The University of Amsterdam at INEX 2003

Jaap Kamps Maarten de Rijke Börkur Sigurbjörnsson

Language & Inference Technology Group, University of Amsterdam
Nieuwe Achtergracht 166, 1018 WV Amsterdam, The Netherlands
E-mail: {kamps, mdr, borkur}@science.uva.nl

ABSTRACT

This paper describes the INEX 2003 participation of the Language & Inference Technology group of the University of Amsterdam. We participated in all three of the tasks, content-only, strict content-and-structure and vague content-and-structure.

1. INTRODUCTION

One of the recurring issues in XML retrieval is finding the appropriate unit of retrieval. For the content-only (CO) task at INEX 2002, we only submitted runs in which whole articles were the unit of retrieval [3]. Much to our surprise, retrieving articles turned out to be a competitive strategy. In [5] we experimented with going below the article level and returning elements. Our experiments showed that a successful element retrieval approach should be biased toward retrieving large elements. For the content-only task this year our aim was to experiment further with this size bias, in order to try to determine what is the appropriate unit of retrieval.

For the strict content-and-structure (SCAS) task the unit of retrieval is usually explicitly mentioned in the query. Our research question for the content-only task does therefore not carry over to the strict content-and-structure task. At INEX 2002, we experimented with assigning an RSV score to elements satisfying an XPath expression. This year we experiment further with the same idea, although our scoring methods are quite different from those of last year.

The vague content-and-structure (VCAS) task is a new task and we could not base our experiments on previous experience. Since the definition of the task was underspecified, our aim for this task was to try to find out what sort of task this was. We experimented with a content-only approach, strict content-and-structure approach and article retrieval approach.

All of our runs were created using the `FlexIR` retrieval system developed by the Language and Inference Technology group. We use a multinomial language model for the scoring of retrieval results.

The structure of the remainder of this paper is as follows. In Section 2 we describe the setup used in our experiments. In Section 3 we explain the submitted runs for each of the three tasks, CO in 3.1, SCAS in 3.2 and VCAS in 3.3. Results are presented and discussed in Section 4 and in Section 5 we draw initial conclusions from our experiments.

2. EXPERIMENTAL SETUP

2.1 Index

We adopt an IR based approach to XML retrieval. We created our runs using two types of inverted indexes, one for XML articles only and another for all XML elements.

Article index

For the article index, the indexing unit is a whole XML document containing all the terms appearing at any nesting level within the ⟨`article`⟩ tag. This is thus a traditional inverted index as used for standard document retrieval.

Element index

For the element index, the indexing unit can be any XML element (including ⟨`article`⟩). For each element, all text nested inside it is indexed. Hence the indexing units overlap (see Figure 1). Text appearing in a particular nested XML element is not only indexed as part of that element, but also as part of all its ancestor elements.

The article index can be viewed as a restricted version of the element index, where only elements with tag-name ⟨`article`⟩ are indexed.

Both indexes were word-based, no stemming was applied to the documents, but the text was lower-cased and stop-words were removed using the stop-word list that comes with the English version on the Snowball stemmer [8]. Despite the positive effect of morphological normalization reported in [3], we decided to go for a word-based approach. Some of our experiments have indicated that high precision settings are desirable for XML element retrieval [4]. Word-based approaches have proved very suitable for achieving high precision.

2.2 Query processing

Two different topic formats are used, see Figure 2 for one of the CO topics, and Figure 3 for one of the CAS topics. Our queries were created using only the terms in the ⟨`title`⟩ and ⟨`description`⟩ parts of the topics. Terms in the ⟨`keywords`⟩ part of the topics have proved to significantly improve retrieval effectiveness [4]. The keywords, which are used to assist during the assessment stage, are often based on human inspection of relevant documents during the topic creation. Using them would have meant a violation of the

