# Searching in an XML Corpus Using Content and Structure*

Yiftah Ben-Aharon Sara Cohen Yael Grumbach Yaron Kanza
Jonathan Mamou Yehoshua Sagiv Benjamin Sznajder Efrat Twito

The Selim and Rachel Benin
School of Engineering and Computer Science
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
{yiftahb, sarina, yagrum, yarok, mamou, sagiv, sznajder, efrat111}@cs.huji.ac.il

## ABSTRACT

This paper presents a system for XML retrieval. The approach consists of a variety of Information Retrieval techniques augmented with the ability to give weights to different fragments of a document, based on the tags. Specifically, term frequencies, inverse document frequencies, proximity among occurrences of keywords, and similarity between keywords and words from the given document, are used. Each technique has been implemented as a separate ranker and the final ranking is done by merging the results of the various rankers.

## 1. INTRODUCTION

An XML document has a structure in addition to content, and an XML search engine should be capable of taking advantage of the structure in order to improve the quality (i.e., precision and recall) of the results. The structure may also be incorporated into the topic (i.e., query) in two ways. First, the topic may include conditions that relate content to structure (e.g., some keyword should appear in the title of the document). Second, the topic may specify the exact fragment of the document that should be returned as an answer. Even if the topic does not have any hint about the structure, the search engine should still be able to find not just the relevant documents, but also the most relevant fragment (or fragments) within each document.

Several different paradigms have been proposed recently for searching XML documents. In XRANK [7], the main idea is a generalization of the Page-Rank [4] technique of Google [1]. In XSEarch [6], the emphasis is on retrieving only those answers that consist of *semantically related* nodes. Neither one of these approaches is suitable for the INEX corpus, which consists of articles from the IEEE digital library. The XRANK approach is not directly applicable to INEX, since the XML documents of INEX do not have cross references in the form of IDREFs or XLinks. The XSEarch approach is irrelevant to INEX, since all the nodes of any single XML document are deemed semantically related.

Our approach consists of a variety of Information Retrieval techniques augmented with the ability to give weights to different fragments of a document, based on the tags. Specifically, we use term frequencies, inverse document frequencies, proximity among occurrences of keywords, and similarity between keywords and words from the given document. Each technique has been implemented as a separate ranker and the final ranking is done by merging the results of the various rankers.

## 2. TOPIC SEMANTICS AND SYNTAX

The query language of a standard search engine is simply a list of keywords, optionally preceded by the + or − sign. In the context of XML, the query language can also contain information about the structure, in the form of path expressions that describe specific parts of a document where the keywords should appear.

In INEX'03 [9], a query is called a *topic* and comprises four parts: (1) *title:* this part describes the topic in a formal syntax, (2) *description:* a description in a natural language of the information that is needed, (3) *narrative:* a more detailed description, and (4) *keywords:* a set of comma-separated *terms*, where a term is a single keyword or a phrase encapsulated in double quotes. Our system uses only the title.

A topic can be either *content only* (abbr. CO) or *content and structure* (abbr. CAS). In a CO topic, the title contains only content-related conditions; it is a set of space-separated terms, optionally preceded by the + or the − sign. For example,

```
+database +''java programming''
```

*This research was supported by The Israel Science Foundation (Grant 96/01).

is a CO topic about "database" and "java programming."

In a CAS topic, the title relates terms to specific locations in documents. The general form of the title is
`CE[filter] CE[filter] ... CE[filter]`,
where each `CE` is a *context element* that specifies a path in the document (using XPath syntax). A `filter` is a Boolean combination of XPath predicates (e.g., a comparison between a path expression and a constant) and predicates of the form `about(path,string)`, where `path` is an XPath expression and `string` is a quoted string of terms (each term could be preceded by a + or −). For example,

```
//article[.//@yr > '2000']
  //sec[about(.,'+''java programming''')]
```

specifies that sections about "java programming" from articles written after 2000 should be retrieved.

