Best-Match Querying from Document-Centric XML

Jaap Kamps[*] Maarten Marx Maarten de Rijke Börkur Sigurbjörnsson
Informatics Institute, University of Amsterdam
Kruislaan 403, 1098SJ Amsterdam, The Netherlands
{kamps,marx,mdr,borkur}@science.uva.nl

ABSTRACT

On the Web, there is a pervasive use of XML to give lightweight semantics to textual collections. Such document-centric XML collections require a query language that can gracefully handle structural constraints as well as constraints on the free text of the documents. Our main contributions are three-fold. First, we outline two fragments of XPath tailored to users that have varying degrees of understanding of the XML structure used, and give both syntactic and semantic characterizations of these fragments. Second, we extend XPath with an `about` function having a best-match semantics based on the relevance of the document component for the expressed information need. Third, we evaluate the resulting query language using the INEX 2003 test suite, and show that best-match approaches outperform exact-match approaches for evaluating content-and-structure queries.

General Terms

Full-text XML querying, XPath, XML Retrieval

1. INTRODUCTION

There is an ever growing availability of semi-structured information on the Web. Increasingly, Web users have access to text documents, equipped with some semantic hints through XML-markup. How can we query data of this kind? We could adopt a standard information retrieval approach: perform best match querying using plain text queries. But this would not allow users to specify constraints on the document structure. Alternatively, we could query the documents using a database approach: perform exact-match using XPath queries. The disadvantage here is that effective query formulation is non-trivial and recall is often too low.

Within the INitiative for the Evaluation of XML Retrieval (INEX) [8], the above shortcomings are being addressed by combining the two approaches. Free text search functionality is added to XPath, in the form of a new function, the `about` function. It has the same syntax as the standard `contains` function. The `about` function has three main features: (1) it allows us to formulate information needs in terms of a mixture of content and structure requirements; (2) it allows us to use best-match querying of document-centric XML; and (3) it has a standard syntax.

User oriented studies from INEX have shown that full XPath is too complex for querying document-centric XML documents, not just for average users but also for experts. Moreover, it is unrealistic to assume that Web users have full knowledge of the structure of the documents they want to query. We discuss several XPath fragments (extended with `about`) that are simpler and, we believe, more effective for querying document-centric XML for users.

Proposals for new query languages should be subjected to a number of sanity checks: we need to understand their *expressive power*, and we need to assess their *effectiveness* with respect to the information need they are meant to address. To address the former we relate the query languages to logical structures, for which expressiveness results are well-known. To address the latter, we discuss several ways in which a search engine can process queries formulated in the language, in order to satisfy a user's information need. We perform a user-oriented evaluation of the approaches, using test suites made available through INEX.

In Section 2 we provide background on querying document-centric XML. In Section 3 we discuss content-oriented flavors of XPath and provide them with semantic characterizations of their expressive power. Section 4 describes the set-up for our user-oriented evaluation, whose results are discussed in Section 5. We conclude in Section 6.

2. QUERYING XML

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*. Data-centric documents are highly structured data marked up with XML tags. An example of data-centric XML is Geographic data in XML [10]. 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 that the order in which XML elements occur is crucial. For our experiments we use the document-centric XML collection that comes with the INEX test suite [8]. It contains over 12,000 computer science articles from 21 IEEE Computer Society journals. The articles are marked up with XML tags. On average an article contains 1532 elements and the average element depth is 6.9. About 170 tag names are used, such as articles $\langle\mathsf{article}\rangle$, sections $\langle\mathsf{sec}\rangle$, author names $\langle\mathsf{au}\rangle$, affiliations $\langle\mathsf{aff}\rangle$, etc.

