An Element-based Approach to XML Retrieval

Börkur Sigurbjörnsson Jaap Kamps Maarten de Rijke

Language & Inference Technology Group, University of Amsterdam
Nieuwe Achtergracht 166, 1018 WV Amsterdam, The Netherlands
E-mail: {borkur, kamps, mdr}@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. Our main strategic lines were to find the appropriate units of retrieval and to mix evidence from several layers in the XML hierarchy.

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 followed an *article-based* approach, i.e. submitted runs in which whole articles were the unit of retrieval [5]. Much to our surprise, this turned out to be a competitive strategy. In [6] 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 we followed an *element-based* approach, and our main aim was to experiment further with this size bias, in order to try to determine what is the appropriate unit of retrieval. Additionally, we experimented scoring elements by mixing evidence from article and element levels.

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. The CAS queries are a mixture of content and structural constraints. We followed an *element-based* approach, and our main aim was to investigate how we could score elements by mixing scores, gained from evaluating the different constraints separately.

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 & 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 [10]. Despite the positive effect of morphological normalization reported in [5], 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

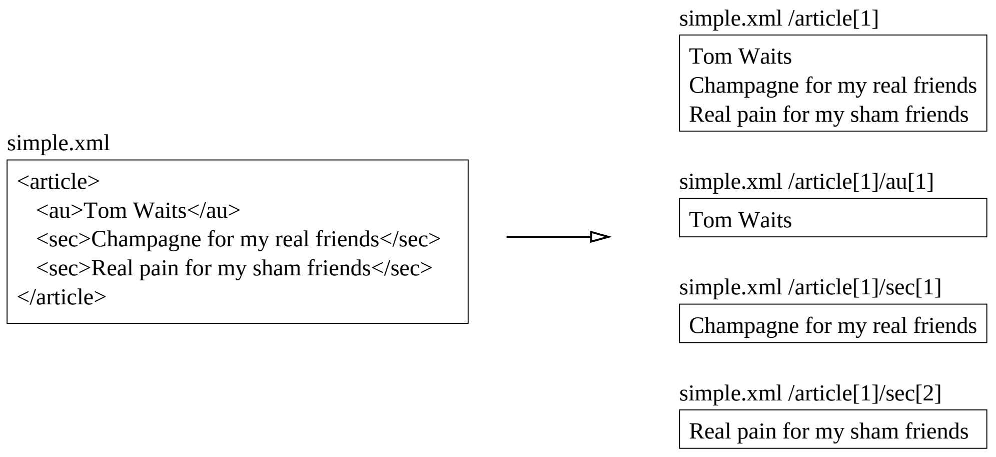
simple.xml
<article>
<au>Tom Waits</au>
<sec>Champagne for my real friends</sec>
<sec>Real pain for my sham friends</sec>
</article>
simple.xml /article[1]
Tom Waits
Champagne for my real friends
Real pain for my sham friends
simple.xml /article[1]/au[1]
Tom Waits
simple.xml /article[1]/sec[1]
Champagne for my real friends
simple.xml /article[1]/sec[2]
Real pain for my sham friends

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

created using only the terms in the ⟨`title`⟩ and ⟨`description`⟩ parts of the topics. Terms in the ⟨`keywords`⟩ part of the topics may 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 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 stop-words were removed.

Blind feedback

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 [7], 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 articles were not considered as feedback terms. The parameters for the feedback were based on experiments with the INEX 2002 collection. 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. That is, we want to estimate the probability

$$P(E,Q) = P(E) \cdot P(Q|E). \tag{1}$$

The two main tasks are thus to estimate the probability of the query, given the element, $P(Q|E)$; and the prior probability of the element, $P(E)$.