We use $T$ to denote a CO or a CAS topic. By a slight abuse of notation, $T$ also denotes the list of all stemmed terms appearing in the title of $T$. Stop words are eliminated. $T_+$ denotes the list of terms in $T$ that are preceeded by a + sign, $T_-$ denotes the list of terms that are preceded by a − sign, and $T_o$ is the list of all the remaining (i.e., optional) terms in $T$.

# 3. AN OVERVIEW OF THE SYSTEM

The design of our system was influenced by two major considerations. First, our goal was to build an extensible system so that various information-retrieval techniques could be combined in different ways and new techniques could be easily added. Second, the system had to be developed in a very short time.

The first consideration led to the decision to implement each information-retrieval technique as a separate ranker and to implement a merger that would merge the results of the individual rankers.

The second consideration influenced the implementation of the topic (i.e., query) processor. In INEX, a topic may include expressions in XPath (augmented with the "about" function) that refer to the structure of the documents to be retrieved. Thus, an XPath processor is needed in order to evaluate a given topic. However, any existing XPath processor cannot be applied to the complete description of a topic that is written in the formal syntax of INEX; instead, it can only be applied separately to each XPath expression that is embedded inside the topic. This is not sufficient for an accurate processing of CAS (content and structure) topics, since when different XPath expressions from the same topic are evaluated separately, it is impossible to tell how to combine their results correctly. So, it seemed that the topic processor would require a complete implementation of an XPath parser (and that would be time consuming). Instead, we implemented (in Java) a parser for INEX topics that creates an XSL stylesheet (i.e., a program written in XSL). Since XPath is included in XSL, we circumvented the need to implement an XPath parser as a part of our topic processor.

Figure 3 depicts the main components of the system. The first step is building the indices, which are described in detail in Section 4. Given a topic, the indices are used to filter the whole corpus in order to retrieve the documents that contain all the required keywords (i.e., keywords preceded by +). Documents that pass through the filtering phase are processed by an XSL stylesheet that is generated from the topic. The XSL stylesheet retrieves from each document all the fragments that are relevant to the processing of the given topic. The retrieved fragments are produced as an XML file (one per document) in a manner that preserves the original hierarchy among these fragments. In the next step, each ranker processes all the XML files and creates a new XML file of the ranked results. In the final steps, the results of the various rankers are merged into a single XML file.

# 4. INDEXING

The system uses several indices when processing topics (i.e., queries). The creation of the indexes is done as a preprocessing step by the *indexer*. The indices are described below.

### *Document-Location Array*

The system assigns a unique *document identifier* (also called *did*) to each document. The *document-location array* is used to associate each *did* with the physical location, in the file system, of the corresponding document.

### *Inverted Keyword Index*

The *inverted keyword index* associates each keyword with the list of documents that contain it. Stop words, i.e., words that are used very frequently in English (e.g., "in," "to," "the," etc.) do not appear in the index. Also, regular stemming, using the Porter's stemmer [12], is done in order to achieve a higher flexibility when searching for a particular keyword. The inverted-keyword index stores stems of words. For each stem $w$, there is a *posting list* of the *did*'s of all the documents that contain some keyword with stem $w$.

### *Keyword-Distance Index*

The *keyword-distance index* stores information about proximity of keywords in the corpus. For each pair of keywords, the system computes a score and the keyword-distance index holds this score. The score reflects the number of occurrences of that pair of keywords in any single sentence. It also reflect the distance between the two keywords when they appear in the same sentence. The score for a given pair of keywords is the sum of the inverse of the distance between the two keywords over all the sentences in all the documents of the corpus. Formally, the score of the pair $(w_i, w_j)$ is

$$D(w_i, w_j) = \sum_{d \in \mathcal{D}} \sum_{s \in d} \sum_{(w_i, w_j) \in s} \frac{1}{distance(w_i, w_j)}$$

where $\mathcal{D}$ is the set of all the documents in the corpus, $d$ is a document, $s$ is a sentence, and $distance(w_i, w_j)$ is the number of words separating $w_i$ and $w_j$. Scores are normalized and $D(w, w)$ is defined to be 1 (the maximum). The keyword-distance index actually stores the scores for pairs of stems rather than complete keywords.