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

Seventh International Workshop on the Web and Databases (WebDB 2004),
June 17–18, 2004, Paris, France

Whereas currently emerging standards for querying XML, such as XPath and XQuery, can be very effective for querying data-centric XML, another approach may be needed for querying document-centric XML. The latter task is a natural meeting point of two disciplines: the XML nature of the documents calls for methods from the database field for querying structure, and the textual nature of the documents calls for approaches from the field of information retrieval (IR) (cf. [14, Section 5]). It is interesting to contrast the two subtasks. As to querying structure, XML query languages such as XPath have a definite semantics. Judging whether an element satisfies an XPath query can be done by a computer (XPath processor), based on the pattern appearing in the XML document, using an *exact match* approach. It is clearly defined which nodes or elements match a given query. An XPath processor will return precisely these elements with no inherent ranking of results. As to querying text, on the other hand, IR uses free text queries. These can be keywords or full sentences describing an information need. An IR system uses a *best match* approach: it attempts to rank the nodes or elements by their topical relevance to the user's query, and returns the results, ordered by estimated relevance.

In IR, queries are not taken literally but as an expression of an implicit, underlying information need. Users generally lack an intimate knowledge of the sorts of documents that may satisfy their information need. As a result, they often have problems formulating precise queries for their information need [1]. E.g., a document that talks about *macintosh computers* might be considered as a valid response to a query about *apple computers*. To a general user *macintosh computers* will convey the same information as *apple computers*. Note that this problem is aggravated in the context of querying document-centric XML; is it realistic to assume that every user is familiar with the document structure used?

Determining whether a document or element answers a query is not a clearly defined mechanical process, but a judgment made by a human assessor based on the rather vague notion of relevance [12]. System performance is evaluated experimentally using a test-set consisting of (1) a collection of documents; (2) a number of queries; and (3) human-made relevance judgments that determine which documents or elements answer particular queries. The quality of an IR system is measured in terms of how accurately the system was able to imitate the human assessors.

3. CONTENT ORIENTED XPATH

To query document-centric XML documents we need a hybrid query language, in which content and structural requirements can be expressed and mixed. At INEX, an XPath-like query language is suggested, 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.

In this section, we will first motivate why XPath needs to be *restricted* and examine some fragments of XPath (Section 3.1). We will then motivate why those fragments need to be *extended* with the `about` function (Section 3.2). We put everything together, by briefly discussing the query format used at INEX (Section 3.3).

3.1 Restricting XPath

Experience from INEX has shown that people—in this case, academics familiar with query languages—have great difficulties in using (the navigational part of) XPath to formulate queries that combine content and structural aspects [11]. The restriction to navigational XPath was originally motivated by the fact that it is a widely used technology, whence it was assumed that it would be easily learnable. This assumption proved to be wrong.

Based on the extensive data described in [11], we argue that the cause of users' difficulties in writing content-and-structure queries is located in a combination of two related items: (1) Users have no, or only incomplete, knowledge of the structure of the documents, that is, of the DTD.[1] (2) Users have problems handling the expressive power of XPath. In particular, the fact that the same query can be expressed in several fundamentally different ways proved problematic for users. These observations lead to two constraints for XPath fragments: it should be possible to formulate information needs even with limited knowledge of the DTD, and the expressive power should be restricted.

A user's knowledge about a set of documents can be naturally formalized in terms of an *indiscernibility relation* over the elements selected by an XPath query: a binary relation that identifies elements in a document. What does such a relation have to do with query languages? We say that a language is *safe* or *well-designed* if indiscernible elements cannot be distinguished by an expression in the query language. This design criterion will help us single out natural XPath fragments. In fact, the fragments discussed below have a perfect fit with two user profiles formalized by an indiscernibility relation: not only are they safe, they are also complete in the sense that every first-order definable set of indiscernible elements can be defined in the language.

Below, we define two user profiles, both capturing users with limited knowledge of the DTD. First, we consider, what we call, *ignorant users* who only know the tag names. Second, we consider *semi-ignorant users*, who know the tag names and have some clue about the hierarchal structure of the elements, without knowing the full details. For both profiles we will design fragments that are *safe* for the sketched user profiles; we interpret this as saying that the chance that a user makes a semantic mistake when describing his information need in terms of XPath is minimal. For clarity, in this subsection we only consider the *navigational* part of XPath. The next subsection deals with the `about` function.