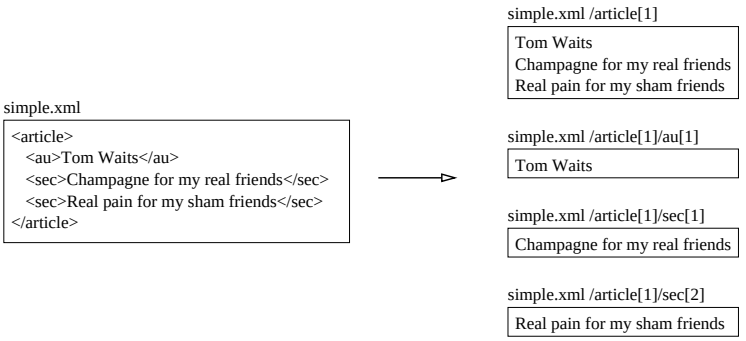

Figure 1: Simplified figure of how XML documents are split up into overlapping indexing units

assumptions underlying a fully automatic run, so we decided not to use them. Our system does not support +, - or phrases in queries. Words and phrases bound by a minus were removed, together with the minus-sign. Plus-signs and quotes were simply removed.

Like the index, the queries were word-based, no stemming was applied but the text was lower-cased and stopwords were removed.

For some of our runs we used queries expanded by blind feedback. We considered it safer to perform the blind feedback against the article index since we do not know how the overlapping nature of the element index affects the statistics used in the feedback procedure. We used a variant of Rocchio feedback [6], where the top 10 documents were considered relevant; the top 501-1000 were considered non-relevant; and up to 20 terms were added to the initial topic. Terms appearing in more that 450 documents were not considered as feedback terms. An example of an expanded query can be seen in Figure 2*c*.

Task specific query handling will be further described as part of the run descriptions in the following section.

2.3 Retrieval model

All our runs use a multinomial language model with Jelinek-Mercer smoothing [2]. We estimate a language model for each of the elements. The elements are then ranked according to the likelihood of the query, given the estimated language model for the element. To account for data sparseness we estimate the element language model by a linear interpolation of two language models, one for the element and another for the collection:

$$P(E|Q) = P(E) \cdot \prod_{i=1}^{k} (\lambda \cdot P_{mle}(t_i|E) + (1-\lambda) \cdot P_{mle}(t_i|C)), \quad (1)$$

where Q is a query made out of the terms $t_1, \ldots, t_k$; E is a language model of an element; C is a language model of the collection; and λ is the interpolation factor (smoothing parameter). We estimate the language models, $P_{mle}(\cdot|\cdot)$ using maximum likelihood estimation. For the collection model we use element frequencies. Assuming a uniform prior probability of elements being relevant, our basic scoring formula for an element E and a query $Q = (t_1, \ldots, t_k)$ is therefore

$$s(E,Q) = \sum_{i=1}^{k} \log \left(1 + \frac{\lambda \cdot \mathrm{tf}(t_i,E) \cdot (\sum_t \mathrm{df}(t))}{(1-\lambda) \cdot \mathrm{df}(t_i) \cdot (\sum_t \mathrm{tf}(t,E))}\right), \quad (2)$$

where $\mathrm{tf}(t,E)$ is the frequency of term t in element E, $\mathrm{df}(t)$ is the element frequency of term t and λ is the smoothing parameter. In most cases we base the probability $P(E)$ on the element length. That is, we add a length prior to the score:

$$\mathrm{lp}(E) = \log \left(\sum_t \mathrm{tf}(t,E)\right). \quad (3)$$

For an exact description of how we apply this length prior, see individual run descriptions in Section 3.

The smoothing parameter λ played a crucial role in our submissions. In [4] we reported on the effect of smoothing on the unit of retrieval. The results obtained there suggested that there was a correlation between the value of the smoothing parameter and the size of the retrieved elements. The average size of retrieved elements increases dramatically as less smoothing (a higher value for the smoothing parameter λ) was applied. Further descriptions on how we tried to exploit this size-smoothing relation are provided in the individual run descriptions.

Smoothing is not the only method applicable to eliminate the small elements from the retrieval set. One can also simply discard the small elements when building the index. Elements containing text that is shorter than a certain cut-off value can be ignored when the index is built. In some of our runs we imitated such index building by restricting our view of the element index to a such a cut-off version. Further details will be provided in the description of individual runs in the next section.

3. RUNS

3.1 Content-Only task

In [5] we tried to answer the question of what is the appropriate unit of retrieval for XML information retrieval. A general conclusion was that users have a bias toward large elements. With our runs for the content-only task we pursued this issue further.