### *Tag Index*

Tags are given weights according to their importance. The weight of each tag is a parameter that can be easily modified

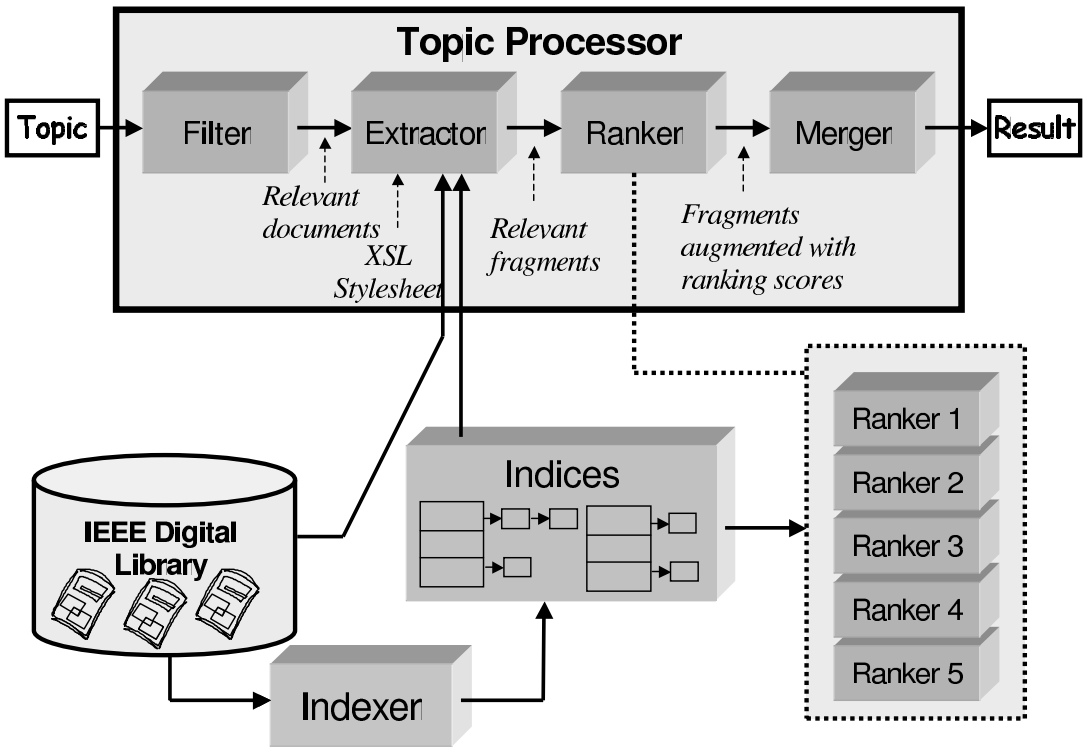


**Figure 1: Overall Architecture**

by anyone who uses the system. A Java property file stores the weight $tw(t)$ of each tag $t$.

### *Inverse-Document-Frequency (idf) Index*

The *document frequency* of a keyword $k$ is the number of documents that contain $k$, divided by the total number of documents in the corpus. The *inverse document frequency* is defined as follows:

$$idf(k) := \log\left(1 + \frac{|\mathcal{D}|}{|\{d \mid d \in \mathcal{D} \text{ and } k \in d\}|}\right)$$

where $\mathcal{D}$ is the set of all the documents in the corpus. The *inverse-document-frequency index* is a hash table that holds the inverse document frequency for the stem $k$ of each keyword.

## 5. TOPIC PROCESSING

The processing of a topic $T$ is done in four phases. In the *filtering phase,* the documents that contain all the keywords in $T_+$ are retrieved from the corpus. In the *extraction phase,* the relevant fragments are extracted from each document. In the *ranking phase,* the fragments from the previous phase are ranked by each ranker. In the *merging phase,* the results of the various rankers are merged together. Next, we describe each phase in detail.

In the filtering phase, for each keyword $k \in T_+$, the posting list $L_k$ of the stem of $k$ is extracted from the inverted keyword index. The intersection $L_T = \cap_{k \in T_+} L_k$ is computed and the result is an XML document that contains a list of all the *did*'s of the documents in $L_T$.

