UAIC: Participation in TEL@CLEF task

Adrian Iftene, Alina-Elena Mihăilă, Ingride-Paula Epure

UAIC: Faculty of Computer Science, “Alexandru Ioan Cuza” University, Romania
{adiftene, elena.mihaila, paula.epure}@info.uaic.ro

Abstract. In 2009 was first time when we built a system in order to participate in the TEL@CLEF competition. In this competition the aim is to build retrieval algorithms on multilingual collections of catalog records from TEL collections. Our system has four main components: the pre-processing and indexing component, the component responsible with applying rules, the translation component and the searching component. This paper presents how we implement these components and how interacts them in order to achieve the desired purpose.

Categories and Subject Descriptors

H.3 [Information Storage and Retrieval]: H.3.1 Content Analysis and Indexing; H.3.3 Information Search and Retrieval; H.3.4 [Systems and Software]: Performance evaluation.

General Terms

Experimentation, Performance, Measurement, Algorithms.

1 Introduction

The aim of the 2009 Ad Hoc[1] track was to improve the last year's experience, with the same three tasks: Tel@CLEF, Persian@CLEF, and Robust-WSD. The general aim of the task was to create good reusable test collections for each of them. The main task offers monolingual and cross-language search on library catalog records in English, French, and German, organized in collaboration with The European Library (TEL[2]) (Agirre et al., 2008).

In 2009, TEL@CLEF evaluated retrieval algorithms on multilingual collections of catalog records. As in 2008, the collections were derived from the English, French and German archives of The European Library. The task is to search and retrieve relevant items from collections of library catalog cards.

Data from this year was very different from the news corpora previously used in the CLEF ad hoc track, consisting of bibliographic data (document surrogates). Whereas in the traditional ad hoc task, the user searches for document containing

[1] Ad-Hoc task: http://www.clef-campaign.org/2009/2009Ad-hoc-tasks.html
[2] TEL: http://www.theeuropeanlibrary.org/

information of interest, here the user will be searching to identify which publications are of potential interest – according to the information provided by the catalog card. The question the user is asking is “Is the publication described by the bibliographic record relevant to my information need?”.

Three target collections were provided: TEL Catalog records in English (Copyright British Library (BL)), TEL Catalog records in French (Copyright Bibliothèque nationale de France (BnF)), and TEL Catalog records in German (Copyright Austrian National Library (ONB)). All three collections are to some extent multilingual and contain documents (catalog records) in many additional languages.

The way in which we built the system for TEL track and it components are presented in Section 2, while Section 3 presents the runs submitted details. Last Section presents conclusions regarding our participation in TEL 2009 track.

2 UAIC System for TEL@CLEF

Our system performs the following operations: pre-processing, indexing, applying rules, translation and searching. The Figure 1 presents the system architecture.

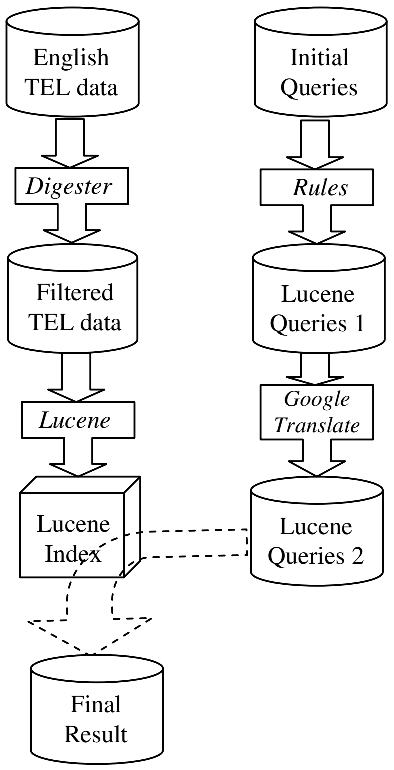

Figure 1: UAIC system used in TEL@CLEF 2009

Details about the main system components are presented below.

2.1 Pre-processing and Indexing

Pre-processing step help us in selection of relevant tags from XML files. We used Digester[3] for selecting just the attributes we are interested in (title and subject). We have an xml configure file named "playRules.xml" for Digester, where we put the rules that Digester must respect when we parse the xml files (from the library).

At this step we use Lucene[4] (Hatcher and Gospodnetic, 2005). We used Lucene in order to index the XML files filtered with Digester and after that to search in this index the relevant documents.

First we created an object of *Document type* in Lucene terminology, which will be assigned to each article that we want to index. We considered the fields: “docno”, “title” and “subject” and we filled them with the corresponding values for the article that we want to index. Pairs (field, value) represents the terms of the current document. The only condition is that the name has to be a String. Add method for the document will take a Field object type that we build using one of static methods in class Field. In the end, we need to consider an IndexWriter for indexing of the Document.

The Document class has a *get()* method that can be used to extract the information that was stored in the index. For example, to get the author from the Document we would code *doc.get("author")*.

Since we added the article itself as *Field.UnStored*, attempting to get it will return null. However, since we added the URL of the article to the index, we can get the URL and display it to the user in our result list.