Probability of the query

Elements contain a relatively small amount of text, too small to be the sole basis of our element language model estimation. To account for this data sparseness we estimate the element language model by a linear interpolation of two language models, one based on the element data and another based on collection data. Furthermore, we assume that query terms are independent. That is we estimate the probability of the query, given the element language model, using the equation

$$P(Q|E) = \prod_{i=1}^{k} \left(\lambda \cdot P_{mle}(t_i|E) + (1-\lambda) \cdot P_{mle}(t_i|C)\right), \tag{2}$$

where Q is a query made out of the terms $t_1, \ldots, t_k$; E is an element; and C represents the collection. The parameter 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. The estimation of this probability can be reduced to the scoring formula for an element E and a query $Q = (t_1, \ldots, t_k)$,

$$s(E,Q) = \sum_{i=1}^{k} \log\left(1 + \frac{\lambda \cdot \mathrm{tf}(t_i,E) \cdot (\sum_t \mathrm{df}(t))}{(1-\lambda) \cdot \mathrm{df}(t_i) \cdot (\sum_t \mathrm{tf}(t,E))}\right), \tag{3}$$

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.

The smoothing parameter λ played an important role in our submissions. Zhai and Lafferty [13] argue that bigger documents require less smoothing than smaller ones. In [4] we reported on the effect of smoothing on the unit of retrieval. The experiments 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) is applied. 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 value for λ gives us an article biased run, whereas a low value for λ introduces a bias toward smaller elements (such as sections and paragraphs).

Prior probabilities

The second major task is to estimate the prior probability of an element. Basing the prior probability of a retrieval component on its length, has proved useful for several retrieval tasks [3, 9]. Length priors are particularly useful for XML retrieval. It is most common to have the prior probability of a component proportional to the length of the component. That is, we calculate a so-called length prior:

$$\mathrm{lp}(E) = \log\left(\sum_t \mathrm{tf}(t,E)\right). \qquad (4)$$

With this length prior, the actual scoring formula then becomes the sum of the length prior and the score for the query probability,

$$s_{\mathrm{lp}}(E,Q) = \mathrm{lp}(E) + s(E,Q). \qquad (5)$$

Although not used here, results have indicated that it might be useful to have the prior proportional to the square or even the cube of the element length [6]. For an exact description of how we apply this length prior, see the individual run descriptions in Section 3.

Mixing evidence

Although we retrieve individual elements from the collection, the elements are not independent from the surrounding elements. It is therefore intuitive to judge elements, not only based on their own merit, but also based on the context in which they appear. In many of our runs we scored elements by mixing evidence from the element itself, $s(E,Q)$, and evidence from the surrounding article $s(A,Q)$, using the scoring formula

$$s_{comb}(E,Q) = \mathrm{lp}(E) + \alpha \cdot s(A,Q) + (1-\alpha)\cdot s(E,Q), \qquad (6)$$

where $s(\cdot,\cdot)$ is the score function from Equation 3 and $\mathrm{lp}(\cdot)$ is the length prior from Equation 4. This mixing could in principle be more cleanly implemented inside the language model framework, using a mixture model.

Index cut-off

Using a length prior and tweaking of the smoothing parameter are not the only methods 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. We also recalculate collection statistics accordingly, making the run equivalent to Further details will be provided in the description of individual runs in the next section.

3. RUNS

3.1 Content-Only task

In [6] 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 wanted to experiment with element length bias. Three length related parameters were introduced in the previous section: value of the smoothing parameter, length prior and index cut-off. All our runs used the normal length prior, formula (4). Cut-off value was set to 20, which is equivalent to having only indexed elements containing at least 20 terms. Our runs differed only in the value given to the smoothing parameter.

```
<?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: Example of a Content-Only topic (Topic 103)

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 mixed evidence from the article and element level. The same combination value $\alpha = 0.4$ in the scoring formula (6), a value chosen after experimenting with 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*). Feedback is a risky business, some terms might help while other might lead the retrieval astray. For this particular query one can imagine that it is useful to include the founding fathers of UML: *Booch*, *Jacobson* and *Rumbaugh*; but it might be misleading to include the publishers: *Longman*, *(John) Wiley (&) sons* and *(Springer) Verlag*.

3.2 Strict Content-And-Structure task

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 [12]. 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

```
<?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: Example of a Content-and-Structure topic (Topic 76)

to cast light on how these expressions and filters could be used to assign scores to elements.

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

$$\texttt{rootPath}[F_r \cup C_r \cup S_r]\texttt{targetPath}[F_e \cup C_e \cup S_e], \qquad (7)$$

where `rootPath` and `targetPath` are XPath path-expressions and F_r, C_r, S_r, F_e, C_e, S_e are sets of filters (to be explained below). 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.