In the extracting phase, an XSL stylesheet is generated from the title of the topic. This stylesheet extracts the relevant fragments from each document that passed the filtering phase. For CAS topics, the relevant fragments are determined by the title. For CO topics, the system has to determine which fragments are relevant. In our system, the fragments that could be returned are determined in advance; this policy was proposed by [3] and is also used in XRANK [7]. Specifically, these fragments are either the whole document, the front matter, the abstract, any section or any subsection.

A potentially relevant fragment must also satisfy some conditions. First, it must include all the terms that are preceded by +. Moreover, it may have to satisfy some predicates, e.g., `.//@yr > '2000'`. Thus, extracting the relevant fragments requires a processor that is capable of parsing titles of CO and CAS topics. An XPath processor is not suitable for the job, since the syntax of titles is more general than that of XPath. In our system, a Java program parses the title and generates an XSL stylesheet that does the extraction. Since XPath is included in XSL, portions of the title that adhere to the XPath syntax can be transplanted into the stylesheet. This lead to a fast implementation of the topic processor.

In the title of a CAS topic, there is a *core path expression* that consists of the concatenation of all the context elements.

There are also several *filter path expressions*, where each one is a path expression that appears in some filter concatenated with all the context elements that precede it. The last element of the core path expression and the last element on any filter path expression are called *target elements*. For example, consider the following CAS title:

```
//a[about(.//b, '...')]//c[about(.//d, '...')]
```

The core path expression is `//a//c`. The fragments that eventually will be returned as answers for this title are `c` elements. The filter path expressions are `//a//b` and `//a//c//d`. Fragments that are `b` and `d` elements should be extracted in order to check the conditions that are specified in the `about` clauses.

Extracting fragments for the path expressions and checking that each one satisfies its corresponding condition is not quite enough. In order for the rankers to work correctly, it is important to know whether a fragment extracted for a `b` element is related to a fragment extracted for a `d` element, in the sense that both have the same `a` element as an ancestor. Therefore, the XSL stylesheet extracts fragments in a manner that preserves the original hierarchy among these fragments. Essentially, the stylesheet has a sequence of nested loops. The nesting of the loops follows the hierarchy dictated by the the core and filter path expressions. Each loop extracts all the fragments for its corresponding element. Each extracted fragment is assigned a level number, which is the level of nesting of its corresponding loop. For example, in the above title, the extracted `d` elements are descendants of the extracted `c` elements and, hence, will have a larger level number. The XSL stylesheet is applied to all the documents that passed the filtering phase and it produces a new XML document, $D_T$, that contains for each extracted fragment *(1)* the URL of the parent document, *(2)* the path of the fragment in the document, *(3)* the stems of keywords from the title that appear in the fragment, *and (4)* the fragment itself. The XML file $D_T$ is given to each ranker. Note that the `about` predicate can be evaluated by the rankers, since the relevant fragments are in $D_T$. The ranking of fragments and the final merging are explained in the next section.

# 6. RANKING THE RESULTS

We implemented five rankers, namely *word-number ranker*, *idf ranker*, *tf-idf ranker*, *proximity ranker* and *similarity ranker*. Each ranker gives scores to the fragments that are listed in the XML file $D_T$. This section describes the five rankers and how their results are merged.

## 6.1 Word-Number Ranker

Recall that $T_o$ is the list of optional terms (i.e., not preceded by the + or − sign) from the title of a given topic $T$. Similarly, $T_-$ is the list of terms that should not appear in the result (i.e., preceded by the − sign). Given a fragment $F$, the number of optional terms that appear in $F$ is $|T_o \cap F|$ and the number of unwanted terms in $F$ is $|T_- \cap F|$. The score given to $F$ by the word-number ranker is

$$|T_o \cap F| + 1 - \frac{\min(|T_- \cap F|, 10)}{10}.$$

Note that the score is increased when the number of optional terms appearing in the fragment $F$ is increased and it is decreased when the number of unwanted terms in $F$ is increased. Also note that the weight that is given to the appearance of a wanted term is an order of magnitude greater than the weight given to the appearance of an unwanted term. Moreover, there is a bound of 10 on the total number of unwanted terms that are taken into account.

