Component Ranking and Automatic Query Refinement for XML Retrieval

Yosi Mass and Matan Mandelbrod

IBM Research Lab,
Haifa 31905, Israel
{yosimass, matan}@il.ibm.com

Abstract. Queries over XML documents challenge search engines to return the most relevant XML components that satisfy the query concepts. In a previous work we described a component ranking algorithm that performed relatively well in INEX'03. In this paper we show an improvement to that algorithm by introducing a document pivot that compensates for missing terms statistics in small components. Using this new algorithm we achieved improvements of 30%-50% in the Mean Average Precision over the previous algorithm. We then describe a general mechanism to apply known Query Refinement algorithms from traditional IR on top of this component ranking algorithm and demonstrate an example such algorithm that achieved top results in INEX'04.

1 Introduction

While in traditional IR we are used to get back entire documents for queries, the challenge in XML retrieval is to return the most relevant components that satisfy the query concepts. The INEX initiative[4] sub classified this task into two sub tasks; Content only (CO) task and Content and Structure (CAS) task. In a CO task the user specifies queries in free text and the search engine is supposed to return the most relevant XML components that satisfy the query concepts. In a CAS task the user can limit query concepts to particular XML tags and to define the desired component to be returned using XPath[11] extended with an about() predicate.

In order to realize the problem in ranking XML components we first examine a typical class of IR engines that use *tf-idf* [8] to perform document ranking. Those engines maintain an inverted index in which they keep for each term among other things the number of documents in which it appears (*df*) and its number of occurrences in each document in the collection (*tf*). These statistics are then used to estimate the relevance of a document to the query by measuring some distance between the two. To be able to return a component instead of a full document search engines should modify their data structures to keep statistics such as *tf-idf* at the component level instead of at the document level. This is not a straight forward extension since components in XML are nested and the problem is how to keep statistics at the component level such that it handles components nesting correctly.

In INEX'03 we described a method [6] for component ranking by creating separate indices for the most informative component types in the collection. For example we created an index for full articles, an index for all sections, for all paragraphs etc. This approach solved the problem of statistics of nested components since in each index we have now components from same granularity so they are not nested. While this approach solved the problem of nested components it introduced a deficiency that could distort index statistics. The problem is that the fine grained indices lack data that is outside their scope which is not indexed at all. For example the *articles* index contains 42,578,569 tokens while the *paragraphs* index contains only 31,988,622 tokens. This means that in the *paragraphs* index ~25% of the possible statistics is missing so for example a term with a low *df* based on the indexed tokens may actually be quite frequent outside the paragraphs so its actual *df* should be higher.

In this paper we describe a method to compensate for this deficiency using document pivot. Using this method we got a consistent improvement of 30%–50% in the mean average precision (MAP) for both INEX'03 and INEX'04 CO topics. On top of this improvement we achieved further improvement by applying Automatic Query Refinement (AQR) to the component ranking algorithm. AQR was studied in [7] in the context of traditional IR engines. The idea there is to run the query in two rounds where highly ranked results from the first round are used to add new query terms and to reweigh the original query terms for the second round. We show how to adopt such AQR algorithms on top of the XML component ranking algorithm.

The paper is organized as follows – in section 2 we describe the document pivot concept and in section 3 we describe how to adopt AQR methods from traditional IR to XML retrieval systems. In section 4 we describe our inverted index and our CO and CAS runs. We conclude in section 5 with discussion of the approaches and with future directions.

2 Component Ranking with Document Pivot

We start by briefing our component ranking approach from INEX'03 as described in [6] and then we show how it was improved using the document pivot concept. As discussed above the problem in XML component ranking is how to keep statistics at the component level such that it handles components nesting correctly. In [6] we solved that problem by creating different indices for the most informative component types. We created an index for articles, index for sections, index for sub sections and index for paragraphs. For simplicity we discuss now our approach for the CO topics.

