The SearX-Engine at INEX'03: XML enabled probabilistic retrieval

Holger Flörke
doctronic GmbH & Co. KG
Adenauerallee 45-49
D-53332 Bornheim, Germany
floerke@doctronic.de

ABSTRACT

In this paper we describe how we used our „out of the box" search engine for INEX'03. The *SearX-Engine* integrates structural information into the query language and the retrieval function. It is based on the widely used probabilistic retrieval (TF*IDF) and uses additional indexes on document structure to evaluate queries.

1. INTRODUCTION

Due to the fact that in the real world a huge amount of structured full-text documents is available, there is a growing need to search within those documents. One simple way is to throw away all structural information and use a well known retrieval method to search an unstructured document collection. But if you deal with finding all relevant documents to a user-query, you will be quite happy about every small piece of information you can use to fulfil the user's information need.

Using the structural information within the documents and maybe the query can help to retrieve more relevant and fewer irrelevant documents. Therefore retrieval systems will do a better job if they take the structure into account. New retrieval algorithms for structured documents have to be designed and implemented.

The main goal of INEX (Initiative for the Evaluation of XML Retrieval) [1] is to promote the evaluation of content-oriented XML retrieval by providing a large test collection of XML documents, uniform scoring procedures, and a forum for organizations to compare their results.

The *SearX-Engine* [2] is a commercial product developed by Doctronic for searching within collections of XML documents. The author of this article is the chief of research and development at Doctronic and is responsible for the main concepts of the *SearX-Engine*. The *SearX-Engine* is integrated into *Xaver*, a multi-channel publishing system for large and structured text collections [3]. *Xaver* is mainly used by professional publishers in the field of law, taxes, or technical documentation.

2. QUERY LANGUAGE

A query in the probabilistic retrieval model can be represented by an unordered set of terms. The user can easily express his information need by specifying some related terms.

To integrate structural assignments and weightings into the query language, we introduce the concept of *roles*. One role (such as 'author' or 'heading') combines all parts of the collection with a common semantics, so the user does not have to know about the specific structure of the underlying collection.

The mapping from the collection data to structural roles is done at indexing time by the content provider. In our scenario the content provider should be seen as the person who prepares the collection for publishing and retrieval. In the majority of cases this is not the author, but the publisher or a technical service provider. The content provider should know about the content of the collection and the potential end user. Therefore he is qualified to define the roles and any other search parameter.

Not only the user level is simplified by the concept of roles. Roles can also help to search across heterogeneous collections where each one provides its own mapping from collection-specific data structures to general roles. At the implementation level roles can help to keep the index structures and algorithms small and handy, because the structural complexity is reduced.

The user can integrate structural information into his query by assigning query terms to structural roles related to his information need. He can also choose one retrieval role, which determines the parts of the collection that should be returned and ranked.

The *SearX-Engine* also supports a mechanism to weight roles. If a query term is found, the score of this occurrence will be influenced by the structural context. This weighting is often made by the publisher, who can provide his knowledge about the data and the assumed information needs of the users.

The application knows the concept of headings, so structural implications (eg scoring an article title should score all sections within this article) can be expressed. Furthermore the term operators '+' (must have) and '–' (must not have) and phrases are supported.

3. RETRIEVAL FUNCTION

The *SearX-Engine* is based on the well known probabilistic retrieval [4]. Within this framework the score of a document consists of two parts. First the inverted document frequency measure represents the entropy of the term occurring in both document and query. The more documents exist containing the term, the smaller the IDF gets. The second part reflects the term frequency, which has to be normalized by the length of the document. Those two parts of the score can be weighted for different collection characteristics by the tuning factors *C* and *K*.

$$\rho(q,d) = \sum_{t_i \in q \cap d} (C + \mathrm{IDF}_i) \cdot \left(K + (1-K) \cdot \frac{f_{id}}{\mathrm{maxfreq}_d} \right)$$

$$\mathrm{IDF}_i = \log \frac{N - n_i}{n_i}$$

f_{id} = frequency of term t_i in d

$\mathrm{maxfreq}_d$ = maximum frequency of any term in d

Within collections of structured documents, the retrieval function should not necessarily score an entire document. It should also be able to score smaller (or larger) elements. A structured query contains terms related to roles, and roles can be weighted by the user and the publisher. Therefore we have to extend the retrieval function in a number of ways.