## 6.2 Inverse-Document-Frequency (IDF) Ranker

We first give the intuition behind the idf ranker. Consider two fragments $F_1$ and $F_2$, such that $F_1 \cap T_+$ and $F_2 \cap T_+$ contain single keywords, $w_1$ and $w_2$, respectively. Also, assume that the intersection of $F_1$ and $F_2$ with $T_-$ is empty. In this case, the word-number ranker returns the same score for $F_1$ and $F_2$. If, however, $w_1$ is a frequent word in the corpus and $w_2$ is a rare one, then $F_2$ should be given a higher score than $F_1$.

Let $F$ be a given fragment. In the idf ranker, a rare keyword that appears in $F$ has a greater effect on the score than a keyword that appears frequently in the corpus. The score of the idf ranker is the following sum of the idf values of the optional words and the unwanted words that appear in $F$.

$$\sum_{k \in \{T_o \cap F\}} idf(k) - \sum_{k \in \{T_- \cap F\}} idf(k)$$

Note that terms of $T_+$ are not considered by this ranker, since all the fragments contain them.

## 6.3 Tf-Idf Ranker

The tf-idf ranker uses a model similar to the vector-space model that is common in information retrieval [2]. We have modified the basic technique so that the weights given to tags will be incorporated in the computation of the ranker's score.

Let $T$ be a given topic and let $F$ be a fragment. We assume that all the words in $T$ and in $F$ are stemmed. We also assume that all the stop words are removed from $F$ and $T$. The score given by the ranker to $F$ with respect to $T$ is computed using a variation of the standard *tfidf* (term frequency, inverse document frequency) method. Next, we briefly describe *tfidf* and how it is computed in our system.

Let $k$ be a term. The *term frequency (tf)* of $k$ in $F$ is the number of occurrences of $k$ in $F$ (denoted as $occ(k, F)$), divided by the maximal number of occurrences in $F$ of any term. That is,

$$tf(k, F) = \frac{occ(k, F)}{\max\{occ(k', F) \mid k' \in F\}}.$$

Note that a term is likely to have a larger term frequency in a small document than in a bigger one.

The inverse document frequency of $k$, $idf(k)$, was defined in Section 4. The *tfidf* of a term $k$ w.r.t. a fragment $F$, denoted by $tfidf(k, F)$, is $tf(k, F) \times idf(k)$. Note that by taking a log in the *idf* factor, the overall importance of the *tf* factor in *tfidf* is increased.

In our system, each tag has a weight. The default weight is 1. A user can modify the weight of any tag. The *accumulated weight* of a word $w$ in an XML file $X$ is the multiplication of the weights of all the tags of the elements of $X$ in which $w$ is nested. That is, the accumulated weight of $w$ is produced by multiplying all the weights of the tags of elements on the path from the root of $X$ to $w$. The effect of the weight of tags on the computation of *tfidf* is as follows. For each occurrence of $k$ in $F$, instead of increasing $occ(k, F)$ by 1, the value of $occ(k, F)$ is increased by the accumulated weight of $k$ for that occurrence.

The value of $tfidf(k, F)$ is normalized as follows.

$$w(k, F) := \frac{tfidf(k, F)}{\sqrt{\sum_{k' \in F} tfidf(k', F)^2}}$$

By definition, $w(k, F)$ is 0 if $k$ does not appear in $F$. We denote by $K$ the set of all the keywords appearing in the corpus. Each fragment $F$ in the corpus is associated with a vector $V_F$ of size $|K|$. For each keyword $k$ of $K$, the vector $V_F$ has an entry $V_F[k]$ that holds $w(k, F)$.

For each topic $T$, we define $V_T$ to be the following vector.

$$V_T[k] = \begin{cases} 1 & \text{if } k \in T_+ \cup T_o \\ -1 & \text{if } k \in T_- \\ 0 & \text{otherwise} \end{cases}$$

The score given to the fragment $F$ by the ranker is the cosine between $V_T$ and $V_F$. The value of this cosine is proportional to the following sum:

$$\sum_{k \in K} V_F[k] \times V_T[k] = \sum_{k \in T} V_F[k] \times V_T[k]$$

Note that the above equality holds because $V_T[k] = 0$ if $k \notin T$.

