The University of Amsterdam at INEX–2002

Maarten Marx, Jaap Kamps, Maarten de Rijke
Language and Inference Technology
ILLC, Universiteit van Amsterdam
Nieuwe Achtergracht 166, 1018 WV Amsterdam
The Netherlands
{marx,kamps,mdr}@science.uva.nl
http://www.science.uva.nl/LIT/

ABSTRACT

This document describes the runs for the INEX–2002 task submitted by the Language and Inference Technology Group at the University of Amsterdam. Besides a description of our experiments some logical problems with the INEX format of the content and structure topics are discussed and an alternative is proposed.

1. INTRODUCTION

The aim of our official runs was to experiment with the effectiveness of different types of morphological normalization for structured corpora. Morphological normalization proved successful for plain text collections [5, 6]. The XML retrieval task departs from the strict Boolean query matching used in traditional database theory, allowing for various gradations of relevance. In particular, related words like morphological variants should share some of their relevance. In order to study the precise effect of morphological normalization, we created plain-word, stemmed, and ngrammed indexes that preserve the XML-structure of the original documents. This allows for both the content-only and content-and-structure topics to be evaluated against all three indexes.

All experiments were carried out with the FlexIR system developed at the University of Amsterdam [5], using the Lnu.ltc weighting scheme. Our indices follow the classical IR model: the documents (in this case the articles in the collection) are the atomic units.

For the content only topics, the XML structure of the documents was not used. For the content and structure topics, we used a two step strategy. We first treated the topic as a content only topic and selected the 1000 highest ranking documents. Then we directly processed (a morphological normalization of) these documents.

Our experiments are described in more detail in Sections 2 and 3. In the remaining sections we discuss encountered problems which are specific to IR with XML documents. In particular we discuss an alternative to the format in which the content and structure topics had to be specified.

2. SYSTEM DESCRIPTION

2.1 The INEX collection

The INEX collection, 21 IEEE Computer Society journals from 1995–2002, consists of 12,135 (when ignoring the volume.xml files) documents with extensive XML-markup. The ten most frequent words in the collection are the following: data (169886), time (161415), system (149249), pp (137334), computer (119732), systems (116221), software (106552), fig (103141), set (99865), and ensp (99723).

2.2 The FlexIR information retrieval system

All submitted runs used FlexIR, an information retrieval system developed at the University of Amsterdam [5]. The main goal underlying FlexIR's design is to facilitate flexible experimentation with a wide variety of retrieval components and techniques. FlexIR is implemented in Perl; as it is built around the standard UNIX pipeline architecture, and supports many types of preprocessing, scoring, indexing, and retrieval tools, which proved to be a major asset for INEX task. The retrieval model underlying FlexIR is the standard vector space model. All our runs used the Lnu.ltc weighting scheme [1] to compute the similarity between a query and a document. For the experiments on which we report in this note, we fixed *slope* at 0.2; the pivot was set to the average number of unique words per document.

2.3 Morphological Normalization

Our aim was to study the effect of morphological normalization on the retrieval. To obtain a zero-knowledge language independent approach to morphological normalization, we implemented an ngram-based method in addition to a linguistically informed method. The ngram length was set to five. For each word we stored both the word itself and all possible ngrams that can be obtained from it without crossing word boundaries. For instance, topic 01 contained the phrase *description logics*. Using ngrams of length five, this becomes:

> *description descr escri scrip cript ripti iptio ption*
> *logics logic ogics*

For the linguistically informed method with which we wanted to contrast the effect of the ngram-method we used Porter stemming [7]. In both approaches we removed words occurring on a stop list with 391 words. Figure 1 contains the original topic 31 and the stemmed version used as FlexIR input.

```
<INEX-Topic topic-id="31" query-type="CO" ct-no="003">
      <Title>
         <cw>computational biology</cw>
      </Title>
      <Description>
         Challenges that arise, and approaches being
         explored, in the interdisciplinary field of
         computational biology.
      </Description>
      ...
</INEX-Topic>
```

(a)

```
.i 31
comput biologi challeng aris approach
explor interdisciplinari field comput
biologi
```

(b)

Figure 1: Topic 31, original (a) and stemmed (b)