The IDF is replaced by the *inverted element frequency* (IEF) depending not only on a term, but also on a role.

$$\mathrm{IEF}(t_i, s_k) = \log \frac{N_{s_k} - n_{t_i,s_k}}{n_{t_i,s_k}}$$

$$N_{s_k} = \text{number of elements having role } s_k$$

$$n_{t_i,s_k} = \text{number of elements having role } s_k \text{ and term } t_i$$

To adapt the IDF, all elements having the same role are handled as a document collection and the entropy over this collection is measured.

The term frequency has to be calculated with respect to the structural conditions in the query and the structural weighting has to be considered.

$$f(t_i, e, s_k) = \sum_{\substack{e' \text{ is descendant-or-self of } e \\ s_k \in \mathrm{roles}(e')}} \mathrm{freq}(t_i, e') \cdot \max(w(s) \mid s \in \mathrm{roles}(e'))$$

Putting the pieces together, the new retrieval function (ignoring the tuning factors *C* and *K*) is:

$$\rho(q, e) = \sum_{\substack{(t_i,s_k) \in q \\ t_i \in e}} \mathrm{IEF}(t_i, s_k) \cdot \frac{f(t_i, e, s_k)}{\mathrm{maxfreq}_e}$$

This formula is able to estimate the relevance of every element to a query with structural assignments and structural weightings.

4. INDEX STRUCTURES

The index structures to evaluate the retrieval function of a query on the collection are quite similar to the well known inverted files [5] used for information retrieval on unstructured texts. There is a lexicon populated with all the terms of the collections, their IEF for each role, and a pointer to the list of postings. The list of postings contains each occurrence for each term. Other than the usual inverted files we have to record the structural context of each occurrence. Because we need to know the complete path of roles from the occurrence to the root of the document structure, integrating the structural information within the list of postings would introduce a huge overhead. Therefore we decided to use a special index for the structure and store links to this index into the list of postings. The structure index is a tree like index structure. It represents the document structure and contains the set of roles for each indexed element and the frequency of the most frequent term of each element. This design reduces the storage requirements, but introduces some runtime disadvantages.

5. INEX'03

To evaluate INEX'03 topics, we made a mapping of the used structural assignments to roles and transformed the topics to our query format described above. Weighting was done to push up hits within article titles, abstracts and keywords. The values of this weighting were guessed by the author and were not calculated from INEX'02 or other collections.

5.1 CO-Topics

For Content-Only queries we decided to rank always whole articles and search for the title and the keywords of the topic description within the whole article. So we did not make any structural assignments besides the weighting. The role *article* is mapped to all articles in the collection.

```
<inex_topic ct_no="35" query_type="CO" topic_id="100">
  <title>+association +mining +rule +medical</title>
  <description>
    Retrieve information about association rule mining in medical databases </description>
  <narrative> We have a medical data mining ... </narrative>
  <keywords>association, mining, rule, medical</keywords>
</inex_topic>
```

↓

```
<query name="INEX">
  <retrieval-role><constant value="article"/></retrieval-role>
  <filter/>
  <query-item>
    <role><constant value="article"/></role>
    <terms>
      <constant value="+association +mining +rule +medical association mining rule medical"/>
    </terms>
  </query-item>
</query>
```

5.2 CAS-Topics

The CAS-Mapping is somewhat more difficult because of the CAS topic format introduced in INEX'03.

The last element in the path of the title is taken as the retrieval element. Every about-predicate creates one query item (pair of role and terms). All the paths within the CAS topics are used to build the set of roles needed by the *SearX-Engine*. The XPath for the role mapping is taken as the name.

```
<inex_topic ct_no="14" query_type="CAS" topic_id="63">
  <title>
  //article[about(.,'"digital library"') AND about(.//p,'+authorization +"access control" +security')]
  </title>
  [...]
</inex_topic>
```

↓

```
<query name="INEX">
  <retrieval-role><constant value="article"/></retrieval-role>
  <filter/>
  <query-item>
    <role><constant value="article"/></role>
    <terms><constant value=" 'digital library' "/></terms>
  </query-item>
  <query-item>
    <role><constant value="article//p"/></role>
    <terms>
      <constant value="+authorization +'access control' +security"/>
    </terms>
  </query-item>
</query>
```

A Filter on an explicit attribute value in the CAS title (eg /article[.//yr <= '2000']) was translated into a *SearX-Engine* filter to exclude document parts based on attribute values. Therefore we are able to map every CAS query to our query format.