For a given CO topic we run the query in parallel on the set of indices and get back a result set from each index with components of that index sorted by the relevance score. So we get a sorted list of articles, a sorted list of section and so on. We then described a method for comparing the different result sets so that we can merge the sets into a single sorted list of all component types. Why do we get different scores in each result set? Our scoring method is based on the vector space model where both the query Q and each document *d* are mapped to vectors in the terms space and the relevance of *d* to Q is measured as the cosine between them using the *tf-idf* statistics as described in Fig. 1 below.

$$score(Q,d) = \frac{\sum_{t_i \in Q \cap d} w_Q(t_i) * w_d(t_i) * idf(t_i)}{\|Q\| * \|d\|}$$

$$w_Q(t) = \frac{\log(TF_Q(t))}{\log(AvgTF_Q)} \qquad w_D(t) = \frac{\log(TF_d(t))}{\log(AvgTF_d)}$$

$$idf(t) = \log\left(\frac{\#DocumentsIntheCollection}{\#DocumentsContaining(t)}\right)$$

Fig. 1. Document scoring function

$TF_Q(t)$ is the number of occurrences of *t* in *Q* and *$TF_d(t)$* is the number of occurrences of *t* in *d*.

$AvgTF_Q$ is the average number of occurrence of all query terms in *Q* and *$AvgTF_d$* is the average number of occurrence of all terms in *d*.

$||Q||$ is the number of unique terms in *Q* and $||d||$ is the number of unique terms in *d*, both scaled by the average document length in the collection.

It can be seen that while scores of components in each index are comparable to each other, scores in different indices are at a different scale. For example the *articles* index has 12,107 components so the *idf* of a relatively rare term is not very large compared to its *idf* in the *paragraphs* index which has 646,216 components. In addition the average document length (number of unique tokens) in the *articles* index is 900 while the average document length in the *paragraphs* index is 37. Since ||d|| and ||Q|| are scaled by the average document length then the denominator of scores in the *paragraphs* index is much lower than in the *articles* index. Combining the *idf* difference and the length normalization difference shows why scores of components in the *paragraphs* index are much higher than scores of components in the *articles* index.

In order to compare the scores in different result sets we described in [6] a normalization formula that ensures absolute numbers that are index independent. This is achieved by each index computing *score(Q,Q)* which is the score of the query itself as if it was a document in the collection. Since the score measures the cosine between vectors, then the max value is expected between two identical vectors. Each index therefore normalizes all its scores to its computed *score(Q,Q)*. The normalized results are then merged into a single ranked list consisting of components of all granularities.

While the approach of creating independent indices solved the problem of overlapping data it introduced another deficiency of missing data. The fine grained indices lack data that is outside their scope which is not indexed. For example the *articles* index contains 42,578,569 tokens while the *paragraphs* index contains only 31,988,622 tokens. The missing data in the fine grained indices can distorts the *idf* statistics of the collection and therefore may affect the quality of the results.

To fix that problem we use this year a concept first mentioned in [9] which uses a document pivot (DocPivot) factor to scale the final component score by the score of its containing article. The final score of a component with original score S_c and with its full article score S_a is then

DocPivot * S_a + (1 – DocPivot) * S_c.

Note that the idea of Pivoted document length normalization was first introduced in [10] but it was mentioned there in the context of normalizing scores inside a single index. We do apply pivoted normalization on ||Q||, ||d|| to compute Score(Q,d) as in Fig.1. on each index separately but we need the new DocPivot concept to scale results between the separate indices.

Assuming that the full *articles* index is the first index then the overall algorithm to return a result set for a given query Q is given in Fig. 2 below. Step c is the new step introduced by the DocPivot.

```
1   For each index i
        a.  Compute the result set $Res_i$ of running Q on index i
        b.  Normalize scores in $Res_i$ to [0,1] by normalizing to score(Q,Q)
        c.  Scale each score by its containing article score from $Res_0$
2   Merge all $Res_i$'s to a single result set Res composed of all components sorted
    by their score
```

Fig. 2. Component ranking algorithm

We experimented with several values of DocPivot on the 2003 CO topics using the inex_eval tool [5] with strict metric[1] and got the graph marked as 2003 in Fig. 3 below.