Ignorant Users. Users formulating queries at INEX did not have a clear idea of the DTD of the collection [11]. Typically, they browsed the documents and picked up some knowledge about the available tags in this manner. For users who know (a subset of) the tag names, but do not (want to) know the structure of the documents, we create an XPath fragment which exactly fits their knowledge. Specifically, our ignorant user is able to ask questions like: "Give me sections about weather forecasting where an author is affiliated in California". In XPath this could be written as:

```
//sec[about(.,'weather forecasting') and
                            //aff[about(.,'California')]
```

More generally, the user can express his information need as a conjunction of two boolean formulas: one restricting

[1] The DTD of the INEX XML document collection was extremely complex. There were 192 different content types, including 11 different tag names for representing paragraphs.

the element of interest, and the other restricting the surrounding document. The following syntax, which we call *non-structure aware XPath* allows this. A query is of the form `//::tag[P]`, where `tag` is either the wild card `*` or a tag name, and `P` is a predicate created using '`and`,' '`or`,' and '`not`' from location paths of the form `//::tag`. Note that when // :: `t` is used in a filter it means "there exists a descendant of the root with tag `t`". I.e., `//::t` simply says that somewhere in the document there is a `t` element.

We turn to a semantic characterization of this fragment. In social network theory [15] several indiscernibility relations have been proposed, including the useful and robust notion of *bisimulation* (a.k.a. 'regular equivalence'). We need the following special "structurally unaware" version.

Definition 1. Let D, D' be documents and B a non-empty binary relation between the elements of D and D'. We call B a *structure unaware bisimulation* if, whenever xBy, then

1. x and y have the same tag name;
2. if there exists an $x' \in D$, then there exists a $y' \in D'$ such that $x'By'$;
3. conversely for $y' \in D'$.

Let $\phi(x)$ be a first-order formula (in one free variable) in a suitable vocabulary; $\phi(x)$ is *invariant under bisimulations* whenever the following holds: for any a, b and bisimulation B, if $\phi(a)$ and aBb hold, then $\phi(b)$ holds as well.

A few comments. First, since we are usually comparing elements within a single document, our notion of indiscernibility relation is an *auto*-bisimulation, where D and D' in Definition 1 are the same document. Secondly, in the usual definition of bisimulation, the clauses in items 2 and 3 above are conditioned on x' (and y') being "structurally" related to x (and y, respectively); but our ignorant user is not aware of the structure, hence we omitted these conditions.

Theorem 2.

1. *Elements that are related by a structure unaware bisimulation cannot be distinguished by a non-structure aware XPath expression.*
2. *Every first-order formula that is invariant under structure unaware bisimulations is definable by a non-structure aware XPath expression.*

We can conclude that this language fits perfectly to the sketched user profile: the first part of the theorem states that it is *safe*, the second that it is *complete.*

Semi-ignorant Users. For semi-ignorant users, we will define two equivalent XPath fragments. One coincides with the fragment proposed in [11] and is supported by the query working group at INEX 2003 [13]. We will show that these fragments have a meaningful semantic characterization. The fact that this fragment fits a common user profile is strong evidence for its naturalness.

Semi-ignorant users have some ideas about the hierarchical structure of the documents. E.g., they know that paragraphs are below sections but, as pointed out in [11], they need not know that there *can* be elements in between. For this reason, [11] proposes Positive Descendant XPath: the fragment of XPath in which only the descendant axis may be used and the booleans in the predicates are restricted to "`and`" and "`or`".