We create a language model for each XML-element. As described in the previous section, since small elements do not provide a large sample space, we get very sparse statistics. We therefore smooth our statistics by combining the element language model with a language model for the whole collection. Zhai and Lafferty [10] argue that bigger documents require less smoothing than smaller documents. A similar effect was witnessed in [4], where less smooth language models retrieved larger elements than more smooth language models.

Increasing the value of λ in the language model causes an occurrence of a term to have an increasingly bigger impact. As a result,

the elements with more matching terms are favored over elements with fewer matching terms. In the case of our overlapping element index, a high lambda gives us an article biased run, whereas a low lambda introduces a bias toward smaller elements (such as sections and paragraphs).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="103" query_type="CO" ct_no="50">
 <title>UML formal logic</title>
 <description>Find information on the use of formal logics
  to model or reason about UML diagrams.</description>
 <narrative>...</narrative>
 <keywords>...</keywords>
</inex_topic>
```

(a) Original topic

```
.i 103
uml formal logic find information use formal logics model
reason uml diagrams
```

(b) Cleaned query (TD)

```
.i 103
uml formal logic find information use formal logics model
reason uml diagrams booch longman rumbaugh itu jacobson
wiley guards ocl notations omg statecharts formalism
mappings verlag sdl documenting stereotyped semantically
sons saddle
```

(c) Expanded query (TD+blind feedback)

Figure 2: CO Topic 103

In our runs we scored elements by combining evidence from the element itself, $s(e)$, and evidence from the surrounding article $s(d)$, using the scoring formula

$$s_{comb}(e) = \mathrm{lp}(e) + \alpha \cdot s(d) + (1-\alpha) \cdot s(e) \qquad (4)$$

where $s(\cdot)$ is the score function from Equation 2 and $\mathrm{lp}(\cdot)$ is the length prior from Equation 3.

We submitted the following runs for the CO task. Since we wanted to experiment with the element size bias, we wanted to compare two runs, one with considerable smoothing ($\lambda = 0.2$) and another with considerably less smoothing ($\lambda = 0.9$).

UAmsI03-CO-lambda=0.9

In this run we set the smoothing parameter λ to 0.9. This value of λ means that little smoothing was performed, which resulted in a run with a bias toward retrieving large elements such as whole articles.

UAmsI03-CO-lambda=0.2

In this run we set the smoothing parameter λ to 0.2 which means that a considerable amount of smoothing is performed. This resulted in a run with a bias toward retrieving elements such as sections and paragraphs.

UAmsI03-CO-lambda=0.5

Here we went somewhere in between the two extremes above by setting $\lambda = 0.5$. Furthermore, we required elements to be either articles, bodies or nested within the body.

All runs used the same combination value $\alpha = 0.4$ in the scoring formula (4), a value chosen after experimenting with the INEX 2002 collection. Only elements longer than 20 terms were considered. Very short pieces of text are themselves not likely to be very informative. One straightforward way to make sure those short elements are not retrieved, is to remove them from the index. The value for the size threshold was justified by experiments on the INEX 2002 collection.

As described previously, queries were created using the terms from the title and description; they were not stemmed but stop-words were removed (See Figure 2*b*). The queries were expanded using blind feedback (See Figure 2*c*). The parameters for the feedback were based on experiments with the INEX 2002 collection. In our score calculations we used the overlapping element index as a basis for the collection language model. In our combinations of article and element scores we did not do any normalization of scores.

3.2 Strict Content-And-Structure task

For the Strict Content-and-structure (SCAS) task the unit of retrieval is usually coded inside the topics. Our research question for the CO task does therefore not carry directly over to the SCAS task.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="76" query_type="CAS" ct_no="81">
 <title>//article[(./fm//yr='2000' OR
  ./fm//yr='1999') AND about(.,'"intelligent
  transportation system"')]//sec[about(.,
  'automation +vehicle')]</title>
 <description>Automated vehicle applications
  in articles from 1999 or 2000 about intelligent
  transportation systems.</description>
 <narrative>...</narrative>
 <keywords>...</keywords>
</inex_topic>
```