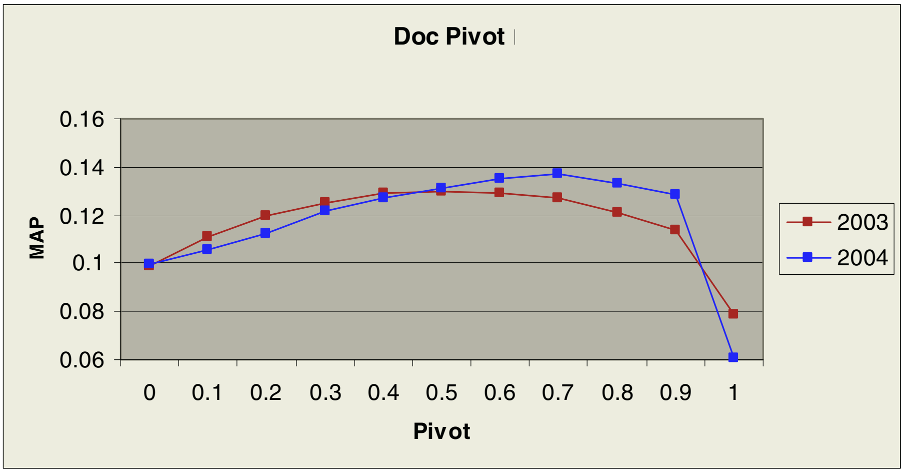

Fig. 3. Doc pivot on 2003/2004 data

[1] The strict metric considers only elements that were assessed as highly relevant.

The MAP with DocPivot = 0 is the result we achieved in our 2003 official submission. We can see that with DocPivot=0.5 we get improvements of 31% over the base 2003 run so we used that value for our 2004 runs.

Later when the 2004 assessments were available we tried those values on the 2004 CO topics using the same metric and got the best MAP for DocPivot = 0.7 (the 2004 graph in Fig. 3) which is 52% improvements over the base run with no DocPivot.

3 Automatic Query Refinement for XML

In this section we describe how to apply Automatic Query Refinement (AQR) on top of our XML component ranking algorithm. AQR was studied in [7] in the context of traditional IR engines. The idea is to run the query in two rounds where highly ranked results from the first round are used to add new query terms and to reweigh the original query terms for the second round. We show now a method to adopt such AQR algorithms on top of our XML component ranking algorithm.

Assume we have an AQR algorithm that can be used to refine query results. Since we have separate indices for different component granularities we can run the AQR algorithm on each index separately. The modified XML component ranking algorithm is described in Figure-4.

```
1   For each index i
        a.  Compute the result set $Res_i$ of running Q on index i
        b.  Apply AQR algorithm on $Res_i$
        c.  Normalize scores in $Res_i$ to [0,1] by normalizing to score(Q,Q)
        d.  Scale each score by its containing article score from $Res_0$
2   Merge all $Res_i$'s to a single result set Res composed of all components sorted by their
    score
```

Fig. 4. Component ranking with AQR

We add step 1.b which is the query refinement step and the rest of the algorithm continues as in the simple case by normalizing and scaling scores in each index and finally merging the result sets.

We describe now a specific AQR algorithm that we used in INEX'04 and discuss some variants of its usage in our XML component ranking algorithm. The AQR algorithm we used is described in [2]. The idea there is to add Lexical Affinity (LA) terms to the query where a Lexical Affinity is a pair of terms that appear close to each other in some relevant documents such that exactly one of the terms appears in the query. The AQR is based on the information gain (IG) obtained by adding lexical affinities to the query. The IG of a lexical affinity L on a set of documents D with respect to a query Q denotes how much L separates the relevant documents in D from the non relevant documents for the query Q. IG is defined as:

$$\boxed{IG_{Q,D}(L) = H_Q(D) - \left[\frac{|D^+|}{|D|} H_Q(D^+) + \frac{|D^-|}{|D|} H_Q(D^-)\right]}$$

Fig. 5. Information Gain of a Lexical Affinity

where $D^+ \subseteq D$ is the set of documents containing L and $D^- \subseteq D$ is the set of documents not containing L. $H_Q(X)$ is the entropy (or degree of disorder) of a set of documents X and is defined as

$$\boxed{H_Q(X) = -p_Q(X)\log(P_Q(X)) - (1 - p_Q(X))\log(1 - p_Q(X))}$$

Fig. 6. Entropy of a group

where $p_Q(X)$ is defined as the probability of a document chosen randomly from X to be relevant to Q. Let $R^+ \subseteq D^+$ be the set of relevant documents in D^+ then $p_Q(D^+) = |R^+| / |D^+|$ and similarly $p_Q(D^-) = |R^-| / |D^-|$. The problem is that we don't know R^+ and R^- so we use the scoring function as an approximation for $p_Q(D^+)$ ($p_Q(D^-)$. We take $p_Q(D^+)$ to be sum of scores of documents in D^+ (a score is between 0 and 1) divided by the max sum of score they could get which is $|D^+|$. We do the same estimation for $p_Q(D^-)$. Note that $H_Q(D)$ is independent of L so to compare the IG of two LAs we don't have to compute it.