2.4 Combined Runs

We also wanted to experiment with combinations of (what we believed to be) different kinds of runs in an attempt to determine their impact on retrieval effectiveness. More specifically, we created a base run using the Porter stemmer and one in which we used ngrams in the manner described above. We then combined these two runs in the following manner. First, we normalized the retrieval status values (RSVs), since different runs may have radically different RSVs. For each run we reranked these values in $[0, 1]$ using:

$$RSV_i' = \frac{RSV_i - min_i}{max_i - min_i}$$

and assigned the value0 to all documents not occurring in the top 1000. This is the Min_Max_Norm considered in [4]. Next, we assigned new weights to the documents using a linear interpolation factor λ representing the relative weight of a run:

$$RSV_{new} = \lambda \cdot RSV_1 + (1 - \lambda) \cdot RSV_2.$$

For $\lambda = 0.5$ this is similar to the simple (but effective) combSUM function used by Fox and Shaw [3]. The interpolation factor λ was set to 0.6 for the ngram run. This higher weight for the ngram run was motivated by experiments on the CLEF [2] data sets.

2.5 Basic Architecture

In this subsection we describe how the runs were performed. The documents were processed as follows. The original xml-docs are standardized, tokenized and, if needed, stemmed or ngrammed. Words in the documents are transformed into records and inverted. This results in an index for running the FlexIR retrieval program.

For the content-only topics, we follow the classical IR approach: Only the words in the title and description fields are selected. These words are, stopped and if needed, stemmed or ngrammed. For an example of this transformation after stemming, see Figure 1.

For the content-and-structure topics, we made two translations: First they are processed similar to the content-only topics: only the words in the title and description fields are selected; from the title field we only select the content of the ⟨cw⟩ field. Then the ⟨title⟩ field is processed to preserve the structural part of the query: the first line contains the topic number, the second line gives the xml-field that needs to be returned, the next line(s) give conditions for the document, consisting of a field name, and the words that are sought. For example, topic 01 with the following ⟨title⟩ field

```
<Title>
   <te>au</te>
   <cw>description logics</cw><ce>abs, kwd</ce>
</Title>
```

becomes after stemming

```
.i 01
article/fm/au
abs|kwd, descript logic
```

This should be read as: retrieve the content of `article/fm/au` if `article/fm/abs` or `article/fm/kwd` contains the words `descript` or `logic`.

For the content-only topics, we simply run the (naive, stemmed, or ngrammed) topics on the (naive, stemmed, or ngrammed) document index. The 100 documents with the highest FlexIR relevance assessment (RSV) are selected.

The runs for the content-and-structure topics are more complex: First we run the first translation of the (naive, stemmed, or ngrammed) topics on the (naive, stemmed, or ngrammed) document index, preselecting the most promising 1000 documents per topic. Our working hypothesis is that all relevant document are in this top 1000. For each topic, we create a special xml-file containing these top 1000 documents. On these so-called docpiles, we run an xml-parser based on Perl's `XML::Twig` that handles xpath expressions, and select the required field(s) from the documents that satisfy the conditions as specified in the second translation of the topic. In addition, we count the number of matching search words. The result is a `twig` file having the raw scored xml-elements in non-sorted order. Finally we select the (maximally) 100 xml-elements that have the highest number of matches, and sort them according to the original FlexIR relevance assessment.

3. RUNS

For the INEX 2002, we submitted three official runs:

UAmsI02Stem Stemmed index and topics, using Lnu.ltc weighting, and feedback.

For the 'content and structure' topics, the stemmed documents were used to create docpiles of the top 1000 documents of the stemmed run. The stemmed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

UAmsI02NGram Ngrammed index and topics, using Lnu.ltc weighting, and feedback. We used ngram-length 5, adding ngrams for words with length ≥ 4, while also keeping the the originals words.

For the 'content and structure' topics, the stemmed documents were used to create docpiles of the top 1000 documents of the ngrammed run. The stemmed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

UAmsI02NGiSt Combined run using 0.6 Ngram, and 0.4 Stemmed.

For the 'content and structure' topics, the stemmed documents were used to create docpiles of the top 1000 documents of the combined ngram-stemmed run. The stemmed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

3.1 Post submission runs for INEX

UAmsI02Word We create a naive, word-based run (still stopping, and lowercasing strings) by using a ngram-length of 100. Again, we use Lnu.ltc weighting, and feedback.