2.2 Applying Rules

At this step, in original query in English we identify for every word the lemma and after that accordingly with the result, we build a new query using the following rules:

- If there are named entities in the query, these entities will be included in title field and also in the subject field, having, of course, a greater relevance in comparison with other words (we used boost factor 2, instead of default boost factor which is 1);
- Else every element of the query is added in the title field, as well as in the subject field;
- Also, any element of the query (non-named entity) is searched in the Lemma file and thus obtaining the lemmas corresponding to the word. These lemmas will be included in the Lucene query having a smaller relevance (boost factor 0.75) than the original word.

In the end, a complex Lucene query is obtained, which is to be sent to the next module (Translation), so that the Lucene query can be translated.

[3] Digester: http://commons.apache.org/digester/
[4] Lucene: http://lucene.apache.org/

For example the query will look like:

```
<query>
  <identifier>10.2452/701-AH</identifier>
  <lang>en</lang>
  <text>subject:document^0.7 subject:documenting^0.7
((subject:documents subject:species)^2.0)
(subject:fauna^2.0 subject:about^2.0 subject:arctic^2.0)
(title:arctic title:animals)</text>
</query>
```

And the result will be:

```
  10.2452/701-AHQ0 010786904 0 0.15470393 runDefault1
  10.2452/701-AHQ0 010786905 1 0.15470393 runDefault1
  10.2452/701-AHQ0 010786906 2 0.15470393 runDefault1
  10.2452/701-AHQ0 011249616 3 0.15470393 runDefault1
  10.2452/701-AHQ0 011249617 4 0.15470393 runDefault1
  ......................................................................
```

2.3 Translation

We used the Google Java Language API[5] to perform the translation of Lucene query from English to French and German.

The Lucene queries were received in an XML document structured as: the root element we have tag *query*, and the children are *id*, *text* (that was the query itself) and the *language* (en in our case). Every query had an *id* assigned which helped us keep the track of the way the translation was performed and the final result was generated in the result XML.

We created a DOM parser to obtain every Lucene query, then we processed it using *java.util.regex* API for pattern matching with regular expressions, so we could separate the words that needed translation from the ones that don't.

Finally, after the translation was performed we merged the three queries: the original one, the one in French and the one in German, and created the output file that had basically the same structure. These queries were put in the *result.xml* file and after that they are sending to the index module.

An example of translate query looks like:

```
<query>
  <identifier>10.2452/701-AH</identifier>
  <text>
  subject:document^0.7 subject:documenting^0.7
   ((subject:documents subject:species)^2.0) (subject:fauna^2.0
   subject:about^2.0 subject:arctic^2.0)
  (title:arctic title:animals) subject:document^0.7
   subject:documentation^0.7 ((subject:documents
```

[5] Google Java Language API: http://code.google.com/p/google-api-translate-java/

```
   subject:espÃ¨ces)^2.0) (subject:faune^2.0 subject:Ã  propos
   de^2.0 subject:Arctique^2.0) (title:Arctique title:animaux)
   subject:Dokument^0.7 subject:Dokumenting^0.7
  ((subject:Dokuments subject:Arten)^2.0) (subject:Fauna^2.0
   subject:Ã¼ber^2.0 subject:Arktis^2.0) (title:Arktis
   title:Tiere)
  </text>
</query>
```

2.4 Searching

At this part we used again Lucene and we created an *IndexSearcher* in order to extract relevant information from previous created Lucene index.
For searching we considered relevant the following values:

- Measure of documents relevance to a query,
- Factors:
  - **tf**: factor of term frequency in document,
  - **idf**: factor of documents with term in index,
  - **boost**: field-level boost,
  - **coord**: factor-based # of query terms in document,
  - **queryNorm**: normalization for query weights.

For example when we are looking for "*Arctic Animals*" with description "*Find documents about arctic fauna species*" the query will look like:

```
title:arctic animals AND "arctic fauna species"
```

with meaning: search for *arctic animals* in field *title* from Lucene index and search for *arctic fauna species* in field *subject* which is default searching field in Lucene index.

As we can see the search will search simultaneously in two fields of the index created: *title* and *subject*.

3 Submitted Runs

We submitted three runs with the following characteristics:

Run 1:
subject has boost 2 and the title has boost 1(default)
[entities have boost 2 and lemmas have boost 0.7]
[subject and title were given in l languages English, French and German]

Run 2:
subject has boost default and the title has boost 2
[entities have boost 2 and lemmas have boost 0.7]

[subject and title were given in l languages English, French and German]

Run 3:

subject and title have the same boost
[entities have boost 2 and lemmas have boost 0.7]
[subject and title were given in l languages English, French and German]

The best result was obtained for Run 1. Statistics for this run are presented in below statistics.