The AQR procedure works as follows: It gets the result set obtained by running the search engine on the query Q (algorithm step 1 in Figure-4) and additional 4 parameters ***(M, N, K, α)*** that are explained below. The AQR first constructs a list of candidate LAs that appear in the top ***M*** highly ranked documents from the result set. Then it takes D to be the set of the top ***N*** (N >> M) highly rank documents and finds the ***K*** LAs with the highest IG on that set D. Those LAs are then added to the query Q and their contribution to *score(Q, d)* for each d is calculated as given by Fig. 1. So for each such new LA we need to calculate its $w_Q(t) * w_d(t) * idf(t)$. Since the added LAs don't appear in the original query Q we take their $TF_Q(t)$ to be the given parameter ***α*** so $w_Q(t) = \alpha / \log(AvgTF_Q)$ for the newly added LAs. We can have several variants for using the above AQR algorithm in the XML component ranking algorithm.

1. The AQR procedure can be applied on each index separately using same (M, N, K, α) parameters or index specific (M, N, K, α) parameters. In this variant different LAs are added to the query for each index.

2. We can apply the first part of the AQR using (M, N, K) on the full *articles* index to find the best LAs. Then apply the last part of the AQR that does the re ranking (with the parameter α) on each index. using the LAs that were extracted from the *articles* index.

The motivation for the 2nd variant is that most informative LAs can be obtained on the full *articles* index since it has the full collection data. In section 4 we describe the (M, N, K, α) parameters used in our runs.

4 Runs Description

We describe now our indices setup and the runs we submitted for the CO, CAS and NLP tracks.

4.1 Index Description

Similar to last year[6] we have created six inverted indices for the most informative components which are {article, sec, ss1, ss2, {p+ip1}, abs}. We removed XML tags from all indices except from the {article} index where they were used for checking CAS topic constraints. Content was stemmed using a Porter stemmer and components with content smaller than 15 tokens[2] were not indexed in their corresponding index.

4.2 CO Runs

Each CO topic has 4 parts : <title>, <description>, <narrative> and <keywords>. This year we could use only the <title> for formulating the query to our search engine. Due to the loosely interpretation of topics as appear in [5] we ignored '+' on terms and we ignored phrase boundaries and instead we use the phrase's terms as regular terms. We still treated '-' terms strictly namely components with '-' terms were never returned.

For example topic 166

```
<title>+"tree edit distance" + XML - image </title>
```

is executed as

```
tree edit distance XML -image
```

We submitted three runs where two of them were ranked 1st and 2nd among the official INEX CO runs. See table 1 below.

4.2.1 Doc Pivot Run

In the run titled CO-0.5 we implemented the Component ranking algorithm as described in Figure-2 using DocPivot=0.5. This run was ranked 2nd in the aggregate metric.

4.2.2 AQR Run

In the run titled CO-0.5-LAREFIENMENT we implemented our AQR algorithm from Figure-4 using M= 20, N = 100, K = 5 and α = 0.9 on all indices. We have imple-

[2] We count 15 tokens without tags. This is roughly equivalent to counting 20 tokens with the tags which is a magic number we used last year and that was used by some other participants.

mented the first algorithm variant where each index computes its own LAs to add. This run was ranked 1st using the CO aggregate metric. We leave for future work experiments with more parameter settings with the two algorithm variants.

Table 1. CO table

TASK:CO			
rank	**Institute**	**avg**	**overlap(%)**
1.	IBM Haifa Research Lab(CO-0.5-LAREFIENMENT)	0.1437	80.89
2.	IBM Haifa Research Lab(CO-0.5)	0.1340	81.46
3.	University of Waterloo(Waterloo-Baseline)	0.1267	76.32
4.	University of Amsterdam(UAms-CO-T-FBack)	0.1174	81.85
5.	University of Waterloo(Waterloo-Expanded)	0.1173	75.62
6.	Queensland University of Technology(CO_PS_Stop50K_099_049)	0.1073	75.89
7.	Queensland University of Technology(CO_PS_099_049)	0.1072	76.81
8.	IBM Haifa Research Lab(CO-0.5-Clustering)	0.1043	81.10
9.	University of Amsterdam(UAms-CO-T)	0.1030	71.96
10.	LIP6(simple)	0.0921	64.29

Some example LAs that were added to queries:

1. For topic 162:

   Text and Index Compression Algorithms

   We got LA pairs (compress, huffman), (compress, gigabyte), (index, loss).
2. For topic 169:

   +"Query expansion" +"relevance feedback" +web

   We got (query, search), (relevance, search), (query, user), (query, result).