The filters in the actual topics were connected with a boolean formula. We ignore this formula and only look at sets of filters. However we treat the filters in quite a strict fashion; the larger 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.

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

$$\texttt{rootPath} = \texttt{//article}$$
$$F_r = \{\texttt{about(.,'"intelligent transportation system"')}\}$$
$$C_r = \emptyset$$
$$S_r = \{\texttt{./fm//yr='2000',./fm//yr='1999'}\}$$
$$\texttt{targetPath} = \texttt{//sec}$$
$$F_e = \{\texttt{about(.,'automation +vehicle')}$$
$$C_e = \emptyset$$
$$S_e = \emptyset$$

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, from an element retrieval run, all elements not having the tag-name $\langle$**`article`**$\rangle$. 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 we used the scoring formula with a length prior (Equation 5). 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. For all the runs, the elements are filtered using an XPath-parser and the strict filters (Figure 3*e*). Any filtering using tag-names used the tag equivalence relations defined in the topic development guidelines. Our three different runs we created as follows.

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 fuzzy XPath expression (Figure 3*d*), 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

$$\begin{aligned} RSV(e) = \alpha \cdot \left(s(r) + \sum_{f \in F_r} s(f) + \sum_{c \in C_r} \max s(c) \right) \\ + (1-\alpha) \cdot \left(s(e) + \sum_{f \in F_e} s(f) + \sum_{c \in C_e} \max s(c) \right), \quad (8) \end{aligned}$$

where F_r, C_r, F_e and C_e represent sets of elements passing the respective filter mentioned in Equation 7; $s(r)$ is the score of the article from the article run; $s(f)$ and $s(c)$ are scores from the filter 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.

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 [8]. 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, scores were calculated using the length prior (formula 5).

4. RESULTS AND DISCUSSION

We evaluate our runs using version 2003.004 of the evaluation software provided by the INEX 2003 organizers. We used version 2.4 of the assessments. Below, all runs are 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 are evaluated using *inex_eval*, the standard precision-recall measure for INEX. At present, two other measures are being 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 of writing, 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.

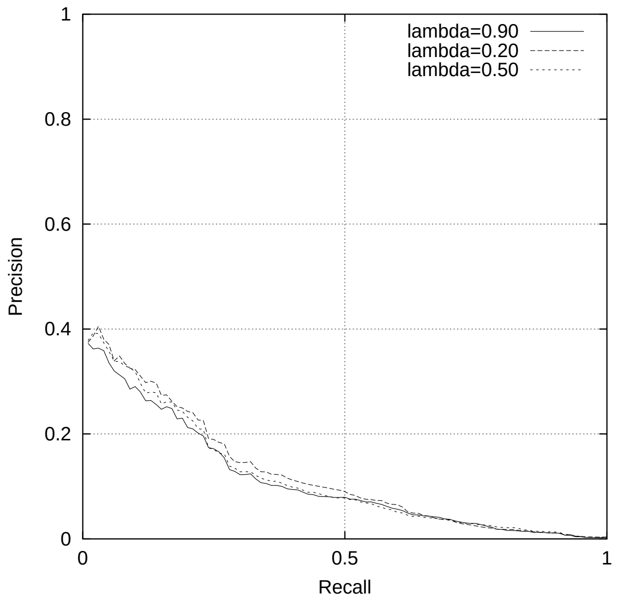

Figure 4: Precision-recall curves for our CO submissions, using the strict evaluation measure