For the 'content and structure' topics, the stemmed documents were used to create docpiles of the top 1000 documents of the word-based run. The stemmed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

UAmsI02NGramOnNGram Ngrammed index and topics, using Lnu.ltc weighting, and feedback. We used ngram-length 5, adding ngrams for words with length ≥ 4, while also keeping the the originals words.

For the 'content and structure' topics, the ngrammed documents were used to create docpiles of the top 1000 documents of the ngrammed run. The ngrammed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

UAmsI02WordOnWord We create a naive, word-based run (still stopping, and lowercasing strings) by using a ngram-length of 100. Again, we use Lnu.ltc weighting, and feedback.

For the 'content and structure' topics, the word-based documents were used to create docpiles of the top 1000 documents of the word-based run. The word-based structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

4. PROBLEMS WITH XML SYNTAX

The first few releases of the collection had a number of problems related to the XML syntax. In our approach we needed to do a morphological transform of the free text part of the documents but leave the XML structure intact. Because we used non forgiving XML parsers like TWIG it was very important to have and to keep valid XML documents. Here are some of the encountered problems:

- Deciding which broken tags to repair. Sometimes authors use tricks like `;tag>` to indicate that a tag should not be evaluated, but taken literally. This trick works with forgiving web browsers, but it is incorrect XML. The correct 'trick' would be `<tag>`, but what quite frequently the author actually meant to write was <tag>. If this is the case then translating `<tag>` to `<tag>` breaks XML validity because one is left with a </tag> somewhere that has no opening <tag> anymore.

- Dealing with embedded TEX and LATEX proved to be quite difficult, because there were math formulas of the shape $$i<k>2$$, which leads an XML parser to believe that <k> is a tag. We decided to remove embedded mathematics using the XML-tags, i.e., ⟨tf⟩...⟨/tf⟩ and ⟨tmath⟩...⟨/tmath⟩. This helps avoiding XML parser errors due to use of sequences, ⟨...⟩, in math portions of the documents.

- Some documents have tags that contain newlines. For example, `so/2001/s5071` has tags like

  ⟨
  fig⟩

  that span two lines. These tags are lost in our index, and gave parser errors. We rectify this by removing newlines that occur inside a tag.

- We used the following forgiving, yet not too liberal regular expression for XML tags:

  ```
  <\/?\w+\s*(\w+\s*=\s*[\'\"]?[^\'\"\>]+[\'\"]?\s*)*\/?>
  ```

  It works fine but matches things like <p[\n]+>, which could cause trouble when operating on the file line by line. The collection contains tag attribute values like <li t="(3.9) "> which the regular expression should catch.

5. TRANSFORMING INEX TOPICS INTO XPATH EXPRESSIONS

Our initial strategy was to use an XML query engine like Kweelt or Twig for the content and structure topics. For this reason we sought a way of automatically translating the INEX topic format into XPATH expressions. This turned out impossible for a number of reasons, one was that the

topic authors used operators for Boolean expressions and joins, but not in a uniform way, nor using uniform notation. Also, as evidenced in the discussion list, the meaning of "," was not always clear.

It seems that these reasons can be overcome once a fixed topic language is given to the authors. We found though two deeper reasons why INEX topics cannot be transformed automatically into XPATH expressions. The first is that the use of (implicit) descendant axis in paths leads to problems of ambiguity and under-specification. The second is due to the fact that one cannot specify a connection between the ⟨te⟩ field (which is to be returned) and the ⟨cw⟩ and ⟨ce⟩ fields (which contain the conditions to be checked). From this we conclude that the INEX topic format is not a suitable question format and propose an alternative.

Incomplete Paths

A very simple INEX topic is

```
<te> article </te>
<cw> logic </cw> <ce> kwd</ce>
```

This means *retrieve articles with "logic" as a keyword.* The equivalent XPATH expression (assuming all articles are in one file and all articles are contained in the XPATH expression `/article`) would be

```
/article[contains(.//kwd,''logic'')]
```

Another simple INEX topic is

```
<te> au </te>
<cw> logic </cw> <ce> kwd</ce>
```

This could be rephrased as *retrieve all authors of articles with "logic" among the keywords.* The corresponding XPATH expression is not possible to give without knowing the exact DTD and the paths to `au` and `kwd`. The naive solution

```
/article//au[
   contains(./ancestor::article//kwd,''logic'')]
```

might be too general. It contains for instance

```
/article/bm/bib/bibl/bb/
  au[contains(../../../../../fm/kwd,''logic'')]
```

but these are authors whose work is cited from articles containing "logic" as a keyword.

