Processing Content-And-Structure Queries for XML Retrieval

Börkur Sigurbjörnsson
borkur@science.uva.nl

Jaap Kamps*
kamps@science.uva.nl

Maarten de Rijke
mdr@science.uva.nl

Informatics Institute, University of Amsterdam
Kruislaan 403, 1098SJ Amsterdam, The Netherlands

ABSTRACT

Document-centric XML collections contain text-rich documents, marked up with XML tags. The tags add lightweight semantics to the text. Querying such collections calls for a hybrid query language: the text-rich nature of the documents suggest a content-oriented (IR) approach, while the mark-up allows users to add structural constraints to their IR queries. We propose an approach to such hybrid content-and-structure queries that decomposes a query into multiple content-only queries whose results are then combined in ways determined by the structural constraints of the original query. We report on ongoing work and present preliminary evaluation results, based on the INEX 2003 test set.

1. INTRODUCTION

Document-centric XML documents contain text, marked up with XML tags, enriching the text with lightweight semantics. The markup can be exploited in several ways. Retrieval engines can use specific tags to try to boost retrieval effectiveness, as illustrated by the effectiveness of using anchor text in web retrieval [1]. Alternatively, if users are aware of the structure they can query the collection by means of so-called content-and-structure (CAS) queries. CAS queries allow users to express their information need very precisely, through constraints on both the content and the structure of desired XML elements. The queries provide constraints both on the *granularity* of results (i.e., the requested unit of retrieval), and on the *content* of the results. Furthermore, they constrain the *environment* in which the results appear.

The Initiative for the Evaluation of XML Retrieval (INEX) provides a test-bed for evaluating XML retrieval using content-and-structure queries. Whereas currently emerging standards for querying XML, such as XPath and XQuery, can be very effective for querying structure, another approach may be needed for querying content. At INEX, a free text search functionality is added to XPath. Unlike XPath, queries in the INEX query language do not have a definite semantics. Evaluating the quality of a system's response to such queries follows standard information retrieval methodology and is based on human made assessments of relevance.

*Currently at Archives and Information Studies, Faculty of Humanities, University of Amsterdam.

TDM'04, the first Twente Data Management Workshop on XML Databases and Information Retrieval, Enschede, The Netherlands
© 2004 the author/owner

In this paper we focus only on elements fulfilling the granularity constraint, and investigate ways of ranking them w.r.t. how well they answer the information need expressed in the query. To answer content-and-structure queries we use existing information retrieval technology. However, existing retrieval systems are not directly applicable since they are usually not equipped with tools for handling structural constraints. We will outline how we are working on extending our information retrieval system to handle content and structure queries. We break the process up into three steps.

- We decompose a content and structure query into a number of IR queries, each of which constrains different parts of the document. We look at these document parts as different sources of evidence.
- We use our information retrieval system to process the IR queries independently. Here we are collecting evidence from different parts of the document.
- We mix the evidence from the multiple sources.

This paper describes work in progress. Partly based on our top scoring runs at INEX 2003 [11], we decided to investigate in detail all the sources of evidence that we can obtain from hybrid content-and-structure queries. In this paper we will report on some preliminary results of using our system to answer such CAS queries.

The paper is organised as follows. Section 2 introduces document-centric XML collections and the information retrieval challenges for such collections. Furthermore, we discuss content-oriented XPath, the query format used at INEX to formulate content-and-structure queries. We will also sketch how a retrieval engine could answer such queries. In Section 3 we explain in more detail how we have extended our own retrieval engine to handle content-oriented XPath queries. Section 4 describes a set of preliminary experiments where we compare several retrieval strategies. The results of those experiments are also discussed. We discuss future work in Section 5. In Section 6 we conclude..

2. CONTENT AND STRUCTURE

XML can be used to mark up content in various ways. Based on the content, XML documents are often broken down into two categories: *data-centric* and *document-centric*. Document-centric documents are loosely structured documents (often text), marked-up with XML. An example of document-centric XML is an electronic journal in XML.