(a) Original topic

```
.i 76
intelligent transportation system automation
vehicle automated vehicle applications in
articles from 1999 or 2000 about intelligent
transportation systems
```

(b) Full content query (TD)

```
.i 76a article
intelligent transportation system
.i 76b sec
automation vehicle
```

(c) Partial content queries(T)

```
//article[about(., "76a")]//sec[about(.,"76b")]
```

(d) Fuzzy structure (T)

```
//article[./fm//yr='2000' or ./fm//yr='1999']//sec
```

(e) Strict structure (T)

Figure 3: CAS Topic 76

The CAS topics have a considerably more complex format than the CO topics (see Figure 3*a* for an example). The description part is the same, but the title has a different format. The CAS title is written in a language which is an extension of a subset of XPath [9]. We can view the title part of the CAS topic as a mixture of path expressions and filters. Our aim with our SCAS runs was to try to cast light on how these expressions and filters could be used to

assign scores to elements. All our runs treat the filters in quite a strict fashion; the larger number the number of filters that are satisfied, the higher the ranking of an element. The difference between our three runs lies in the way we decide the ranking of results that satisfy the same number of filters.

More precisely, we consider the topic title of CAS topics to be split into path expressions and filters as follows.

$$\texttt{rootPath}[F_r \cup C_r \cup S_r]\texttt{targetPath}[F_e \cup C_e \cup S_e], \quad (5)$$

where `rootPath` and `targetPath` are path expressions and F_r, C_r, S_r, F_e, C_e, S_e are sets of filters (to be explained below). The filters in the actual topics were connected with a boolean formula. We ignore this formula and only look at sets of filters. We distinguish between three types of filters.

Element filters (F) F is a set of filters that put content constraints on the current element, as identified by preceding path expression (`rootPath` or `targetPath`). Element filters have the format `about(.,'whatever')`

Nested filters (C) C is a set of filters that put content constraints on elements that are nested within the current element. Nested filters have the format `about(./path, 'whatever')`

Strict filters (S) S is a set of filters of the format `path op value`, where `op` is a comparison operator such as = or >=; and value is a number or a string.

As an example, the title part of Topic 76 in Figure 3*a* can be broken up into path expressions and filters such as:

`rootPath` = `//article`

$F_r = \{$`about(.,'"intelligent transportation system"')`$\}$

$C_r = \emptyset$

$S_r = \{$`./fm//yr='2000'`, `./fm//yr='1999'`$\}$

`targetPath` = `//sec`

$F_e = \{$`about(.,'automation +vehicle')`

$C_e = \emptyset$

$S_e = \emptyset$

We calculate the retrieval scores by combining 3 base runs. The base runs consist of an *article run*, a ranked list of articles answering the full content query (Figure 3*b*); an *element run*, a ranked list of target elements answering the full content query (Figure 3*b*); and a *filter run*, a ranked list of elements answering each of the partial content queries (Figure 3*c*). More precisely the base runs were created as follows.

Article run

We created an article run from the element index by filtering away all elements not having the tag-name ⟨`article`⟩. We used a value $\lambda = 0.15$ for the smoothing parameter. This is the traditional parameter settings for document retrieval. We used the full content query (Figure 3*b*), expanded using blind feedback. For each query we retrieved a ranked list of 2000 most relevant articles.

Element run

We created an element run in a similar fashion as for the CO task. Additionally, we filtered away all elements that did not have the same tag-name as the target tag-name (the rightmost part of the `targetPath`). For topics where the target was a '*' we considered only elements containing at least 20 terms. We did moderate smoothing by choosing a value of 0.5 for λ. We used the full content queries (Figure 3*b*), expanded using blind feedback. For each query we retrieved an exhaustive ranked list of relevant elements.

Filter run

We created an element run in a similar fashion as for the CO task, but using the partial content queries (Figure 3*c*). No blind feedback was applied to the queries. We filtered away all elements that did not have the same tag-name as the target tag-name of each filter. For filters where the target was a '*' we considered only elements containing at least 20 terms. We did minor smoothing by choosing the value 0.7 for λ. For each query we retrieved an exhaustive ranked list of relevant elements.