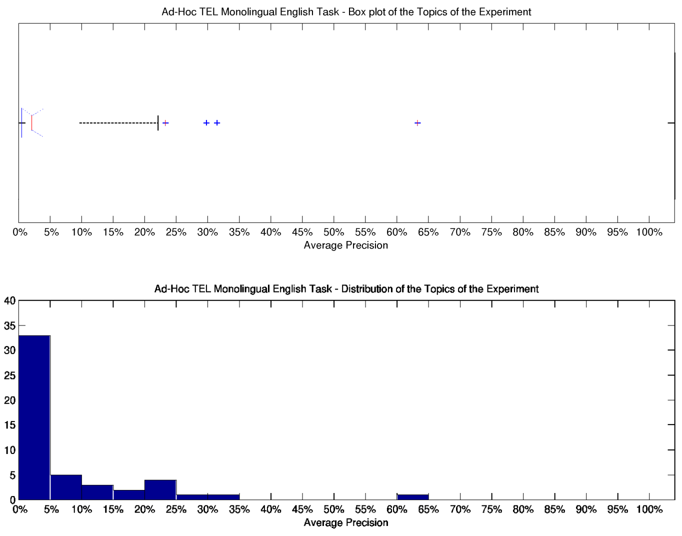

Figure 2: Average Precision Histogram for UAIC Run 1

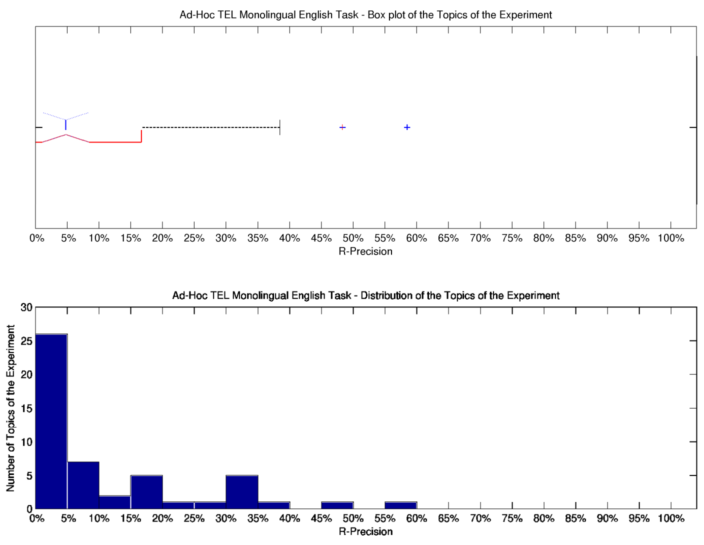

Figure 3: R Precision Histogram for UAIC Run 1

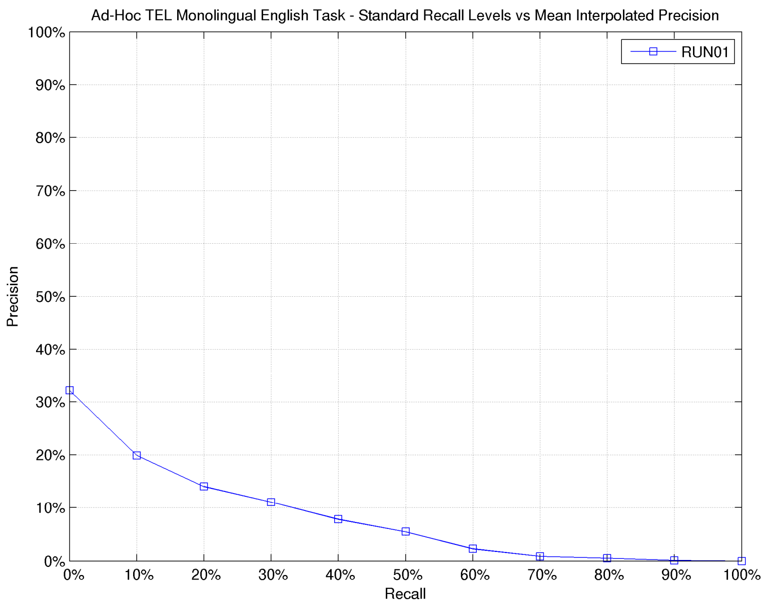

Figure 4: Standard Recall Levels vs Mean Interpolated Precision for UAIC Run 1

4 Conclusions

The presented system has four components: (1) pre-processing and indexing, (2) applying rules, (3) translation and (4) searching. First component after selection of relevant tags with Digester, uses Lucene libraries in order to create a Lucene index. Second component transforms the initial query from natural language to Lucene query and it adds to Lucene query boost factors. Third component translates the Lucene query from English to French and German and obtained a more complex Lucene query with elements in all three languages. The last component searches in Lucene index using Lucene queries and offered like output the system result.

From three runs submitted the first run was the better.

Acknowledgements

The authors would like to thank to the students Ionuț-Alexandru Bujdei, George-Leonard Chetreanu, Alexandra Apopoaiei and their colleagues from group 2B, second year, for their help and support at different stages of system development.

References

1. Agirre, E., Di Nunzio, G. M., Ferro, N., Mandl, T. and Peters, C.: CLEF 2008: Ad Hoc Track Overview. *In Proceedings of the CLEF 2008 Workshop*. 17-19 September. Aarhus, Denmark. (2008)
2. Hatcher, E. and Gospodnetic, O.: Lucene in action. *Manning Publications Co*. (2005)