## 6.4 Proximity Ranker

### *Lexical Affinities for Text Retrieval*

The idea behind the proximity ranker is to use *lexical affinities* (abbr. LA) of words. The ranker takes advantage of the correlation between words that appear in a single phrase in a certain proximity.

The notion of lexical affinities for text retrieval was first introduced by Saussure [13]. Later, it was developed by Maarek and Smadja [10], in the context of information retrieval.

Essentially, our ranker works as follows. Given a topic $T$ containing the terms $t_1, \ldots, t_n$, the ranker creates a list that contains all possible pairs of distinct words $(t_i, t_j)$, such that $t_i < t_j$ (words are compared lexicographically). For each fragment $F$, whenever the ranker finds in $F$ an occurrence of a pair $(t_i, t_j)$ in a single sentence, the score given to $F$ is increased. Different increasing policies can be used.

### *Lexical Affinities for XML Retrieval*

The following explains how LA retrieval is adapted to XML, in general, and to our system, in particular.

Two words that appear very far from each other should not be considered as a LA. A maximal distance must be defined, such that when exceeded, the two words are not considered to be correlated. Martin [11] showed that 98% of LA's relate words that are separated by at most five words within a single sentence. Maarek and Smadja [10] used this result by searching for co-occurrences in a sliding window (within a single sentence) of size 5. We have adapted this result to the context of XML as explained below.

In XML, structure and content are combined. Due to this lack of separation between structure and content, an XML file can have a logical unit of text in which the text does not appear in a sentence delimited by full stops, but rather delimited by tags. For example, consider the following XML fragment.

```
<author>John Washington</author>
<address>New Jersey State</address>
```

The absence of a full stop between Washington and New Jersey State could be mistakenly interpreted as a case where Washington State is a LA. In order to avoid such mistakes, we consider a closing tag followed by an opening tag as a delimiter of a logical unit.

When looking for lexical affinities in a topic (i.e., query), special attention must be paid to the structure of the topic in order to avoid an attempt to pair words that do not appear under the same tag. For words that are not under the same tag, a LA should not be created. For example, consider the following topic title.

```
//article//fm[
 (about(.//tig, '+software +architecture')
  or about(.//abs, '+software +architecture'))
  and about(., '-distributed -Web')]
```

In this topic, the pairs "software architecture" and "distributed Web" should be considered as LA's. The pairs "distributed architecture" and "software Web" should not be considered as LA's.

For words that appear under the same tag, but some of them in quotation marks, the LA's in quotation marks are given a larger weight. For example, consider the following topic.

```
/article[about(./fm/abs,
'"information retrieval" "digital libraries"')]
```

The pairs "information retrieval," "digital libraries," "information digital," "information libraries," "retrieval digital" and "digital libraries" are all considered as LA's. However, the occurrences of "information retrieval" or "digital libraries" in a fragment get a larger weight than the occurrences of "retrieval digital" or "digital libraries."

## 6.5 Similarity Ranker

The idea behind the similarity ranker is that if two words appears very frequently in proximity in the corpus, then they should be considered as related concepts. For example, if we find that "SQL" and "databases" are two words that frequently appear together, then we may conclude that

the two words are closely related. Therefore, when looking for documents about databases, we may as well search for documents about SQL.

Let $F$ be a fragment of a document $D$. $F_T$ denotes the terms appearing either in $F$, in the title of $D$ or in the abstract of $D$. As usual, $T$ denotes the terms in the title of a given topic. The similarity ranker computes the score of $F$ w.r.t. the given topic $T$ according to the formula

$$\prod_{k \in T} \sum_{w \in F_T} (tw(tag) * D(k, w))$$

where $tag$ is the tag with the largest weight among those containing $w$. The similarity ranker uses the keyword-distance index in order to get the value of the distance $D(k, w)$. The tag index is used in order to get the value of $tw(tag)$.