For all the base runs the length prior from equation 3 is added to the score.

From the base runs we created three runs which we submitted: one where scores are based on the element run; another where scores are based on the article run; and a third which uses a mixture of the element run, article run and filter run.

UAmsI03-SCAS-ElementScore

The articles appearing in the article run were parsed and their elements that matched any of the element- or nested-filters were kept aside as candidates for the final retrieval set. In other words, we kept aside all elements that matched the title XPath expression, where the about predicate returns the value `true` for precisely the elements that appear in the filter run. The candidate elements were then assigned a score according to the element run. Additionally, results that match all filters got 100 extra points. Elements that match only the target filters got 50 extra points. The values 100 and 50 were just arbitrary numbers used to guarantee that the elements matching all the filters were ranked before the elements only matching a strict subset of the filters. This can be viewed as a coordination level matching for the filter matching.

UAmsI03-SCAS-DocumentScore

This run is almost identical to the previous run. The only difference was that the candidate elements were assigned scores according to the article run instead of according to the element run.

UAmsI03-SCAS-MixedScore

The articles appearing in the article run are parsed in the same way as for the two previous cases. The candidate elements are assigned a score which is calculated by combining the RSV scores of the three base runs. Hence, the score of an element is a mixture of its own score, the score of the article containing it, and the scores of all elements that contribute to the XPath expression being matched. More precisely, the element score was calculated using the formula

$$RSV(e) = \alpha \cdot \left(s(r) + \sum_{f \in F_r} s(f) + \sum_{c \in C_r} \max s(c) \right) + (1-\alpha) \cdot \left(s(e) + \sum_{f \in F_e} s(f) + \sum_{c \in C_e} \max s(c) \right), \quad (6)$$

where F_r, C_r, F_e and C_e represent sets of elements passing the respective filter mentioned in Equation 5; $s(r)$ is the score of the article from the article run; $s(f)$ and $s(c)$ are scores from the filter

	MAP	p@5	p@10	p@20
$\lambda = 0.9$	0.1091	0.3308	0.2769	0.2250
$\lambda = 0.2$	**0.1214**	0.3231	**0.2923**	**0.2423**
$\lambda = 0.5$	0.1143	**0.3462**	**0.2923**	0.2346

Table 1: Results of the CO task

run; and $s(e)$ is the score from the element run. In all cases we set $\alpha = 0.5$. We did not have any training data to estimate an optimal value for this parameter. We did not apply any normalization to the RSVs before combining them.

For all the SCAS runs, the elements are also filtered using the strict filters (Figure 3*e*). Any filtering using tag-names used the tag equivalence relations defined in the topic development guidelines.

3.3 Vague Content-And-Structure task

Since the definition of the task was a bit underspecified, we did not have a clear idea about what this task was about. With our runs we tried to cast light on whether this task is actually a content-only task, a content-and-structure task or a traditional article retrieval task.

UAmsI03-VCAS-NoStructure

This is a run that is similar to our CO runs. We chose a value $\lambda = 0.5$ for the smoothing parameter. We used the full content queries, expanded by blind feedback. We only considered elements containing at least 20 terms.

UAmsI03-VCAS-TargetFilter

This run is more similar to our SCAS runs. We chose a value $\lambda = 0.5$ for the smoothing parameter. We used the full content queries, expanded by blind feedback. Furthermore, we only returned elements having the same tag-name as the rightmost part of `targetPath`. Where the target element was not explicitly stated (*-targets), we only considered elements containing at least 20 terms.

UAmsI03-VCAS-Article

This run is a combination of two article runs using unweighted combSUM [7]. The two runs differ in the way that one is aimed at recall but the other at high precision. The one that aims at recall used $\lambda = 0.15$ and the full content queries, expanded by blind feedback. The high precision run used $\lambda = 0.70$ and as queries only the text appearing in the filters of the topic title. The RSV values of the runs were normalized before they were combined.

For all the VCAS runs, the length prior from equation 3 was added to the score.

4. RESULTS

Our runs were evaluated using version 2003.004 of the evaluation software provided by the INEX 2003 organizers. We used version 2.3 of the assessments. All runs were evaluated using the strict quantization; i.e., an element is considered relevant if and only if it is highly exhaustive and highly specific.