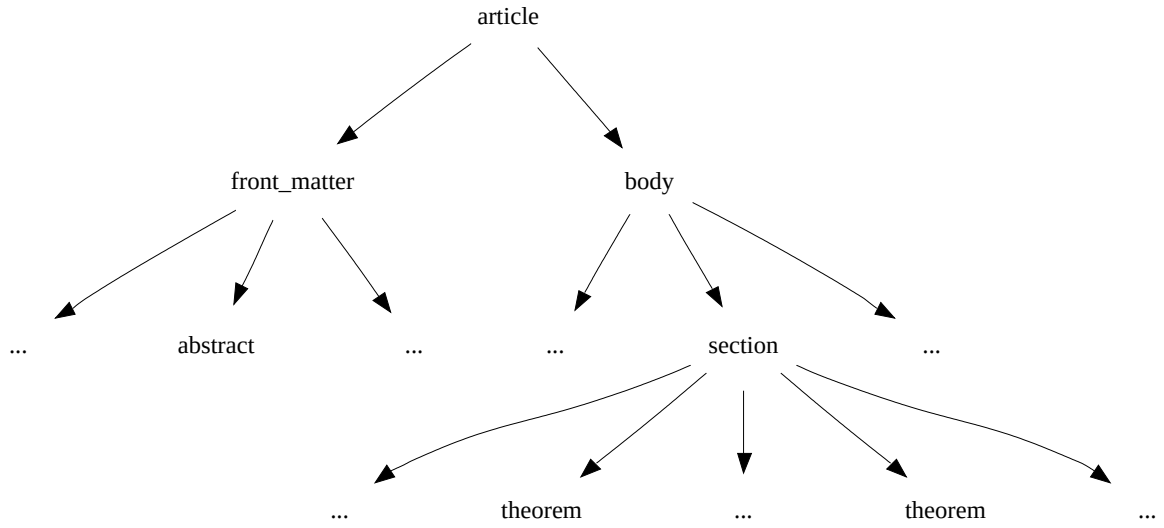

Figure 1: Example XML document

Document-centric XML is sometimes referred to as *narrative* XML, reflecting the importance of the order in which XML elements occur.

For our experiments we use the document-centric XML collection that comes with the INEX test-suite [7]. It contains over 12,000 computer science articles from 21 IEEE Computer Society journals. The documents are marked up with XML tags. On average a document contains 1532 elements and the average element depth is 6.9. The markup has around 170 tag names, such as articles ⟨article⟩, sections ⟨sec⟩, author names ⟨au⟩, affiliations ⟨aff⟩, etc. Figure 1 shows an example of the structure of an XML document, similar to those present in the INEX collection. The tag-names mostly explain the layout of the articles but give little information about their content.

To query document-centric XML documents we can use a hybrid query language, in which content and structural requirements can be expressed and mixed. At INEX, an XPath-like query language has been introduced which is appropriate for XML information retrieval. The syntax of the language looks like XPath, but does not have the same strict semantics. It can be seen as an *extension* of a *subset* of XPath. The subset used at INEX 2003 was simple, yet INEX participants had great difficulty in formulating correct queries in this language (see [10] for details). Only the child- and descendant-axis were used and the maximum number of predicates allowed was two. Based on user studies, a smaller subset has been chosen for INEX 2004 [12].

At INEX, XPath is extended with the `about` function, aimed at facilitating free-text search. Although `about` function has the same syntax as the XPath function `contains`, their semantics are radically different. Because of its strict, boolean character, the `contains` function is not suitable for querying text rich documents. The semantics of the `about` function is meant to be very liberal. It is meant to be used to rank elements according to their relevance for a given content requirement. Consider the XML element

`<affiliation>Stanford University</affiliation>`.

A human assessor is likely to decide that the function

`about(.//affiliation,'California')`

should return true on this string; in contrast, an XPath processor equipped only with `contains` would have difficulties trying to do the same.

The content-and-structure query language allows users to describe their desired results more precisely. The structural requirements used in queries are of course limited by the structural properties of the collection. The query format working group at INEX 2003 expressed some concerns that the structure of the INEX collection did not give many natural opportunities to exploit structure [12]. The working group did however identify some natural structural requirements, such as the co-occurrence of concepts in an element, and references to author names or affiliations.

Example. Suppose a user is interested in information about the safety of collision detection algorithms of flight traffic control systems. She formulates her information need in a mixture of structural and content constraints:

```
//article[about(.//abstract, flight traffic control
system)]//section[about(., collision detection
algorithm) and about(.//theorem, safety)]
```

She believes that articles that are really focused on "flight traffic control systems" do say so in their abstract. Note that the abstract itself is not supposed to be returned as an answer to this query. This is a constraint on the *environment* of the answer. Since she is interested in a certain aspect of traffic control systems, namely the "collision detection algorithm", she specifies that she wants to zoom in on a section about that particular aspect. Finally, she assumes that a good collision detection algorithm should be proved to be safe. Hence she adds the last `about` function, which says that it is desirable that the sections returned contain a theorem about safety.

From a user perspective the query above should not be taken literally, but interpreted using vague semantics. A user is likely to judge a section relevant, even if it is in an article which has no abstract, as long as it addresses the information need of the user. Since the retrieval engine is made to serve the user, it should try to approach the problem from a user perspective. The search engine should thus look

at the query as a hint about the content and environment of desired results.

We will now sketch how we extended an information retrieval engine to handle the example content-oriented XPath query above. A more detailed description of the extension is in Section 3. The task of the retrieval engine is to rank elements, in this case sections, according to their relevance for the query. We divide this process up into three steps:

- *decomposition*, where we break the query into a number of IR queries, together with sources against which these queries need to asked;
- *retrieval*, where we collect evidence about relevant elements from each source; and
- *mixture*, where we mix the evidence from the multiple sources to provide a ranked list of elements to return to the user.

Decomposition. We start by decomposing the query into a set of `about` functions. Each `about` function is further split into a pair (`location path`, `content description`), which consists of, indeed, a location path and a content description. For our running example, we get

- (`//article//abstract`, `flight traffic control system`)
- (`//article//section`, `collision detection algorithm`)
- (`//article//section//theorem`, `safety`)

Each pair represents a potential source of evidence, where the location path is used to locate the source and the content description is used to collect evidence from the source.

Retrieval. For each `about` function, we use an information retrieval engine to assign scores to elements, that are in the node-set of the *location path*, reflecting how relevant they are to the *content description*. Let's take the first of the `about` functions listed above as an example. We first look at the node-set returned by the XPath location path `//article//abstract`, which returns all abstracts of all articles in the collection. We use a retrieval engine to assign a retrieval score to each of the abstracts, reflecting how relevant they are to the query "flight traffic control system". We do the same for the other `about` functions. Hence we have a ranked list of abstracts, a ranked list of sections, and a ranked list of theorems.

Mixture. Now we must decide which elements to return to the user, and in which order to return them. First, we need to locate the appropriate elements that can be returned. We use the XPath location path `//article//section` to this end; its node-set contains all sections of all articles in the collection. We refer to those elements as *target elements*. Next, we assign a score to each of the sections. This is done based on the three `about` functions. Let's start with the middle `about`. The sections get a score reflecting to which extent they themselves are relevant to the query "collision detection algorithm". Now let's look at the first `about` function. The score of the sections is increased if they are contained in articles that have an abstract which is relevant to the query "flight traffic control system". The increase in score depends on how relevant the abstract was to the query. Finally, we look at the third `about` function. The score of a section is increased if it contains a theorem which is relevant to the query "safety".

Although the retrieval approach has been sketched, there are still some technical issues to be resolved. Should we normalise scores from the different sources? If so, how? We seem to have three different types of sources of evidence: from the target itself, from "above", and from "below". Should those sources have equal weight in the combination? What if there are multiple relevant theorems inside a section? Should they all contribute? While we don't have final answers, we will address those issues in the next section.

3. PROCESSING CONTENT AND STRUCTURE QUERIES

3.1 Decomposition

We will now describe, in a more formal manner, how we decompose a given content and structure query into a number of pairs consisting of a location path and content-only query.

For each XPath query q_{xpath} we define a set of `about` functions $A(q_{\text{xpath}})$. Each `about` function, a, is a pair consisting of a location path p_a and a natural language query q_a. We denote by $E(p_a)$ the node-set that can be located via the location path. We think of this node-set as our source of relevance for that particular `about` function. We will then use the natural language query q_a to collect actual evidence of relevance from that source. More precisely, we will use a traditional IR system to rank the elements of the node-set, w.r.t. how well they fulfill the information need expressed in the natural language query.

3.2 Retrieval

Each of the pairs defined above can be used to locate sources of relevancy and to collect evidence from those sources, using an information retrieval system. In this paragraph we describe the retrieval system used in the latter step and discuss which parameters might influence our results.

Indexing. Since we are interested in information needs that combine structural and content aspects, we index both the text and the XML structure of the full INEX collection.

Inverted indexes are efficient for testing whether a term occurs in a document or element [14]. We build an inverted *element index*, a mapping from words to the elements containing the word. Each XML element is indexed separately. That is, for each element, all text nested inside it is indexed. Hence, the indexing units overlap. Text appearing in a nested XML element is not only indexed as part of that element, but also as part of all its ancestor elements. To index the XML trees we use pre-order and post-order information of the nodes in the XML trees [4].

Retrieval model. For the ranking of elements, our retrieval engine uses a multinomial language model with Jelinek-Mercer smoothing [5]. 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, $P(e)$. Note that since we use our retrieval engine to rank each `about` function separately, our queries are natural language strings or a list of query terms.

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

$$P(q|e) = \prod_{i=1}^{k} \left(\lambda \cdot P_{\text{mle}}(t_i|e) + (1-\lambda) \cdot P_{\text{mle}}(t_i|c)\right), \quad (2)$$

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 (often called the *smoothing parameter*). We estimate the language models, $P_{\text{mle}}(\cdot|\cdot)$ using maximum likelihood estimation. For the collection model we use element frequencies (i.e., in how many elements does a given term occur). The estimation of this probability can be reduced to the scoring function, $s(q, e)$, for an element e and a query $Q = (t_1, \ldots, t_k)$,

$$s(e,q) = \sum_{i=1}^{k} \log\left(1 + \frac{\lambda \cdot \text{tf}(t_i,e) \cdot \left(\sum_t \text{df}(t)\right)}{(1-\lambda) \cdot \text{df}(t_i) \cdot \left(\sum_t \text{tf}(t,e)\right)}\right). \quad (3)$$

Here, $\text{tf}(t, e)$ is the frequency of term t in element e. The $\text{df}(t)$ is the element frequency of term t, that is, the number of indexed elements in which t occurs. The λ is the smoothing parameter. In the experiments in this paper we keep the smoothing parameter fixed at a value 0.2.

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 [6, 13]. It is most common to have the prior probability of a component proportional to its length. That is, we calculate a so-called length prior:

$$\text{lp}(e) = \log\left(\sum_t \text{tf}(t,e)\right). \quad (4)$$

With this length prior, the actual scoring formula becomes the sum of the length prior (Equation 4) and the score for the query probability (Equation 3),

$$s_{\text{lp}}(e,q) = \text{lp}(e) + s(e,q). \quad (5)$$

3.3 Mixture

At this stage, we have decomposed the query and identified potential sources of relevance. We have used an information retrieval engine to collect evidence from those sources. Now it is time to put things together.

We look at the target elements, i.e., the elements returned by the target location path. For each target element e, we need to estimate how relevant it is to the content-oriented XPath query q_{xpath}.

We have a set of `about` functions, $A(q_{\text{xpath}})$, from Section 3.1. In Section 3.2, we calculate scores for each `about` function separately. First, we take an `about` function $a \in A(q_{\text{xpath}})$ and a target element e and we need to calculate the score of e for that particular `about` function. Let us now define $E(p_a)$ to be the node-set of the location path of a. We now define a function χ_a which connects the elements e' of node set $E(a)$ to our target elements e:

$$\chi(e,e') = \begin{cases} 1 & \text{if } e \text{ and } e' \text{ are connected by } q_{\text{xpath}} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

The notion of *connection* between two elements will not be explained further here but we refer to the W3C XPath semantics [15]. In our running query example we would say that for the `about` function for theorems, a target section element is only connected to its descendant theorem elements. Similarly, for the `about` function for abstracts, a target section is connected to all abstracts that are contained within the same article. Finally, for the remaining `about` function, a target section is only connected to itself.

Now we can use the function χ_a to define the score of a target element e w.r.t. an `about` function a:

$$s_{\text{about}}(e,a) = \max_{e' \in E(a)} \chi_a(e,e') \cdot s_{\text{about}}(e', q_a) \quad (7)$$

where q_a is the content description of a. We can calculate $s(e', q_a)$ using Equation 3 since q_a is a natural language query. Alternatively, we could incorporate a length prior in the score by using Equation 5. Perhaps some other prior is more helpful.

When there are multiple elements e' that are related to the target element e we choose to let only the highest ranking element e' contribute to the score of e (the max function). In terms of the example before, if a section has multiple relevant theorems, only the most relevant one contributes to the scoring of the section. Alternatively, we could have used an average or a sum instead of the maximum.

Now that we have for each target element, a score for each `about` function, we need to combine it into one final score which measures the relevance of the target element to the XPath query q_{xpath}. We simply assign a score to an element by summing up its scores for each `about` function:

$$s_{\text{xpath}}(e, q_{\text{xpath}}) = \sum_{a \in A(q_{\text{xpath}})} \alpha_a \cdot s(e,a), \quad (8)$$

where α_a is a parameter for fixing the weight that the `about` function a has in the total score of target element e. In its simplest form the scoring formula would use the value 1 for all about functions.

What this means in terms of the example topic and the Figure 1 is that the section is assigned a score according to its own relevancy, according to the relevancy of the article abstract, and the relevancy of its most relevant theorem.

In the actual queries, `about` functions could in principle be connected in a predicate using the logical operators 'AND' and 'OR'. We do not use these additional requirements in our computation. Our approach works with a *set* of `about` functions. We are liberal in the sense that we score elements even if only one of the functions returns a score. We can thus say that we treat all the queries as if they only used 'OR'. However, we do sum scores over all functions. Thus, the more functions that return a positive score, the higher the total score. Thus, we can say that we have a bias toward treating the queries as if they only used 'AND'. This will,

however, not result in coordination level matching for the set of `about` functions.

4. EXPERIMENTS AND RESULTS

The extension of our information retrieval system, as suggested in Section 3, is quite involved. If we want to convince ourselves that this extension is worth the effort, a minimal requirement is that it outperforms some simpler extensions.

We create runs using the 30 INEX 2003 CAS queries. We evaluate our runs against version 2.5 of the INEX assessments. We will only look at a strict interpretation of the assessments. That is, an element in relevant if, and only if, it is highly exhaustive and highly specific (see [3, page 204]). We will discuss our results w.r.t. two evaluation measures. We will look at *mean average precision* (MAP), which measures how well systems return only relevant elements. We also look at *recall*, which measures how many of the relevant elements are found by systems. The measures are applied on the top 1000 elements returned by our system.

4.1 Baseline runs

The minimal requirement for a system that claims to use structural requirements effectively, is that it outperforms a system that does not use the structural requirements at all. We will compare our extended system to two simple baselines, both created using the non-extended version of our system and content-only queries. As queries we use the concatenation of terms appearing in the `about`-functions of the content and structure queries. For our running example we would use the query:

```
flight traffic control system collision detection
algorithm safety
```

The only structural requirement we use is the granularity constraint. That is, for our running example we would only rank elements appearing in the node-set of the XPath expression `//article//section`.

Here we take into account the sets of equivalent tag names as defined in [3, page 197]. These sets define, for example, the set of tagnames used to mark up sections in the INEX collection. In earlier work, such as the related set of runs reported in [9], we did not consider these tag name equivalences, resulting in lower scores than those reported here.

Document-based run. For our first baseline run, we use a document retrieval system to rank the elements. That is, from the top ranking documents, we collect all elements that match the granularity constraint. One can view this as answering the query

```
//article[about(., query)]//section
```

This is probably the simplest extension of a traditional retrieval system to answer content and structure queries.

Element-based run. Our second baseline run uses our element based retrieval system. We rank the elements using the score they get when retrieving from the element index. One can view this as answering the query

```
//article//section[about(., query)]
```

This is probably the simplest element-based extension of an information retrieval system.

Run	MAP		Recall
Document-based run	0.2465		0.7268
Element-based run	0.3209	+30.2%*	0.7153

Table 1: Results for the baseline runs. The improvement of the element-based run is calculated relative to the document-based run.

Run	MAP		Recall
Environment-based run	0.3219	+0.3%	0.7067
combSUM-doc-ele-env	0.3627	+13.0%	0.7854

Table 2: Results when using structure, improvement is measured over the element-based run

The results of our baseline runs can be seen in Table 1. Not surprisingly, the element-based run has higher mean average precision than the document based one. It is interesting to note, however, that the two runs have similar recall scores. A first glance at these results suggests that the baselines perform quite well and will be hard to beat.

4.2 Using structure

In our baseline runs, the granularity constraint was the only structural constraint we considered. We will now look at how well our extended system works.

Environment-based run. We now create a run using the system we described in Section 3. That is, we do consider the structural constraints expressed by the users. Through the structural constraints the user implicitly adds content constraints both on the target element itself, and on the surrounding elements (the environment in which the target elements reside). We will refer to this run as the *environment-based* run since it scores elements not only by looking at their own content but also the content of their environment (surrounding elements) as far as these are spelled out in the query. The result can be seen in the first row of Table 2. In terms of the mean average precision, there is only a slight improvement over the element-based run. If we look at the precision-recall plots (Figure 2) we see that the environment-based run does result in better initial precision.

combSUM-doc-el-env. The environment-based run does not improve significantly over the element-based run, in terms of mean average precision. It does, however, improve initial precision, indicating that considering the structure has some positive effect. Earlier, it was shown that the combination of element score and article score can lead to improved retrieval performance [11], both for content-only and for content-and-structure queries. This prompts us to combine the three runs: the document-based run, the element-based run and the environment-based run. For the combination we use the combSUM algorithm [2]. Results can be seen in the second row of Table 2. There is quite an improvement over the underlying runs.

4.3 Discussion

Figure 2 plots the precision-recall graph for the two baseline runs, and the two structure runs.

The fact that our environment-based run has higher initial precision than the baselines, seems to indicate that users can indeed express their information need more precisely using

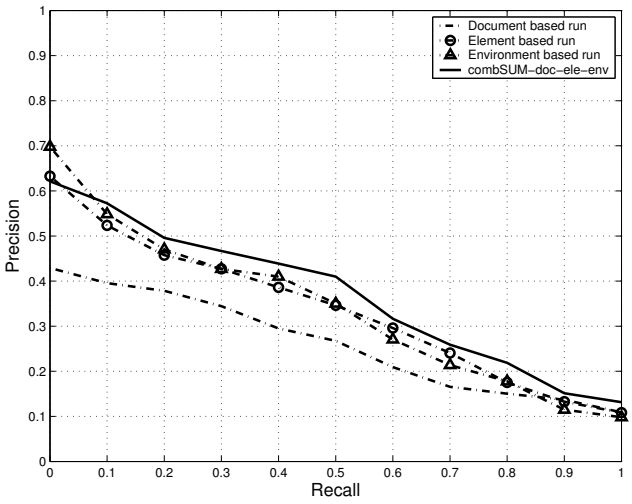

Figure 2: Precision-recall graphs for the baseline and structure runs

a mixture of content and structural requirements.

The improved initial precision of the environment-based run and the success of the three-way combination of runs seem to indicate that the mixing of evidence from multiple sources can improve retrieval effectiveness. Further investigation is needed to discover the importance of the contribution of the individual runs.

We need a larger set of topics to be able to properly test our method. The 30 INEX 2003 topics are not sufficiently many to get a reliable comparison of methods. The structural requirements add a new dimension to the topics. There is thus a possibility of more diversity in topics and hence it will be more difficult to find one silver bullet that works for all topics. We look forward to expanding our test set with the queries of INEX 2004. We believe that both users and systems need time to understand the possibilities that can be exploited using this query language.

5. FUTURE WORK

Since we have only recently started experimenting with our method, we have quite a bit of future work to do. In this section we will list a few questions we would like to answer for each of our three steps.

Decomposition. We have seen that it is useful to calculate scores of articles and elements using queries which consist of all terms in all `about` functions. We would like to investigate how useful this is precisely. Furthermore, we would like to investigate whether, instead of using all terms, we should only use terms that appear in `about` functions which are applied on a descendant-or-self node-set. In our running example we would then have used the query "collision detection algorithm safety" for our element-based run.

Retrieval. Our overlapping textual element index contains large amounts of redundant information. This data redundancy was extremely useful in our initial extensions of our content-based retrieval system. Now, when we have extended our system with support for both content and structure, we can, and plan to, move to a non-overlapping element index for content. This move will reduce the disk usage and disk access, but will increase the on-line computation needed. We believe that the savings in disk access can outweigh the additional cost of run-time computation.

We plan to play with the parameters in our language model. The value for the smoothing parameter is known to affect initial precision, and also the size of the returned elements [16, 8]. Both effects are usually measured w.r.t. the quality of the retrieved elements. We are, however, using language models to rank answers to each `about` function separately. It is not clear whether all `about` functions should use the same value for the smoothing parameter. It is plausible that `about` functions used to rank target elements need different values than `about` functions used to rank other related elements. While it is probably useful to go for recall when scoring target elements, it might be useful to go for high precision when scoring other elements.

A similar argument might hold for the length prior. Length priors, and extreme values for them, have been shown to be particularly important for XML retrieval [8]. The main contribution of the length prior has been for the content-only XML retrieval task, where the granularity of the result elements is unknown. For that task the main challenge is to bridge the length gap between an average element and an average relevant element. For the task we are evaluating in this paper, the granularity of the result elements is usually specified in the query. We are thus not faced with the same length problem as for the content-only task. Hence, we did not apply any extreme length priors for the CAS task, but use the normal length prior described in Equation 4. We conjecture that the length prior is more useful when scoring target elements than when scoring other elements. It is also plausible that the application of length priors depends on the granularity constraint: retrieval of articles might depend less on length prior settings than, for example, retrieval of paragraphs.

Mixture. The mixture is sensitive to the actual retrieval status values calculated, be it proper probabilities, logs of probabilities, or other scores. The normalization of similarity scores may be essential if the different probability estimations are incompatible. We plan to experiment with various rank-equivalent estimates, and specific normalization procedures. This will also allow us to narrow down how different `about` functions contribute to a successful retrieval result.

There are several methods we could use to choose between multiple relevant elements. Currently we have only used the max function, but we could also use all the elements by averaging or summing over max functions.

Vague content-and-structure. INEX 2003 offered evaluation of retrieval using content-and-structure queries without strict interpretation of the granularity constraint. We have yet to investigate how our methods perform on that task.

6. CONCLUSIONS

Document-centric XML collections contain text-rich documents, marked up with XML tags. Querying such collections calls for a hybrid query language: the text-rich nature of the documents suggests a content-oriented (IR) approach, while the mark-up allows users to add structural constraints to their IR queries. This paper introduced an

architecture for dealing with hybrid content-and-structure queries on top of an existing retrieval engine. We propose a three-way strategy that (i) decomposes a query into multiple content-only queries, (ii) which are issued against a retrieval engine treating each XML element as a document, and (iii) the results are then combined in ways determined by the structural constraints of the original query.

We have shown that ignoring the structure is suprizingly effective; for example, the element-based run outperforms our official runs at INEX 2003 (which ranked 1 and 2). As a result, the baseline for using structure is set quite high. Still, our initial experiments using the extension showed some promising results. Taking structure into account results in higher initial precision for the environment-based run. Moreover, in combination with the document-based and element-based runs, it leads to further improvements of retrieval effectiveness.

7. ACKNOWLEDGMENTS

Jaap Kamps was supported by the Netherlands Organization for Scientific Research (NWO) under project number 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.

8. REFERENCES

[1] N. Craswell, D. Hawking, and S. Robertson. Effective site finding using link anchor information. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 250–257. ACM Press, 2001.

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

[3] N. Fuhr, M. Lalmas, and S. Malik, editors. *INEX 2003 Workshop Proceedings*, 2004.

[4] T. Grust. Accelerating XPath location steps. In *Proc. SIGMOD*, pages 109–120. ACM Press, 2002.

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

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

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

[8] J. Kamps, M. de Rijke, and B. Sigurbjörnsson. Length normalization in XML retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference*, 2004.

[9] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. Best-match querying for document-centric XML. In *Proceedings Seventh International Workshop on the Web and Databases (WebDB 2004)*, pages 55–60, 2004.

[10] R. A. O'Keefe and A. Trotman. The simplest query language that could possibly work. In *INEX 2003 Workshop Proceedings*, pages 167–174, 2004.

[11] B. Sigurbjörnsson, J. Kamps, and M. de Rijke. An element-based approch to XML retrieval. In *INEX 2003 Workshop Proceedings*, pages 19–26, 2004.

[12] B. Sigurbjörnsson and A. Trotman. Queries, INEX 2003 working group report. In *INEX 2003 Workshop Proceedings*, pages 167–170, 2004.

[13] 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.

[14] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes*. Morgan Kaufmann, 1999.

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

[16] 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 ACM SIGIR Conference*, pages 334–342, 2001.