6. EVALUATION

We have submitted two CO and two VCAS runs. Within each track there was one run created automatically and one created manually from the INEX topic description. The results of the CO submissions are shown in figure 1 (automatic) and 2 (manual). Because at this time there are no metrics for the VCAS available,

we have treated the VCAS submissions like SCAS. The results are given in figure 3 (automatic) and 4 (manual).

The size of the indexed data was 876MB, that's 111% of the original size of the collection. This size includes all the data, the search indexes, and a phrase index for the efficient evaluation of phrase searches. Each CO topic was evaluated in 20s, each CAS topic in 26s on average (RedHat Linux 8.0 on a P4/2.4GHz). As opposed to an end user query, the given timings contain the construction of the whole result set and the transformation to the INEX submission format.

7. CONCLUSION

We were able to index the INEX document collection and evaluate the INEX topics without any modification of the "out of the box" *SearX-Engine*. The index structures needed by the engine are quite small and each topic could be evaluated fast. The results of the submitted runs show a good retrieval quality. We are satisfied with the overall performance of the *SearX-Engine*.

The INEX'03 was a good exercise to show the flexibility of our *SearX-Engine*. The results of the experiments may influence the future development to improve performance and index sizes on the one hand and the retrieval quality on the other. Without INEX, retrieval quality improvements would be much harder.

8. REFERENCES

[1] Initiative for the Evaluation of XML Retrieval. `http://inex.is.informatik.uni-duisburg.de:2003`

[2] doctronic SearX-Engine `http://www.doctronic.de/produkte/searx.html`

[3] doctronic Xaver-Publishing. `http://www.doctronic.de/produkte/xaver.html`

[4] N. J. Belkin and B.W. Croft. Retrieval Techniques. *Annual Review of Information Science and Technology*, 22:109-145, 1987.

[5] William B. Frakes, and Recardo Baeza-Yates. Information Retrieval – Data Structures and Algorithms. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

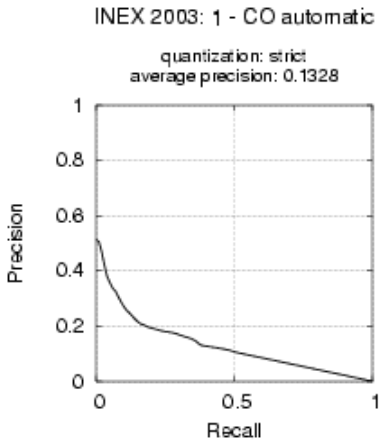

Figure 1. Evaluation of the automatically generated CO runs with `inex_eval_ng` (overlapping considered).

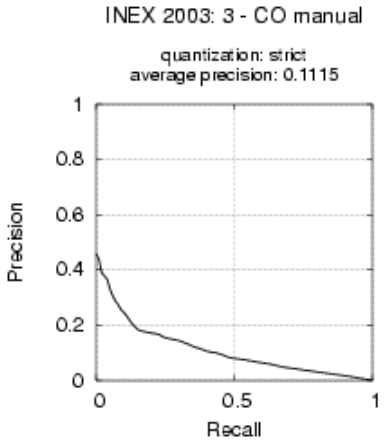

Figure 2. Evaluation of the manually generated CO runs with `inex_eval_ng` (overlapping considered).

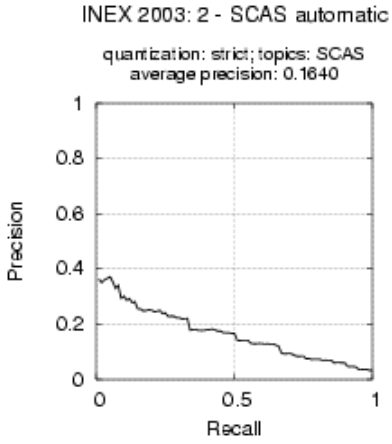

Figure 3. Evaluation of the automatically generated SCAS runs with `inex_eval`.

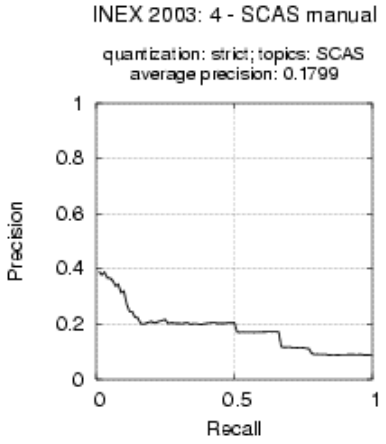

Figure 4. Evaluation of the manually generated SCAS runs with `inex_eval`.