4.1 Content-Only task

Table 1 shows the results of the CO runs. Figure 4 shows the precision-recall plots. The CO runs at INEX 2003 were evaluated using *inex_eval*, the standard precision-recall measure for INEX.

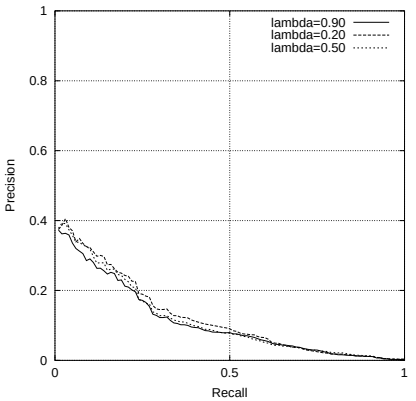

Figure 4: Precision-recall curves for our CO submissions, using the strict evaluation measure

	MAP	p@5	p@10	p@20
ElementScore	0.2650	**0.4273**	**0.3455**	0.2432
DocumentScore	0.2289	0.2909	0.2636	0.2136
MixedScore	**0.2815**	0.4000	0.3318	**0.2773**

Table 2: Results of the SCAS task

Furthermore, two other measures were developed, *inex_eval_ng(s)*, a precision recall measure that takes size of retrieved components into account; and *inex_eval_ng(o)*, which considers both size and overlap of retrieved components [1]. At the time when this report is written, a working version of the latter two measures had not been released. We will therefore only report on our results using the inex_eval measure.

It looks like our runs are very similar. There is not much difference in scoring and the graphs look the same. It seems that index size cut-off reduces the effect of the smoothing parameter reported in [4], where the absence of smoothing provided bias toward larger elements. Here the cut-off already eliminates the smallest elements and there is less need for extreme size bias.

According to the inex_eval measure, the run using $\lambda = 0.2$ has over all highest MAP score. The run using $\lambda = 0.5$ and filter out elements outside the ⟨`bdy`⟩ tag, gives slightly higher precision when 5 elements were retrieved. The run using $\lambda = 0.2$ does however catch up quite quickly. The runs seem to be so similar that any differences are unlikely to be statistically significant.

4.2 Strict Content-And-Structure task

Table 2 shows the results of the SCAS runs. Figure 5 shows the precision-recall plots. The run using the combination of element-, document- and filter-RSVs has higher MAP than the other two runs. The run based on element scores has slightly lower MAP than the combination run. The run based on document scores has the lowest MAP.

The run based on element scores outperforms the other two at low recall levels. We can see from the table that the element based run has the highest precision after only 5 or 10 documents have been retrieved. The combination run catches up with the element score based run once 20 documents have been retrieved. This indicates

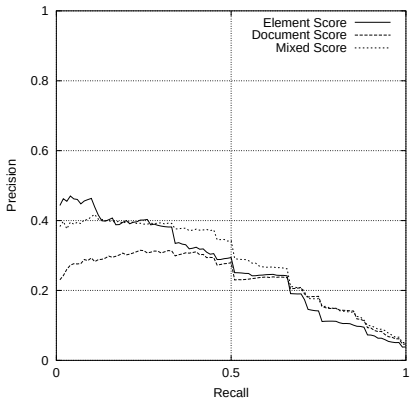

Figure 5: Precision-recall curves for our SCAS submissions, using the strict evaluation

	MAP	p@5	p@10	p@20
NoStructure	**0.1270**	**0.2880**	0.2520	**0.1980**
TargetFilter	0.0647	**0.2880**	**0.2640**	0.1960
Article	0.0627	0.2080	0.1800	0.1300

Table 3: Results of the VCAS task

that the coordination level matching for the filter matching, works well for initial precision, but is not as useful at higher recall levels.

4.3 Vague Content-And-Structure task

Table 3 shows the results of the VCAS runs. Figure 6 shows the precision-recall plots. Treating the VCAS task as a CO task, results in a higher MAP than either treating it as an SCAS task or an article retrieval task. The main difference lies in the recall. By limiting the set of returned elements, by considering the structure or retrieving articles, we discard elements that have a big potential of being relevant to the user. Hence we can never obtain maximal recall.