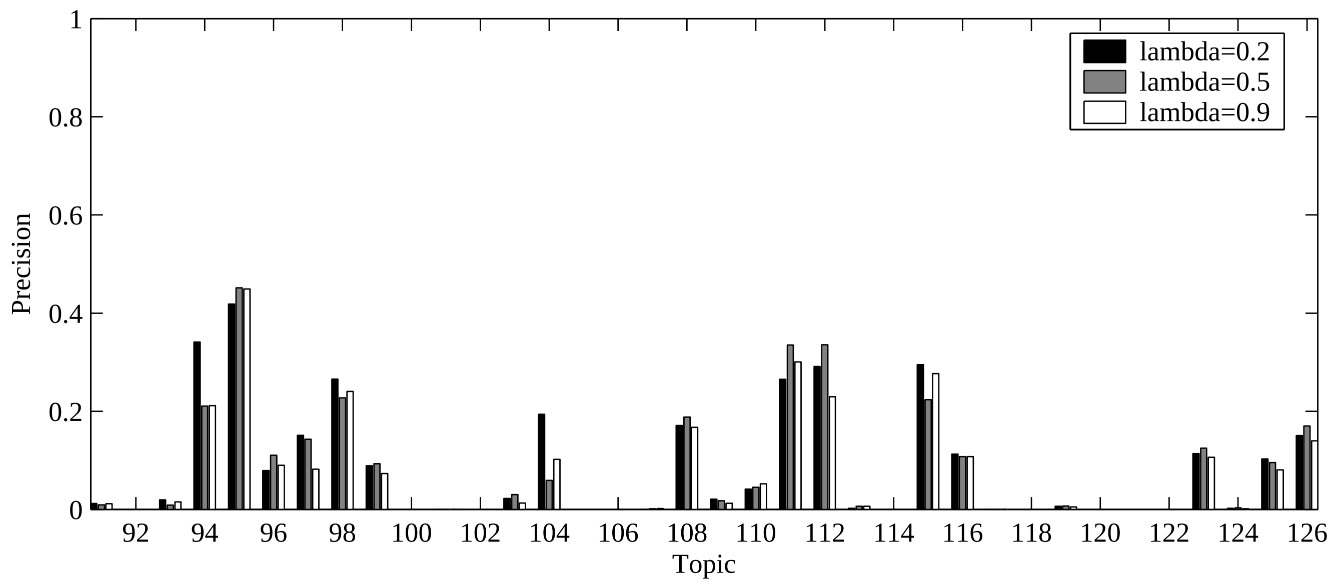

Figure 5: Precision for each of the CO topics. Note that assessments for topics 105, 106, 114, 118, 120, and 122 have not been completed. Furthermore, topics 92, 100, 102, and 121 have no strict judgments.

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

Table 1: Results of the CO task

According to the inex_eval measure, the run using $\lambda = 0.2$ has over all highest MAP score. The run that uses $\lambda = 0.5$ and filters 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.

Despite the similarity between the runs, let's take a closer look and see if there is any difference. Table 2 shows, for each run, the average length of retrieved elements and average length of the relevant elements retrieved. The table shows that the runs are indeed different. We are using the smoothing parameter to introduce a different length bias, the higher the value we give to the length prior, the larger elements we get on average. The difference between average length of retrieved elements and the average length of relevant elements retrieved, might indicate that a more length biased length prior is needed. Figure 5 shows the average precision of our runs for each topic separately. We see that for a vast majority of the topics the different runs give more or less the same score.

	Average element length	
	retrieved	relevant
$\lambda = 0.2$	1,335	2,499
$\lambda = 0.5$	1,839	2,965
$\lambda = 0.9$	2,166	3,330

Table 2: Some statistics of our submitted runs

From Figure 5 we see that our runs are far from being stable between topics. For 15 out of 30 assessed topics we score practically nothing at all. For 9 topics our score lies between 0.05 and 0.2. For 5 topics we score between 0.2 and 0.4. Finally only one topic reaches over 0.4. Let's take a closer look at the 15 topics where we score practically nothing. For 4 of them there were no strict judgments, i.e. no element was assessed as highly exhaustive and highly specific. A further 7 topics had 10 or less strict judgments. The remaining 4 had 21–90 strict judgments each. For all the 11 topics where were 10 or fewer strict judgments, we score poorly. For those topics the task turned out to be a real needle-in-the-haystack problem.

4.2 Strict Content-And-Structure task

In this section we will refer to our thee different runs as element-based, document-based and mixed. Table 3 shows the results of the SCAS runs. Figure 6 shows the precision-recall plots. The mixed run has higher MAP than the other two runs. The element-based run has slightly lower MAP than the mixed run. The document-based run has the lowest MAP.