With the DTD for the INEX collection, the following seems the correct translation. It retrieves authors of articles in the INEX databases with logic among the keywords.

```
/article/fm/au[contains(../kwd,''logic'')]
```

But of course this is an interpretation of the original topic.

Comma's

Inside the `te, cw` and `ce` tags a comma separated list may occur. According to the instructions comma should be read as disjunction. This may lead to ambiguity, as the following example shows. Consider the topic *retrieve all author or editor names containing "John".* The following XPATH expression just gives that

```
//author/name[contains(.,''John'')] |
//editor/name[contains(.,''John'')]
```

Note that "|" denotes the join (union) of the two sets of author names.

It seems impossible to formulate this as an INEX topic. The obvious

```
<te> //author/name,//editor/name</te>
<cw> John </cw> <ce>//author/name,//editor/name</ce>
```

can not be correct. How could we translate this to an XPATH expression while keeping the connection between what is being returned and what is being checked? The join has to be done after the two (independent) retrievals. We can not specify this topic in the INEX format.

Our translation

We translated the INEX topics to XPATH expressions as follows:

- we replaced `<te>A,B,...</te>` (disjunctive search tags) by `<te>/article</te>`.
- we expanded element names in `te` and `ce` tags to unique paths (as in the example above). In case of a choice we used the whole topic to determine which path was meant.
- As the files contained at most one article (at least that was our assumption) we could work with the following translation. This has to be adjusted in case there are more articles in one file. Consider the following example INEX topic

  ```
  <te> T </te>
  <cw> K1,K2</cw><ce> CT1</ce>
  <cw> L1</cw><ce> CT2,CT3</ce>
  ```

  This topic would be translated into the XPATH expression

  ```
  T[(contains(CT1,K1) or contains(CT1,K2))
  and (contains(CT2,L1) or contains(CT3,L1))].
  ```

Proposal for INEX content-and-structure topic format

A union of xpath expressions in the last format is a good alternative to INEX topics. It provides more expressive power (because of the use of the context node in the contains expressions), and it is not ambiguous (because the implicit use

of descendant axis is forbidden; only complete paths which are valid under the DTD can be used).

We think that such a strict format yields better results, both in retrieval and in assessment. With the INEX format, topic translation was a creative process. Even the topic description was often not complete enough to yield a unique interpretation.

6. ACKNOWLEDGMENTS

We want to thank Willem van Hage for his technical support. Jaap Kamps was supported by the Netherlands Organization for Scientific Research (NWO), grant # 400-20-036. Maarten Marx received support from NWO grant 612.000.106. Maarten de Rijke was supported by grants from NWO, under project numbers 612-13-001, 365-20-005, 612.069.006, 612.000.106, 220-80-001, and 612.000.207.

7. REFERENCES

[1] C. Buckley, A. Singhal, and M. Mitra. New retrieval approaches using SMART: TREC 4. In D. Harman, editor, *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*, pages 25–48. NIST Special Publication 500-236, 1995.

[2] CLEF. Cross language evaluation forum, 2002. `http://www.clef-campaign.org/`.

[3] E. Fox and J. Shaw. Combination of multiple searches. In *Proceedings TREC-2*, pages 243–252, 1994.

[4] J. Lee. Combining multiple evidence from different relevant feedback networks. In *Database Systems for Advanced Applications*, pages 421–430, 1997.

[5] C. Monz and M. de Rijke. Shallow morphological analysis in monolingual information retrieval for Dutch, German and Italian. In C. Peters, M. Braschler, J. Gonzalo, and M. Kluck, editors, *Evaluation of Cross-Language Information Retrieval Systems, CLEF 2001*, volume 2406 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2002.

[6] C. Monz, J. Kamps, and M. de Rijke. The University of Amsterdam at CLEF-2002. In C. Peters, editor, *Results of the CLEF 2002 Cross-Language System Evaluation Campaign*, pages 73–84, 2002.

[7] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.