Looking at the precision at low recall levels, the difference between

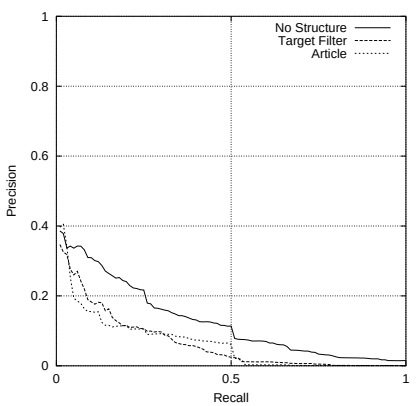

Figure 6: Precision-recall curves for our VCAS submissions, using the strict evaluation

the two element retrieval runs is not so great. If we look at precision after 5, 10 and 20 elements have been retrieved, we see that treating the VCAS task as an SCAS task performs comparable to treating it as a CO task. The strict implementation of the task can even help early precision.

5. CONCLUSIONS

This paper described our official runs for the INEX 2003 evaluation campaign. Our main research question was to further investigate the appropriate unit of retrieval. Although this problem is most visible for INEX's CO task, it also plays a role in the element and filter base runs for the CAS topics. With default adhoc retrieval settings, small XML elements dominate the ranks of retrieved elements. We conducted experiments with a number of approaches that aim to retrieve XML elements similar to those receiving relevance in the eyes of the human assessors. First, we experimented with a uniform length prior, ensuring the retrieval of larger sized XML elements [5]. Second, we experimented with Rocchio blind feedback, resulting in longer expanded queries that turn out to favor larger XML elements than the original queries. Third, we experimented with size cut-off, only indexing the element that contain at least 20 words. Fourth, we experimented with an element filter, ignoring elements occurring in the front and back matter of articles. Fifth, we experimented with smoothing settings, where the increase of the term importance weight leads to the retrieval of larger elements [4]. Finally, we combined approaches in various ways to obtain the official run submission. We plan to give an overview of the relative impact of these approaches in the final proceedings of INEX.

Our future research focuses on the question of what is the appropriate statistical model for XML retrieval. In principle, we could estimate language models from the statistics of the article index similar to standard document retrieval. An alternative is to estimate them from the statistics of the element index, or from a particular subset of the full element index. In particular, we smooth our element language model with collection statistics from the overlapping element index. Arguably, this may introduce biases in the word frequency and document frequency statistics. Each term appearing in an article usually creates several entries in the index. The overall collection statistics from the index may not best estimator for the language models. In our current research we investigate the various statistics from which the language models can be estimated.

6. REFERENCES

[1] N. Gövert, G. Kazai, N. Fuhr, and M. Lalmas. Evaluating the effectiveness of content-oriented XML retrieval. Technical report, University of Dortmund, Computer Science 6, 2003.

[2] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, 2001.

[3] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. The Importance of Morphological Normalization for XML Retrieval. In N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas, editors, *Proceedings of the First Workshop of the Initiaitve for the Evaluation of XML Retrieval (INEX)*, pages 41–48. ERCIM Publications, 2003.

[4] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. Topic Field Selection and Smoothing for XML Retrieval. In A. P. de Vries, editor, *Proceedings of the 4th Dutch-Belgian Information Retrieval Workshop*, pages 69–75. Institute for Logic, Language and Computation, 2003.

[5] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. XML Retrieval: What to Retrieve? In C. Clarke, G. Cormack, J. Callan, D. Hawking, and A. Smeaton, editors, *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 409–410. ACM Press, 2003.

[6] J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System — Experiments in Automatic Document Processing*. Prentice Hall, 1971.

[7] Joseph A. Shaw and Edward A. Fox. Combination of multiple searches. In D.K. Harman, editor, *Proceedings TREC-2*, pages 243–249. NIST, 1994.

[8] Snowball. The snowball string processing language, 2003. `http://snowball.tartarus.org/`.

[9] XPath. Xml path language, 1999. `http://www.w3.org/TR/xpath`.

[10] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM Press, 2001.