	MAP	p@5	p@10	p@20
ElementScore	0.2987	**0.4160**	**0.3520**	0.2540
DocumentScore	0.2314	0.2960	0.2680	0.2160
MixedScore	**0.3182**	0.4000	0.3440	**0.2860**

Table 3: Results of the SCAS task

The element-based run 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 mixed run catches up with the element-based run once 20 documents have been retrieved. This indicates that coordination level matching for the filter matching, works well for initial precision, but is not as useful at higher recall levels.

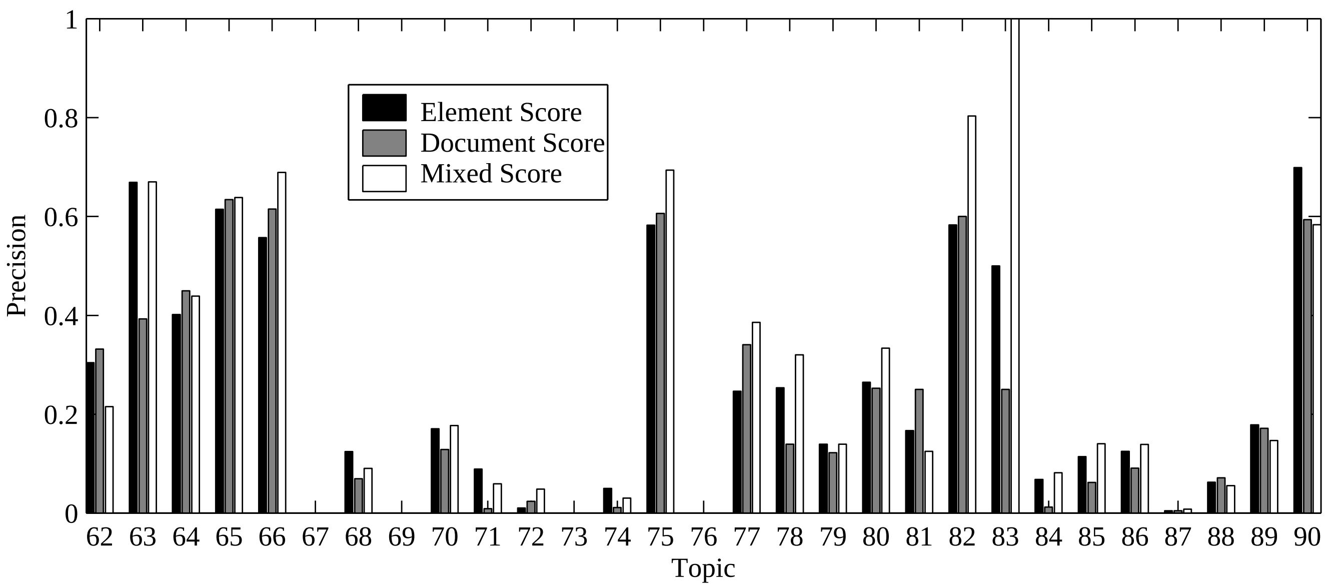

Figure 7: Precision for each of the SCAS topics. Topics 61, 67, 69, 73, and 76 have no strict judgments.

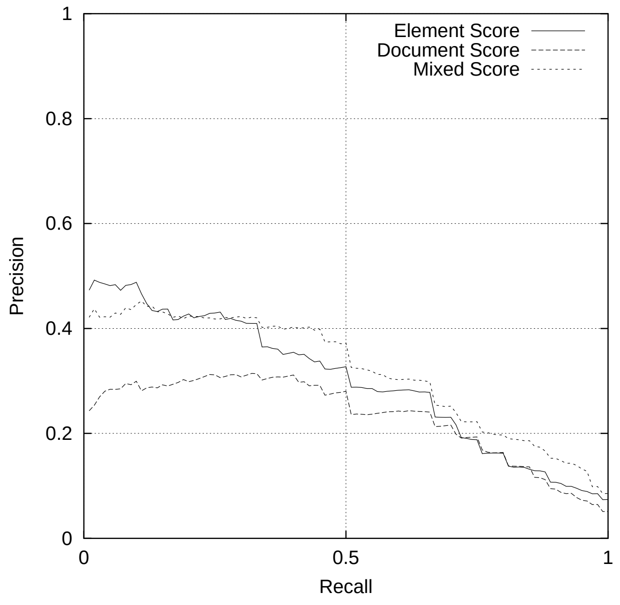

Figure 6: Precision-recall curves for our SCAS submissions, using the strict evaluation