This ranker can be seen as an automatic query refinement. It differs from the work of Jing and Croft [8], since we do not use a probabilistic approach. It also differs from the work of Carmel et al. [5], since our refinement uses a global analysis of the *whole* corpus and assigns weights to all the co-occurrences in the fragment, rather than just to a limited number of LA's.

### 6.6 Merging the Results of the Rankers

Each fragment is given a score by each ranker. The overall score of a result according to a ranker is the sum of the scores given by the ranker to the different fragments composing the result.

A crucial issue is to determine the relative weight of each ranker in the final phase of merging the results of the various ranker. Tackling this issue requires extensive experimentation with the system. So far, only a rudimentary merger has been implemented and it is based on the simple idea of merging the results by lexicographically sorting the scores of the five rankers. The relative positions of the five rankers in the lexicographic sort is given in a configuration file and can be easily modified by the user through a browser.

We have experimented with different orders of the rankers; in all of them, the word-number ranker was first and idf ranker was second. Results were produced for the following three orders of the rankers:

- Word Number, Idf, Proximity, Similarity, Tf-Idf.
- Word Number, Idf, Similarity, Proximity, Tf-Idf.
- Word Number, Idf, Tf-Idf, Proximity, Similarity.

We always chose word number and idf to be the first and second rankers, since early experiments with the system indicated that it gave the best results. The proximity ranker, the similarity ranker and the tf-idf ranker were essentially used to tune the ranking of the first two rankers.

The following two restrictions were applied to the creation of the XML file that contains the final ranking of the fragments. First, the final result is limited to 1500 fragments. Secondly, at most 5 fragments from any single document could appear in the final result. These limitations could be easily modified by the user.

## 7. CONCLUSION AND FUTURE WORK

The main contribution of our work is a design of an extensible system that is capable of combining different types of rankers in a manner that takes into account both the structure and the content of the documents. Traditional as well as new information-retrieval techniques can be incorporated into our system, and the ranking score of each technique can be easily modified to include the weights assigned to tags. Our system is also extensible in the sense that it can be easily adapted to changes in the formal syntax of titles, due to the implementation of the topic processor by means of XSL.

Two major issues remain for future work. One is improving the efficiency of the system. The second is improving the quality (i.e., recall and precision) of the results. This requires extensive experimentation with the current rankers as well as with new ones. In particular, we plan to modify the merger so that it will use a single formula to aggregate the scores of the various rankers, rather than sorting the scores lexicographically. Towards this end, further experimentation is needed in order to find the optimal weight of each ranker relative to the other rankers.

## 8. REFERENCES


[1] `http://www.google.com`.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.

[3] G. Bahlotia. Keyword searching and browsing in databases using banks. In *ICDE Conference*, 2002.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[5] D. Carmel, E. Farchi, Y. Petrushka, and A. Soffer. 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, 283-290*, Tampere, Finland, June 2002.

[6] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic search engine for XML. In *Proc. 2003 VLDB International Conference on Very Large Databases*, Berlin (Germany), 2003.

[7] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *Proc. 2003 ACM SIGMOD International Conference on Management of Data*, San Diego (California), June 2003.

[8] Y. Jing and W. Croft. An association thesaurus for information retrieval. In *Proc. of RIAO-94, 4th International Conference "Recherche d'Information assistee par Ordinateur", 146-160*, New York (NY), 1994.

[9] G. Kazai, M. Lalmas, and S. Malik. Inex'03 guidelines for topic development, May 2003.

[10] Y. Maarek and F. Smadja. Full text indexing based on lexical relations: An application: Software libraries. In *Proceedings of the 12th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 189-206*, Cambridge, MA, June 1989.

[11] W. Martin, B. Al, and P. van Sterkenburg. On the processing of a text corpus: from textual data to lexicographical information. In *Lexicography: Principles and Practice, Ed. R.R.K Hartman, Applied Language Studies Series, Academic Press*, London, 1983.

[12] M. Porter. An algorithm for suffix stripping. In *Program*, chapter 14(3), pages 130–137. 1980.

[13] F. D. Saussure. *Cours de Linguistique generale, Quatrieme edition.* Librairie Payot, Paris, France, 1949.