4.3 CAS Runs

We applied an automatic translation from XPath[11] to XML Fragments[1] which is the query language used in our search engine. XML Fragments are well-formed XML segments enhanced with

- '+/-' on XML tags and on content
- Phrases on content (" ")
- Parametric search on XML tag's value
- An empty tag (<>) that is used as parenthesis.

We can view any XML Fragment query as a tree[3] with the semantics that at each query node, '+' children must appear, '-' children should not appear and others are optional and only contribute to ranking. If a node doesn't have '+' children then at least one of its other (non '-') children must appear.

For example the query

```
<article>
        <abs>classification</abs>
        <sec>experiment compare</sec>
</article>
```

will return articles with *classification* under <abs> **or** with *experiment* or *compare* under <sec>. Note that the default semantics in XML Fragments is OR unless there are '+'s.

The same query with '+' on the tags -

```
<article>
        +<abs>classification</abs>
        +<sec>experiment compare</sec>
</article>
```

will return articles with *classification* under <abs> **and** with *experiment* or *compare* under <sec>.

Finally the query

```
<article>
        +<abs>classification</abs>
        <sec>experiment compare</sec>
</article>
```

will return only articles with *classification* under <abs>. Articles with *experiment* or *compare* under <sec> will be returned with higher ranking since the child <sec>experiment compare</sec> is optional.

The empty tag is used as a kind of parenthesis so the query

```
<title>
         <>+network +security</>
         <>+database +attributes</>
</title>
```

will return documents with *network* and *security* **or** with *database* and *attributes* under the <title> while the query

[3] For a query given as several disjoint fragments we add a dummy <root> node to make the all query a valid XML data.

```
<title>
        +<>network security</>
        +<>database attributes</>
</title>
```

will return documents with *network* or *security* **and** with *database* or *attributes* under its <title>.

The automatic transformation from an INEX modified XPath expression of the form

```
//path1[path1Predicates]//path2[path2Predicates]
```

to XML Fragments works as follows: It first creates a query node <path1> with two children: The first is a mandatory empty tag (+<>) surrounding path1Predicates and the second is the node <path2> prefixed with a '+'. The path1 and path2 Predicates are translated to nodes where 'about' predicates for the current node *('about(,. "text")')* are transformed to just *text* and about predicates for sibling nodes *('about(//path, "text")')* are transformed to *<path>text</path>*. For example the INEX CAS topic 131

```
<title>//article[about(.//au,"Jiawei Han")]//abs[about(.,"data  mining")]</title>
```

is translated to the following XML Fragments query.

```
+<article>
      +<>
        <au>"jiawei han"</au>
      </>
      +<abs>
        +<>
              "data mining"
           "</>
      </abs>
</article>
```

To support AND/OR between XPath predicates we use the empty tag where predicates that are ANDed are transformed to XML Fragments under '+<>' tag and predicates that are ORed are transformed to XML Fragments under <> with no prefix. For example topic 134.

```
<title>//article[(about(.,"phrase search") OR about(.,"proximity search") OR
                        about(., "string matching")) AND
                   (about(.,tries) OR about(.,"suffix trees") OR
                        about(.,"PAT arrays"))]//sec[about(.,algorithm)]
</title>
```

is transformed to

```
<article>
     +<>
       +<>
          <>"phrase search"</>
          <>"proximity search"</>
          <>"string matching"</>
       </>
       +<>
          <>"tries"</>
          <>"suffix trees"</>
          <>"pat arrays"</>
       </>
      </>
```

```
        +<sec>
         +<>algorithm</>
        </sec>
</article>
```

The interpretation of structure constraints in a CAS topic can vary from *Strict* interpretation (SCAS) where all structure constraints should be met to a loosely *Vague* interpretation (VCAS) where structure constraints are just a hint. In this year it was decided to follow the later VCAS flavor so in our runs we ignored the '+' on tags and similar to the CO case we ignored '+' on content and phrase boundaries. Ignoring '+' changes everything to OR semantics therefore the empty tags have no meaning and can be ignored. For example the above topic 131 is then equivalent to

```
<article>
        <au>jiawei han</au>
        <abs>data mining</abs>
</article>
```

We still keep the XML Fragments semantics that nodes with a single child must have that child so the above query will return only results which have *jiawei* or *han* under <au> or that have *data* or *mining* under the <abs>.