Let us now try to analyze individual topics and topic groups. Figure 7 shows the average precision for our SCAS runs, individually for each topic. We see that the our performance is topic dependent. For this task, we do not see as clear correlation between precision and total number of relevant elements, as we saw for the content-only task. Since the target element is usually specified, this is less of a needle-in-the-haystack problem. To try to understand this better we look at performance over three different classes of topics.

Table 4 shows mean average precision for three different classes of target elements. First we look at the class of topics where the target is ⟨`article`⟩, then we look at the class where the target is ⟨`sec`⟩, and finally we look at the class of other topics (where the target is either *, ⟨`abs`⟩, ⟨`p`⟩, ⟨`vt`⟩ or ⟨`bb`⟩). The second column in the table shows how many topics there are in each class. The remaining columns show the performance of each run. The difference of each run is calculated using the overall performance of that run as baseline. Before we continue it must be said that the results must be taken with a grain of salt; they are based on very few topics, the classes only contain 10, 8 and 7 topics respectively.

Target	#	elem.-based		doc.-based		mixed	
article	10	0.3298	+10%	0.3142	+36%	0.3526	+11%
sec	8	0.2354	-21%	0.2364	+2.2%	0.2810	-13%
other	7	0.2569	-14%	0.1712	-26%	0.3199	+0.53%

Table 4: Average precision of our runs for the SCAS topics, clustered by tag name of the target element

For the class of topics where the target is an article, all runs perform well relative their overall performance. Compared to each other, the element-based run and document-based run perform similarly. The only difference is the value chosen for the smoothing parameter λ. For this class, the mixed run scores better than the other two runs, giving further evidence of how structure can help improve article retrieval [11].

For the class of topics where sections are the target, the performance of the document-based run is similar to it's overall performance. The element-based run and the mixed run perform poorly relative to their overall performance. Compared to each other, the mixed run still performs somewhat better than the other two runs. Again there is not much difference between the element-based run and the document-based run. This is surprising since one would have guessed that the element-based run would perform better.

For the class of the remaining topics, the performance of the mixed run is similar to it's overall performance. The other two runs perform poorly relative to their overall performance. Compared to each other, the mixed run is still better than the other two. Now the element-based run is clearly better than the document-based run.

Overall we can say safely that, our runs perform better on topics where the target element is an article, compared to the performance for other target-type classes. When the different runs are compared to each other, the mixed run performed consistently better than the other two. The element-based run only differentiated itself from the document-based run when the task was to find the smaller elements such as paragraphs and abstracts.

4.3 Vague Content-And-Structure task

At the time of writing the evaluation metric of the Vague Content-And-Structure task had not been released. Hence there are no results to discuss for this task.

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 [6]. 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.

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 be the best estimator for the language models. In our current research we investigate the various statistics from which the language models can be estimated.

6. ACKNOWLEDGMENTS

Jaap Kamps was supported by the Netherlands Organization for Scientific Research (NWO) under project numbers 400-20-036 and 612.066.302. 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, 612.000.207, and 612.066.302.

7. 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] D. Hiemstra and W. Kraaij. Twenty-One at TREC-7: Ad-hoc and cross-language track. In E.M. Voorhees and D.K. Harman, editors, *The Seventh Text REtrieval Conference (TREC-7)*, pages 227–238. National Institute for Standards and Technology. NIST Special Publication 500-242, 1999.

[4] J. Kamps, 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. 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.

[6] 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.

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

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

[9] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Proceedings of the 19th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 21–29. ACM Press, 1996.

[10] Snowball. The snowball string processing language, 2004. `http://snowball.tartarus.org/`.

[11] R. Wilkinson. Effective retrieval of structured documents. In W. Bruce Croft and C. J. van Rijsbergen, editors, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 311–317. Springer-Verlag New York, Inc., 1994.

[12] XPath. XML Path Language, 1999. `http://www.w3.org/TR/xpath`.

[13] 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.