We sketch two possible ways in which semi-ignorant users might pose queries. Suppose a user is interested in 'bisimulation' theorems which appear in sections about 'XPath.' He knows about the theorem tag ⟨theorem⟩ and the section tag ⟨sec⟩; he also knows that theorems can be nested somewhere inside sections. This user might ask:

(1)
```
//sec[about(.,'XPath')]//theorem[about(.,
                                  'bisimulation')]
```

Another user might formulate the same need as:

(2)
```
//theorem[about(.,'bisimulation') and
            ancestor::sec[about(.,'XPath')]]
```

The two users seem to engage in different mental processes when formulating their queries. The first thinks top-down: zoom in on a relevant section and then specify what sort of information should be retrieved from that section. The second approaches the problem bottom-up: determine a segment of interest and then think about sections that might contain the segment. The authors of this paper disagree on which scenario is more natural. Both scenarios can be captured in an XPath fragment, and we will show that the two fragments are equivalent.

To admit formulation (2) above, we need to allow both descendant and ancestor relations. We provide O'Keefe and Trotman's fragment [11] with a double characterization: a semantic one in terms of simulations, and a syntactic one, as a fragment of a well-known language in computer science, the temporal logic CTL. First, we need some definitions.

Definition 3. *Positive Temporal XPath* consists of queries of the form //`tag`[`P`], where `P` is in the following restriction of navigational XPath:

- the only axis relations are `descendant` and `ancestor`;
- only boolean `and` and `or` can be used in filters.

As none of the above two XPath fragments contains negations, bisimulation is too strong a notion [9]. As a general fact, positive fragments correspond to simulations, which are bisimulations from which one of the directions is dropped. We use $<$ to denote the descendant relation between elements; i.e., $x < y$ means that y is a descendant of x.

Definition 4. Let D, D' be documents and B a non-empty binary relation between the elements of D and D'. We call B a *temporal simulation* if, whenever xBy, then

1. x and y have the same tag names;
2. if there exists an $x' \in D$ such that $x < x'$, then there exists a $y' \in D'$ such that $y < y'$ and $x'By'$;
3. similarly when $x' < x$.

Temporal simulations correspond to users that know the element hierarchy: note that both elements below *and* above have to be simulated. The next theorem is an analogue of Theorem 2 for Positive Descendant XPath: it is both safe and complete for semi-ignorant users.

Theorem 5. *Let X be a set of nodes. The following are equivalent on trees.*

1. *X is definable by a first-order formula in one free variable in the signature with $<$ and unary predicates which is preserved under temporal simulations.*
2. *X is definable as the answer set of a Positive Descendant XPath formula.*
3. *X is definable as the answer set of a Positive Temporal XPath query.*

The proof uses ideas from modal logic [3, Theorem 2.78] together with ideas from [2, Theorem 3.2]. We conjecture that the language in item 3 of Theorem 5 is exponentially more succinct than the language in item 2.

3.2 Extending XPath

Now that we have looked at restrictions of the navigational part of XPath to "manageable" fragments, we look at extensions with the `about` function. Although `about` has the same syntax as the XPath function `contains`, their semantics are radically different. Because of its strict, boolean character, `contains` is not suitable for text rich documents. The semantics of `about` is meant to be very liberal. Consider the element `<aff>'Stanford University'</aff>`. A human assessor will likely decide that `about(.//aff,'California')` returns true if that element is below the node of evaluation; but an XPath processor equipped only with `contains` would have difficulties trying to do the same. As a more elaborate example, look at the following query (against a collection containing several articles):

> Find articles where the author is affiliated in California. From those articles return sections about weather forecasting systems.

In a hybrid syntax, mixing content and structure, this would be something like

```
//article[about(.//au//aff,'California')]//sec[
            about(.,'weather forecasting systems')]
```

This query has two content-based restrictions, linked by a structural constraint. The semantics of this query is not strict. I.e., while the navigational XPath constraint on the resulting node set is to be interpreted strictly, the semantics of `about` is necessarily vague. In the spirit of information retrieval, the ultimate decision of relevance is in the hands of a human assessor, who may bring lots of context and world knowledge to his judgment. E.g., a human assessor is likely to judge a section about 'storm prediction systems' to be relevant to the information need expressed above.

