improvement in generalized quantization is more than 50% in the case of CO+S queries and 7% in the case of VVCAS queries. These results confirm the classical IR results and the generative structure effectiveness to refine structured queries. Results are still negative in the case of CO queries.

Discussion We compared different approaches according to each query type. We notice that improvements obtained by Content and Structure Oriented RF are more important in case of VVCAS query type. The Structure-Oriented RF approach is the most effective according to CO+S queries (in generalized quantization). In *ep/gr* curve of CO queries (figure 1), we notice that Structure-Oriented RF are effective according the precision effort values where ($gr \in [0.5, 0.53]$, $[0.59, 0.68]$ and $[0.73, 0.75]$). RF is consequently partially benefic for CO queries. We will look for methods extending these intervals in our future works.

Table 20. Impact of Content and Structure-Oriented RF

	MANxCG [1500] strict	MANxCG [1500] gen	MAep strict	MAep gen
AI-VVCAS-RF	0.0228	0.0082	0.0005	0.0031
RI-VVCAS-RF	0.01031	0.0319	0.0985	0.0451
AI-COS-RF	-0.0132	0.0535	0.0001	0.0107
RI-COS-RF	-0.1067	0.4773	0.4588	0.5537
AI-CO-RF	-0.1960	-0.1812	-0.0010	-0.0415
RI-CO-RF	-0.7736	-0.7451	-0.8441	-0.6684

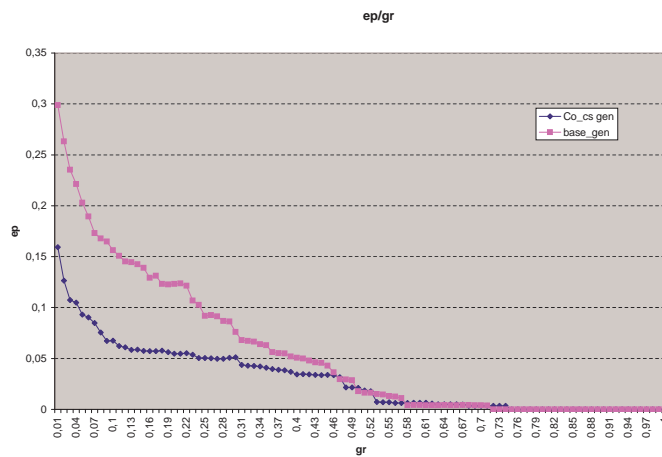


Fig.1. Comparison of the base run and of the Structure Oriented RF approach in case of CO queries.

References

1. M. Boughanem, T. Dkaki, J. Mothe, and C. Soule-Dupuy. Mercure at TREC-7. In *Proceedings of TREC-7*, 1998.
2. T. Grust. Accelerating XPath location steps. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA*, pages 109–120, 2002.
3. Y. Mass and M. Mandelbrod. Component ranking and automatic query refinement for XML retrieval. In *Proceedings of INEX 2004, Springer*, pages 73–84, 2005.
4. J. Rocchio. *Relevance feedback in information retrieval*. Prentice Hall Inc., Englewood Cliffs, NJ, 1971.
5. K. Sauvagnat. *Modle flexible pour la recherche d'information dans des corpus de documents semi-structurs*. PhD thesis, Toulouse : Universit Paul Sabatier, 2005.
6. K. Sauvagnat and M. Boughanem. Using a relevance propagation method for adhoc and heterogeneous tracks at inex 2004. In *Proceedings of INEX 2004, Springer - LNCS - , Dagstuhl, Germany*, pages 337–348, March 2004.
7. K. Sauvagnat, M. Boughanem, and C. Christment. Using relevance propagation for processing content-and-structure queries. *Information Systems, special issue on SPIRE 04, to appear*, 2006.
8. K. Sauvagnat, L. Hlaoua, and M. Boughanem. Xml retrieval: What about using contextual relevance ? In *Proceedings of SAC 2006, Dijon, to appear*, 2006.