To decide which element to return we followed the XPath target element semantics that defines the last element in the XPath expression as the element to be returned up to the equivalent tags as defined in [5]. We run the CAS topics using a minor modification of step 1 in the algorithm in Figure-2 above: The *articles* index in addition to creating its result set also check the query constraints and mark valid components to be returned. The other indices then return in their results set only components that were marked valid by the *articles* index.

Obeying the target element constraint resulted in a low 38% overlap and as a result our official run got low MAP of 0.065 in the aggregate inex_eval metric[5]. It seems like assessors ignored the target elements as for example in the above topic 131 while <abs> was defined as the target element still many full articles were assessed as most relevant for that topic. Later we tried to weakness the target element constraint and return more elements and we got much better results of 0.120 MAP.

4.4 NLP Runs

We submitted one CO run and one CAS run. For the CO run we used the topic's <description> part and just applied the algorithm from Figure-2 with DocPivot=0.5. This run got MAP of 0.1286 using the aggregate inex_eval metrics[5]. For the CAS run we similarly used the topic's <description> with same DocPivot=0.5 but ignored the XPath target element as if it was a CO topic. This run got MAP of 0.05.

5 Discussion

We have presented two extensions to our last year's XML component ranking algorithm. The first extension introduces a document pivot that scales scores of components by the score of their containing article. This method achieved improvements of 31% over our base CO run in INEX'03 and 52% over our base CO run in INEX'04.

We then described an algorithm to apply existing AQR algorithms on top of our XML component ranking algorithm and demonstrated an example such AQR method using Lexical Affinities with Maximal Information Gain. Our two runs that implemented those extensions were ranked 1st and 2nd in the CO track. The space of possible AQR parameter combinations and the variants for their usage in XML is quite large and we still have to explore the best combination that would give best results. For Vague CAS (VCAS) we still need to find the correct balance of how much to constrain the structure and the target element. Some initial tests already improved our MAP by an order of magnitude over our official runs.

Acknowledgement

We would like to thank the INEX organizers for the assessment tool and for the inex_eval tool they have supplied.

References

1. Broder A.Z., Maarek Y., Mandelbrod M. and Y. Mass (2004): "Using XML to Query XML – From Theory to Practice". In Proceedings of RIAO'04, Avignon France, Apr , 2004.
2. Carmel D., Farchi E., Petruschka Y., Soffer A.: Automatic Query Refinement using Lexical Affinities with Maximal Information Gain. In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2002.
3. Carmel D., Maarek Y., Mandelbrod M., Mass Y., Soffer A.: Searching XML Documents via XML Fragments, In Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Toronto, Canada, Aug. 2003
4. INEX, Initiative for the Evaluation of XML Retrieval, http://inex.is.informatik.uni-duisburg.de
5. INEX'04 Participants area, http://inex.is.informatik.uni-duisburg.de:2004/internal/
6. Mass Y., Mandelbrod M.: Retrieving the most relevant XML Component, Proceedings of the Second Workshop of the Initiative for The Evaluation of XML Retrieval (INEX), 15-17 December 2003, Schloss Dagstuhl, Germany, pg 53-58
7. Ruthven I., Lalmas M. : A survey on the use of relevance feedback for information access systems, Knowledge Engineering Review, 18(1):2003.
8. Salton G. : Automatic Text Processing – The Transformation, Analysis and Retrieval of Information by Computer, Addison Wesley Publishing Company, Reading, MA, 1989.
9. Sigurbjornsson B., Kamps J., Rijke M. : An element based approach to XML Retrieval, Proceedings of the Second Workshop of the Initiative for The Evaluation of XML Retrieval (INEX), 15-17 December 2003, Schloss Dagstuhl, Germany, pg 19-26.
10. Singhal A., Buckley C., Mitra M.. : Pivoted document length normalization, *Proceedings of SIGIR'96,* pp 21--29, 1996.
11. XPath – XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath2