Processing Content-Oriented XPath Queries

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, 1098 SJ Amsterdam, The Netherlands
http://ilps.science.uva.nl/

ABSTRACT

Document-centric XML collections contain text-rich documents, marked up with XML tags that add lightweight semantics to the text. 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. Hybrid queries tend to be more expressive, which should lead—in principle—to better retrieval performance. In practice, the processing of these hybrid queries within an IR systems turns out to be far from trivial, because a delicate balance between structural and content information needs to be sought. We propose an approach to processing 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 evaluate our methods using the INEX 2003 test-suite, and show (1) that effective ways of processing of content-oriented XPath queries are non-trivial, (2) that there are differences in the effectiveness for different topics types, but (3) that with appropriate processing methods retrieval effectiveness can improve.

Categories and Subject Descriptors

H.2 [**Database Management**]: H.2.4 Query processing; H.2.8 Database Applications; H.3 [**Information Storage and Retrieval**]: H.3.1 Content Analysis and Indexing; H.3.3 Information Search and Retrieval; H.3.4 Systems and Software; H.3.7 Digital Libraries

General Terms

Experimentation

Keywords

XML Retrieval, XPath, Content and structure

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, much like anchor text can be used to boost effectiveness in web retrieval [4]. Alternatively, if users are aware of the structure of documents in a collection, they can query the collection by means of so-called content-and-structure (CAS) queries. Such queries allow users to provide constraints both on the *granularity* of results (i.e., the requested unit of retrieval) and on the *content* of the results. Furthermore, they may constrain the *environment* in which the results appear. Thus, CAS queries allow users to express their information need very precisely, through constraints on both the content and the structure of desired XML elements.

Currently emerging standards for querying XML, such as XPath and XQuery, do not seem to be optimally suited for expressing CAS queries. At the Initiative for the Evaluation of XML Retrieval (INEX, [9]), a content-oriented extension (of a subset) of XPath has been proposed, which allows one to mix free text search with structural constraints of the type described above. In this paper we are interested in processing CAS queries formulated in this query language. In particular, we focus on investigating ways of ranking elements that fulfill the granularity constraint w.r.t. how well they answer the information need expressed in the query. How can we use existing information retrieval technology to answer content-and-structure queries? Existing retrieval systems are not directly applicable since they are usually not equipped with tools for handling structural constraints.

The main contribution of this paper is a methodology for extending information retrieval systems to handle CAS queries. Our proposal is to process the queries in three steps:

Decomposition Decompose a CAS query into a number of IR queries, each of which constrains different parts of the document. We look at these document parts as different (potential) sources of evidence.

Retrieval Use existing information retrieval system technology to process the IR queries independently, collecting evidence from different parts of documents.

Mixture Rank the relevant elements by mixing the evidence from the multiple sources.

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

CIKM'04, November 8–13, 2004, Washington, DC, USA.
Copyright 2004 ACM 1-58113-874-1/04/0011 ...$5.00.

In addition to a description of our methodology for handling CAS queries with existing IR systems, we provide a user-oriented evaluation of the methodology, using the 2003 test set made available through INEX. Our main aim here is to evaluate the relative effectiveness of different decompositions of CAS queries, looking at all CAS topics, as well as at particular topic types.

The paper is organized 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 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. We describe a set of user-oriented experiments in Section 4 aimed at assessing the retrieval effectiveness of our methodology. The results of those experiments are discussed in Section 5. In Section 6 we offer conclusions.

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

2.1 INEX Collection

For our experimental evaluation we will use the document-centric XML collection that comes with the INEX test-suite [9]. 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.

2.2 INEX Query Language

To query document-centric XML documents we should use a hybrid query language, where content and structural requirements can be expressed and mixed. Such a content-and-structure query language should allow users to describe their desired results more precisely. 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 viewed as an *extension* of a *subset* of XPath.

At INEX, XPath is extended with the `about` function, aimed at facilitating free-text search. Although `about` has the same syntax as the XPath function `contains`, their semantics are quite different. Because of its strict, boolean character, the `contains` function is not suitable for 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')

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

The subset of XPath used at INEX 2003 was simple, only the child- and descendant-axis were used. In the INEX 2003 test set, none of the queries used more than two predicates. That is, only queries with one predicate:

```
location-path[abouts],
```

and two predicates:

```
location-path[abouts]location-path[abouts].
```

were used. Each `location-path` is an XPath location path using only child-axis, descendant-axis, and node-tests. Each predicate, `[abouts]`, is a boolean combination of `about`-functions. The location path used within the `about`-functions was also limited to child- and descendant-axes. Even this limited fragment of XPath turned out to be too complex for the INEX query creators [16]. Based on user studies, a smaller subset has been chosen for INEX 2004 [20].

2.3 INEX CAS Queries

To make matters more concrete, let us look at an example of a CAS query in the INEX 2003 format; this example will be our running example in the remainder of the paper.

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 the abstract. Since she is interested in a certain aspect of the 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 the desired results.

2.4 Research Questions

Using the content-oriented XPath example query just given we provide an informal description of our method for extending IR systems in order to process CAS queries; a more

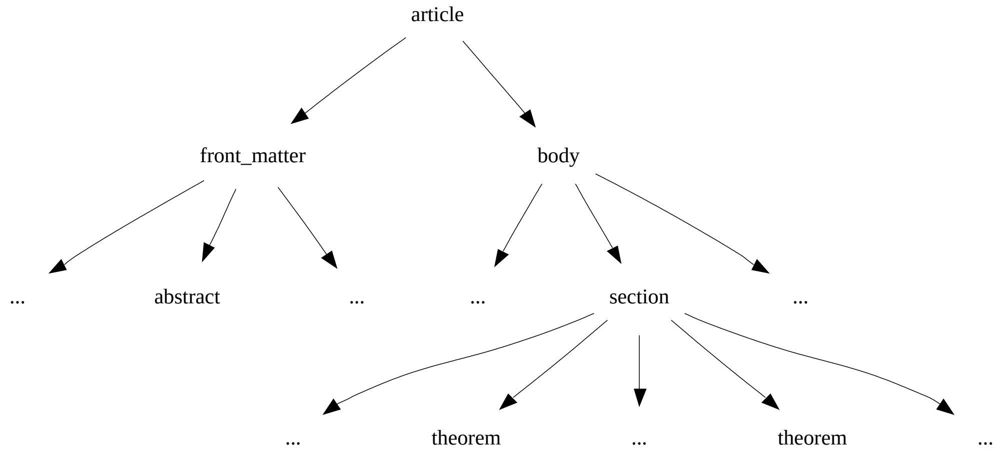

Figure 1: Example XML document

detailed description of the extension is given in Section 3. For our example, 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;
- *retrieval*, where we collect evidence about relevant elements from each source;
- *mixture*, where we mix the evidence from the multiple sources to provide a ranked list of elements to return to the user.

Our formalism opens up many interesting research questions. In each step of our approach there are numerous questions we could try to answer.

The "decomposition" phase is all about breaking the query up to different content-only queries used to collect evidence from different sources. We start by decomposing the query into a set of `about` functions. Each `about` function is further divided up into a location path (the structural part used to locate the source) and a content description (the content part used to collect evidence from the source). For our running example query, we could get evidence from the following three sources: an abstract about "flight traffic control system;" a section about "collision detection algorithm;" and a theorem about "safety."

Additional sources of evidence can be derived more indirectly from the query by merging two or more `about`-functions into one that puts restrictions on their common ancestor. It may also be natural to introduce new `about`-functions by propagating query terms up and down the query. For example, in terms of our running example, we would like to score articles w.r.t. whether they discuss "flight traffic control system" even if they do not have an abstract.

In the "retrieval" phase we have the same questions as for any other retrieval task. We need to try to understand the nature of the task and try to adjust the retrieval parameters accordingly. In our framework we would like to find out whether different sources of evidence require different retrieval settings. Bias towards long XML elements has shown to be useful for unstructured queries (CO queries) [10]. In the case of structured queries (CAS queries), length bias is likely to depend on the length distribution of the elements in the node-set being evaluated (evidence source). As an example, a length bias might be more useful for unspecified target elements than for <`abstract`> elements. Similarly, from some evidence sources high recall might be important while high precision might be more appropriate for others.

Similarly, in the "mixture" phase, should we normalize scores from the different sources? If so, how? We seem to have three different types of evidence sources: 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?

The ability to express structural requirements adds a new dimension to information retrieval topics. The structural properties used in queries can be very diverse. Due to this diversity it might be more difficult to come up with a silver bullet that works for all query types. We will look at two classifications of the INEX CAS topics. The classifications will allow us to gain a better understanding of the effects of different retrieval strategies: What is the relative effectiveness of different processing methods for different classes of topics? Do we need different methods for different topic types? Or are there methods that improve for all topics?

While we do not have final answers for all of the issues raised, issues related to the "decomposition" phase will be addressed in quite some detail below.

3. PROCESSING CONTENT AND STRUCTURE QUERIES

In this section we explain our methodology for processing CAS queries using a standard IR system. As mentioned previously, we divide this process into three steps: *decomposition*, where we transform the original query into a number of IR queries; *retrieval*, where we use the IR queries to collect evidence from various locations of the documents; and *mixture*, where we mix the evidence from the various locations to provide a ranked list of elements to return to the user.

Special attention will be given to different ways of decomposing CAS queries into sequences of traditional IR queries. In the experiments on which we report in Section 4 we evaluate the retrieval effectiveness of different decomposition methods, while we keep the retrieval and mixture processes fixed.

3.1 Decomposition

The aim of the decomposition step in our methodology is to take an input CAS query and rewrite it into a sequence of traditional IR queries. More precisely, we rewrite it into a sequence of pairs consisting of a location path and content-only query. We will now describe several ways of performing this decomposition.

Our first decomposition manner is called *environment-based* decomposition and amounts to a very faithful interpretation of the original CAS query. Specifically, for each XPath query q_{XP} we define a set of `about` functions $A(q_{XP})$. Each `about` function, a, is a pair consisting of a location path p_a and a content description q_a. The content description can be either a natural language description of the information need or a list of query terms representing the information need. For our running example, we decompose the query into three pairs of the form (`location path`, `content description`), one for each `about` function:

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

Each pair represents a *source of evidence*, which is located using the location path; and a *content description*, which is used to collect the evidence from the source. More precisely, we denote by $E(p_a)$ the set of all nodes that can be located via the location path. We think of this node-set as our source of relevance for the `about` function a. We use the natural language query q_a to collect actual evidence of relevance from that source. More precisely, in the "retrieval" phase 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 content description.

What about alternative decomposition methods? Recall that in ad-hoc IR, content-oriented queries are not taken literally, but as an expression of an implicit underlying information need. Users do not have intimate knowledge of the content and structure of the documents that satisfy their information need. As a result, they may find it difficult to formulate exact queries for their information need [2]. The INEX CAS assessments are done mainly with respect to whether the retrieved content fulfills an underlying information need [12]. Hence, it may be argued that structural queries and content-only queries should be interpreted in the same non-strict manner, because users need only have a rather vague idea about the structure of the elements that answer their information need. For our running example, our user finds it plausible that relevant sections appear in articles with an abstract about flight traffic control system. However, she would probably judge a section relevant if it answers her underlying information need, even if it appears in an article which has no abstract. Hence, as with ad-hoc IR queries, CAS queries should thus not be taken literally, and non-strict interpretations should be considered for both structure and content constraints.

The extent to which one adopts a non-strict interpretation of the structural constraints in CAS queries may vary. One extreme point of view is to ignore the structural requirements altogether. We will create a decomposition which we refer to as the *content-only* decomposition of our queries. That is, we form a content-only query by collecting all the query terms appearing in any of the `about`-functions. We refer to this content-only query as the *full content query*. For our running example we would create 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`. We rank those elements using the score they would get from en element based retrieval system for content only queries. One can view this as answering the query:

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

For our example query and decomposition formalism, we would use only one pair

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

Ignoring the structural requirements altogether might be considered too extreme a reaction to the content oriented behavior of the users. This decomposition, however, can be implemented using a simple element-based XML retrieval system and serves as a baseline for our more involved decompositions.

Now, the environment-based and the content-only decompositions may be viewed as two extremes in a broad spectrum of options, where the former may be too strict and the latter too liberal. Which sensible decomposition methods sit somewhere in between those two extremes? We introduce two more query decompositions, in which we extend the environment-based decomposition by adding new `about`-functions. The added `about`-functions are made by propagating query terms upward and downward in the query.

We refer to the first query extension as *partial propagation* of query terms. The new `about`-functions are created by propagating query terms upward to the level of the predicates. That is, for each predicate we add a new `about`-function which contains all query terms used in any nested `about`-function. For our running example we add the following two (decomposed) `about`-functions:

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

In terms of the running example, the intuition is that we want to rank articles even if they do not have an abstract. Similarly we want to rank sections even they do not discuss their theorem explicitly inside a <theorem>-element.

Another way to make the queries less strict is to propagate query terms both upward *and* downward. We will refer to this query type of decomposition as *full propagation* of query terms. We extend the environment-based decomposition by adding a new `about`-function to each predicate. The `about`-functions use the predicates' location path as their own but use the full content query as content description. That is, we add to each predicate, an `about`-function of the form:

```
about(., full content query).
```

For our running example we would add the following two (decomposed) `about`-functions:

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

Note that this decomposition differs from the partial propagation only for topics that have more than one predicate.

Each of the decompositions described above results in a list of (location-path, content description) pairs and will be processed the same way in the two following steps of our retrieval process. Each pair is used to collect relevance information in the retrieval step. Then the results from different pairs are mixed in the mixture step. For the content-only decomposition, the mixing step can obviously be considered as trivial since there is only one pair.

3.2 Retrieval

Each of the (location-path, content description) pairs defined above can be used to locate sources of relevancy and to collect evidence from those sources using a standard information retrieval engine. That is, 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*. Taking the first `about` function of the environment-based decomposition of our running 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 status value 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. In this way we obtain, for the environment-based decomposition, a ranked lists of abstracts, sections, and theorems, respectively. In this subsection we briefly 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 collection. Inverted indexes are efficient for testing whether a term occurs in a document or element [24]. 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 [6].

Retrieval model. Since we use our retrieval engine to rank each `about` function separately, the queries fed to the IR engine are lists of query terms. For the ranking of elements with respect to relevancy for a given query, our retrieval engine uses a multinomial language model with Jelinek-Mercer smoothing [7]. We estimate a language model for each element. The elements are then ranked according to the likelihood of the query, given the estimated language model for the element. That is, we 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 (often called the *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 function, $s(Q, E)$, for an element E and a query $Q = (t_1, \ldots, t_k)$, which we compute as

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

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 value for the smoothing parameter is known to affect initial precision, and also the size of returned elements [26, 10]. 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.

Prior probabilities. The second major task in the retrieval model 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 [8, 21]. 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:

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

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),

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

As an aside, length priors, and extreme values for them, have been shown to be particularly important for XML retrieval [10]. 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. In our setup of the content and structure task we are usually not using language models to rank the target elements directly. Furthermore, for the task we are evaluating, the granularity of the result elements is generally specified in the query. It is thus not certain that the results for the content only task carry over to the content and structure task.

3.3 Mixture

At this stage, we have decomposed the query and identified sources of relevancy. We have used an information retrieval engine to collect evidence from those sources. Now it is time to put things together, and to do so, we focus on 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_{XP}.

We have a set of `about` functions, $A(q_{XP})$, from Section 3.1. In Section 3.2, we calculated scores for each `about` function separately. First we take an `about` function $a \in A(q_{XP})$ and a target element e and we need to calculate the score of e for a. Let $E(p_a)$ 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:

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

The notion of *connection* between two elements will not be explained further here but we refer to the W3C XPath semantics [25]. 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. a `about` function a:

$$(7) \qquad s(e, a) = \max_{e' \in E(a)} \chi_a(e, e') \cdot s(e', q_a),$$

where q_a is the content description of a. We can calculate $s(e', q_a)$ using Equation 5 since q_a is a natural language query.

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.

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_{XP}. We simply assign a score to an element by summing up its scores for each `about` function:

$$(8) \qquad s(e, q_{XP}) = \sum_{a \in A(q_{XP})} \alpha_a \cdot s(e, a),$$

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. That is indeed what we do for the experiments introduced in this paper.

In the actual queries, `about` functions could in principle be connected in a predicate using the logical operators 'AND' and 'OR'. We treat such cases as follows. We are liberal in the sense that we score elements even if only one of the functions returns a score, independent of whether 'AND' or 'OR' was used. Thus, 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 have a bias toward treating the queries as if they only used 'AND,' without, however, implementing full-blown coordination level matching on the level of `about` functions.

Let's now look at what the mixture does in terms of our running example and the environment-based decomposition. In the mixture, the engine must decide which elements to return to the user, and in which order. For the former task, the engine uses the location path

`//article//section`,

whose node-set contains all sections of all articles in the collection. We will refer to those elements as *target elements.* Next, the engine assigns 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".

4. EXPERIMENTS

The extensions of our information retrieval system to deal with CAS queries are quite involved. Are they worth the effort? In particular, how effective are our decomposition strategies? To address these questions we experiment with the INEX 2003 content-and-structure queries, which consists of the collection whose details were described in Section 2, 25 topics, as well as human relevance assessment for those topics. We use version 2.5 of the strict assessments, and an element is considered relevant if, and only if, it has been judged highly relevant and highly exhaustive [5].

4.1 Runs

We compare the decomposition strategies outlined in Section 3 (combined with the same retrieval approach and mixing strategy in all three cases) with the simple baseline which ignores the structural constraints expressed in the queries. This leads to the runs described below. All runs return a ranked list of 1000 most relevant elements for each query.

Element-based run. For comparison we consider a simple baseline, based on the content-only decomposition described in Section 3, that does not use the structural requirements at all. This is probably the simplest element-based extension of an information retrieval system that is able to process

CAS queries. We will refer to this run as the *element-based run.*

Environment-based run. The first run of our extended system is created using the environment-based decomposition without any query term propagation. We will refer to this run as the *environment-based run.*

Partial-propagation run. The second run of our extended system is created using the decomposition and partial term propagation extension described in Section 3. We will refer to this run as the *partial propagation run.*

Full-propagation run. The third run of our extended system is created using the decomposition and full term propagation extension described in Section 3. We will refer to this run as the *full propagation run.*

4.2 Metrics

We will compare the four runs using three standard information retrieval measures: *mean average precision*, which is a measure of how well a system returns only relevant elements; *recall at 1,000*, which is a measure of how well the system performs in retrieving all the relevant elements; and *precision at 10* which measures how well a system returns only relevant elements from the viewpoint of a user who only looks at the first 10 elements retrieved [1].

5. RESULTS

We present our results in three subsections. First, we average the results over all topics. After that we present results separately for distinct classes of topics. The first classification we consider is based on the granularity constraint of the topics. We will refer to this as a *semantic classification* of the topics. The second classification we consider is based on how many predicates the topic has. We refer to this as a *syntactic classification* of the topics.

5.1 Results over all Topics

Table 1 shows the performance of the four approaches. The results are averaged over all 25 topics. The first striking and somewhat disappointing observation is that, when using the non-extended queries, the rather non-trivial extension of our system (the environment-based run) does not outperform the rather trivial extension (the element-based run). The environment-based run is even slightly worse in terms of recall and early precision. At first glance it seems that all the effort of implementing the extension was for nothing. However, if we look at the results of runs using the extended queries (full- and partial- propagation), we see that the extended system does outperform the element-based run, both w.r.t. mean average precision and recall. For the precision at 10, our extended system is slightly inferior to the element-based system.

Note that none of our improvements is statistically significant. Significant improvement is difficult to achieve, having only 25 CAS queries. Furthermore, due to the structural constraints, the CAS queries are much more varied than traditional content-only topics. This suggests that it may be worth to try and group the CAS queries together and see how our methods perform on specific types of CAS queries—this is what we do in the following two subsections.

5.2 Results for Semantic Classification

The content and structure queries can be divided into classes based on the structural requirements they contain. It is interesting to see whether one approach works better for one class than another. We divide the queries into three distinct classes, depending on their granularity constraint:

- The first class consists of the 10 topics whose target element is whole articles.
- The second class contains the 8 topics whose target element is sections.
- The remaining 7 topics are grouped together in the third class. The topics in the class differ among themselves, and could thus be broken up further into several classes. The lack of available topics does however not allow for further breaking up of topics.

The above classification by granularity constraint is motivated by the intuition that retrieval effectiveness is dependent on the unit of retrieval. For example, the retrieval of an article can be done based on the content of the article, but the retrieval of a section could be based on the content of the section itself as well as the surrounding elements (such as the surrounding article).

Table 2 shows the results of evaluating our runs over the different target classes. Table 2(a) shows the results for CAS queries whose target is whole articles. For this class of queries the element-based run outperforms the environment-based run. A probable explanation for this result is that for some query terms, the environment-based run judges the articles based on their best sub-elements, but not on the overall quality of the article. This is in line with findings from earlier experiments carried out by Wilkinson [23], where documents were assigned the score of their single best section. The propagation of query terms adds the following `about`-function to each query,

- (`//article`, *full content query*).

Hence, for each query, the score of the articles is based both on the content of the article itself and on the content of the most relevant sub-elements. The use of propagated queries does indeed lead to an improvement in mean average precision when compared to the element based run. This outcome is again in line with the results of Wilkinson [23] where documents were assigned a score based on their own score and the scores of their sections. The precision at 10 articles retrieved is a bit disappointing but at this point we have not come up with an explanation of this performance. All runs achieved perfect recall.

Table 2(b) shows results of evaluating our runs over the class of topics whose target element is sections. Here we can see that the non-trivial extension of our system is effective for achieving improvement for both mean average precision and precision at 10 sections retrieved. Previous experiments by Wilkinson [23] showed that the retrieval of sections could benefit from taking into account both the score of the section itself and the score of the surrounding document. When users create structured elements to retrieve sections, they often put search constraints on the section itself and the surrounding article. Our environment-based run makes use of these constraints and thus bases its score on both the section and the surrounding article. Therefore, we see an

Run	MAP		Recall		P@10	
Element-based	0.3209		0.7153		0.3200	
Environment-based	0.3219	+0.3%	0.7067	-1.2%	0.3080	-3.8%
Partial propagation	0.3462	+7.9%	0.7396	+3.4%	0.3200	+0.0%
Full propagation	0.3519	+9.7%	0.7424	+3.8%	0.3120	-2.5%

Table 1: Results over all 25 topics

Run	MAP		Recall		P@10	
Element-based	0.3744		1.0000		0.3100	
Environment-based	0.3557	-5.0%	1.0000	+0.0%	0.2500	-19.4%
Partial propagation	0.4092	+9.3%	1.0000	+0.0%	0.2900	-9.4%
Full propagation	0.4092	+9.3%	1.0000	+0.0%	0.2900	-9.4%

(a) Results over the 10 topics with articles as target

Run	MAP		Recall		P@10	
Element-based	0.2436		0.6713		0.3250	
Environment-based	0.2673	+9.7%	0.6323	-5.8%	0.3673	+13.0%
Partial propagation	0.2718	+11.6%	0.6379	-5.0%	0.3750	+15.4%
Full propagation	0.2808	+15.3%	0.6407	-4.6%	0.3375	+3.8%

(b) Results over the 8 topics with sections as target

Run	MAP		Recall		P@10	
Element-based	0.3330		0.6284		0.3286	
Environment-based	0.3359	+0.9%	0.6651	+5.8%	0.3286	+0.0%
Partial propagation	0.3412	+2.5%	0.7615	+21.2%	0.3000	-8.7%
Full propagation	0.3514	+5.5%*	0.7661	+21.9%	0.3143	-4.4%

(c) Results over the 7 remaining topics

Table 2: Results over different classes of topics based on target constraint

improvement over the element-based run, which assigns a score to a section based solely on its content. The propagation of query terms can lead to further improvements when retrieving sections. Often, the propagated queries add to the queries a new source of evidence for relevancy. In this case, the additional sources seem to be useful for the system to achieve higher precision. The price for the increase in precision is a decreased performance in terms of recall.

Table 2(c) shows results of evaluating our runs over the class of remaining topics (whose target is neither articles nor sections). Explaining these results is a bit problematic since the topics do not have any positive property in common. They ended up in this class simply because they did not belong to either one of the two previous classes. Further analysis of the queries in this class has to wait until we have extended our test-set with the INEX 2004 test set, which should be available by late 2004; we are hopeful that this extension will allow us to extend our classification with more classes based on the granularity constraint.

5.3 Results for Syntactic Classification

The INEX 2003 queries can be divided into two classes based on how many predicates they contain. One-predicate queries have the form

```
location-path[abouts].
```

Two-predicate queries have the form

```
location-path[abouts]location-path[abouts].
```

Looking at these two classes separately is interesting because the queries in the latter class tend to be more complex than the ones in the former class. Intuitively, the latter class stands to gain more from complex query processing than the former one.

Table 3 shows the results of evaluating our runs over the two classes. It turns out that the more complex queries do indeed gain more from the complex query processing. The complex query processing hurts the simple queries for almost all measures, though. Our complex query processing does, however, help the complex queries, independent of the measure used. When we use query propagation we even manage to get statistically significant improvement in mean average precision when compared to the element-based run. For the complex queries there is a noticeable difference between using full or partial term propagation. Full propagation helps mean average precision while partial propagation helps precision at 10.

We can sum this up by saying that query term propagation leads to improvements for all queries. The complexity of the query roughly corresponds to the amount of improvement: the improvement for queries having two predicates is statistically significant.

6. DISCUSSION AND CONCLUSIONS

Hybrid queries tend to be more expressive, which should lead—in principle—to better retrieval performance [18, 22]. In practice, the processing of these hybrid queries within an IR system turns out to be far from trivial, because a delicate

Run	MAP		Recall		P@10	
Element-based	0.3486		0.7314		0.3529	
Environment-based	0.3364	-3.5%	0.7048	-3.6%	0.3176	-10.0%
Partial propagation	0.3683	+5.7%	0.7105	-2.9%	0.3412	-3.3%
Full propagation	0.3683	+5.7%	0.7105	-2.9%	0.3412	-3.3%

(a) Results over the 17 topics having one predicate

Run	MAP		Recall		P@10	
Element-based	0.2621		0.6667		0.2500	
Environment-based	0.2911	+11.1%	0.7126	+6.9%	0.2875	+15.0%
Partial propagation	0.2992	+14.2%*	0.8276	+24.1%	0.2750	+10.0%
Full propagation	0.3172	+21.0%*	0.8391	+25.9%	0.2500	+0.0%

(b) Results over the 8 topics having two predicates

Table 3: Results over classes of topics containing one or two predicates

balance between structural and content information needs to be sought. Several theoretically motivated frameworks have been proposed to process content and structure queries, dating back at least to [15]. Lalmas and Rölleke propose a probabilistic object-relational framework to model representation and retrieval strategies that account for vagueness w.r.t. both content and structural parts of queries [13]. Similarly, Piwowarski and Gallinari suggest an algebra for probabilistic XML retrieval [17]. These approaches are theoretically motivated and probabilistically sound, but have not yet been evaluated empirically.

Within the INEX initiative, many teams have evaluated their systems empirically. Carmel et al. extend the vector space model for searching XML documents via XML fragments [3]. XML fragments are a proper subset of the query language presented in this paper. They experiment using the INEX 2002 test collection and their best performing run was a baseline run which ignored the structural parts of the queries. Their initial results indicate that the success of the baseline is due to the simplicity of the queries. Liu et al. show that manually configured index and tag weights can lead to good retrieval performance [14]. They experiment their system using the INEX 2003 test collection.

In this paper, we presented an IR-approach to hybrid content-and-structure queries for structured documents. Our strategy is to decompose the query into pairs of location paths and content descriptions, then issue these as separate queries to a standard retrieval engine, and, finally, produce a final ranking that takes into account the scores of the different sources of evidence.

We experimented with an *environment-based* approach, which processes the hybrid queries step-by-step, and contrasted it with an *element-based* approach, a single simple query that ranks all elements satisfying the structural constraint of the target element, with respect to all the query content words. Both approaches turn out to be quite effective, outperforming our official runs at INEX 2003 (which ranked 1 and 2 over all submissions [5, 19]). The *element-based* approach is surprisingly effective, and sets a high baseline for the other approaches. The complex processing of the *environment-based* approach does not seem to pay off: there is only a marginal gain in MAP. This is a disappointing result, considering the additional expressiveness and control exercised by users formulating hybrid content-and-structure queries.

To further analyze this result, we decided to break down the topics both semantically (topics asking for articles, asking for sections, and the rest) and syntactically (topics having a single predicate, and those having two or more). The analysis revealed interesting differences: the *environment-based* approach decreases performance of topics asking for articles, but increases performance for topics asking for sections. In a similar vein, the *environment-based* approach decreases performance for topics with one predicate, but increases for topics with two or more predicates. The complex query processing fails to deliver for article topics and topics having only one predicate. Our best explanation for the relative effectiveness of the *element-based* approach for these topics is that this is due to the use of the longer "full content queries"—queries containing content words from every `about`-function in the CAS-query.

We decided to take this idea further and experiment with the propagation of query terms. We consider a *partial propagation* approach in which query terms are propagated to their ancestors in the location-paths, and a *full propagation* approach which propagates query terms both up and down the location-paths. The outcome is positive overall: now the article topics do improve, as do the topics having one predicate. The section topics, and the topics with more than two predicates improve even further. When looking at the score over all topics, we see that the decomposition, retrieval, and mixing strategy does pay off when we propagate query terms over the location-paths.

The INEX initiative holds great promise for furthering our understanding of effective ways of querying document-centric XML. Content-oriented XPath queries can be viewed as a natural extension of traditional fielded search, catering for users having varying degrees of knowledge of the markup-structure of the documents [11]. However, the richer language for content-and-structure queries results in a greater variety between the topics. In combination with the small number of topics available from INEX 2003 this implies that it is difficult to obtain significant improvements for almost any method.

Despite these limitations, we view our current results as a first, important step toward understanding the processing of hybrid queries. Our future research is on further decompositions of the queries (including the integration of evidence from blind relevance feedback), on retrieval parameters (length priors and smoothing, and specific settings depending on the unit of retrieval), and on ways of fusing the evidence obtained (including the weighting and normalization of scores). These experiments are all greatly facilitated by our three step strategy: decomposition, retrieval, and mixture.

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, 612.066.-302, 264-70-050, and 017.001.190.

8. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[2] N. J. Belkin, R. N. Oddy, and H. M. Brooks. ASK for Information Retrieval: Part I. Background and Theory. *Journal of Documentation*, 38(2):61–71, 1982.

[3] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML documents via XML fragments. In *Proceedings of the 26th Annual International ACM SIGIR Conference*, pages 151–158. ACM Press, 2003.

[4] N. Craswell, D. Hawking, and S. Robertson. Effective site finding using link anchor information. In *Proceedings of the 24th Annual International ACM SIGIR Conference*, pages 250–257. ACM Press, 2001.

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

[6] T. Grust. Accelerating XPath Location Steps. In *Proc. SIGMOD*, pages 109–120. ACM Press, 2002.

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

[8] 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.

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

[10] J. Kamps, M. de Rijke, and B. Sigurbjörnsson. Length normalization in XML retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference*, pages 80–87, 2004.

[11] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. Best-match querying from document-centric XML. In S. Amer-Yahia and L. Gravano, editors, *Proceedings Seventh International Workshop on the Web and Databases (WebDB 2004)*, pages 55–60, 2004.

[12] G. Kazai, M. Lalmas, and B. Piwowarski. INEX'03 Relevance Assessment Guide. In *INEX 2003 Workshop Proceedings*, pages 204–209, 2004.

[13] M. Lalmas and T. Rölleke. Modelling Vague Content and Structure Querying in XML Retrieval with a Probabilistic Object-Relational Framework. In *Proceedings of the 6th International Conference on Flexible Query Answering Systems, FQAS 2004*, volume 3055 of *Lecture Notes in Computer Science*, pages 432–445. Springer, 2004.

[14] S. Liu, Q. Zou, and W. W. Chu. Configurable indexing and ranking for XML information retrieval. In *Proceedings of the 27th annual international conference on Research and development in information retrieval*, pages 88–95. ACM Press, 2004.

[15] G. Navarro and R. Baeza-Yates. A language for queries on structure and contents of textual databases. In *Proceedings of the 18th Annual International ACM SIGIR Conference*, pages 93–101, 1995.

[16] R. A. O'Keefe and A. Trotman. The Simplest Query Language That Could Possibly Work. In *INEX 2003 Workshop Proceedings*, pages 167–174, 2004.

[17] B. Piwowarski and P. Gallinari. An algebra for probabilistic xml retrieval. In *Proceedings of the first Twente Data Management Workshop on XML Databasesand Information Retrieval*, pages 59–66, 2004.

[18] T. Schlieder and H. Meuss. Querying and ranking XML documents. *Journal of the American Society for Information Science and Technology*, 53:489–503, 2002.

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

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

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

[22] A. Trotman. Searching structured documents. *Information Processing and Management*, 40:619–632, 2004.

[23] R. Wilkinson. Effective retrieval of structured documents. In *Proceedings of the 17th ACM SIGIR Conference*, pages 311–317, 1994.

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

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

[26] 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.