3.3 INEX query format

At INEX, navigational XPath extended with the `about` function and attribute-value comparisons is used to express so-called Content-And-Structure (CAS) topics.[2] These topics contain the same three parts as traditional IR topics [6, 4]: title, description and narrative. The description and narrative describe the information need in natural language. The title describes the information need using XPath and the about function. At INEX 2003, full XPath was allowed. At INEX 2004 descendant positive XPath (i.e., the restricted fragment for semi-ignorant users) is used [13].

At INEX, relevance is assessed by humans, solely on the basis of the narrative, using a best-match approach. However, the granularity constraint of the results is evaluated using an exact-match approach. In the case of our example topic above, only elements exactly matching the XPath expression `//article//sec` can possibly be relevant. Further relevance assessments of the sections are made by a human assessor, i.e., the judgement whether the sections are written by Californians and are about weather forecasting systems.

4. EXPERIMENTAL SETUP

We turn to evaluation and comparison of several methods of giving answers to the queries. Specifically, our aim is to compare two opposing views of query processing with respect to their effectiveness in satisfying users' information needs. First, we use an *exact match approach* which performs somewhat strict XPath processing. Second, we use a *best-match approach.* For our evaluation, we use the INEX 2003 test suite, consisting of 30 content-and-structure queries (together with human assessments that we use to score systems' outputs). First,we explain the retrieval methods used, and in Section 5 we present and discuss our results.

4.1 Indexing the collection

For both the exact match approach and the best-match approach we will work with an indexed version of the documents in the collection. 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 [16]. 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 [5].

4.2 Exact match approaches

Our task is to return a (ranked) list of relevant elements, given a query expressed in the INEX query language. In exact match approaches to this task we want to answer the queries using an XPath processor. Thus, we need to translate the `about` function into XPath. The straightforward solution is to replace `about` with the `contains` function. The queries can then be processed using an XPath processing engine. Note, however, that the text part of the `about` function is often a list of potentially helpful keywords, and it is very unlikely that any element would contain this particular list of keywords as a substring. Therefore, we break the `about` function up into either a conjunction or disjunction of `contains` functions, one for each word. E.g., using conjunctions we translate the query:

```
(3)  //article[about(.//au//aff,'California')]//sec[
               about(.,'weather forecasting systems')]
```

into the query

```
//article[contains(.//au//aff,'California')]//sec[
     contains(., 'weather') and contains(.,
           'forecasting') and contains(.,'systems')].
```

In the disjunctive translation, `or`'s are used instead of `and`'s; in our experiments, we made runs with both types of translation (conjunctive and disjunctive). Note that the retrieval is boolean, and, moreover, that XPath provides no means for ordering the results; the ordering that we used in the lists of elements returned in response to a query is random.

4.3 Best-match approaches

How do we process INEX queries using a best-match approach? Best-match approaches as used in IR allow us to

[2] "Topic" is IR parlance for a formal expression of an information need.

estimate to which extent an element answers a textual query. In particular, we can estimate to which extent an element fulfills an about function. Briefly, we break up each XPath query into several textual queries, one for each about function. For our running example (3) we get the queries

(Q3a) California

(Q3b) weather forecasting systems,

which we refer to as *partial content queries*. We refer to the concatenation of those queries

(Q3) California weather forecasting systems

as the *full content query*. Our element-based retrieval system can assign scores to elements, by estimating how well an element answers the information need expressed in the query. We refer to the path `//article//sec` as the *target constraint*. And to the paths `//article//au//aff` and `//article//sec` as the *path constraints* of the respective about functions. We use our retrieval system to define semantics for the about function. An element is in the answer set of the about function if it fulfills the path constraint of the function, and our retrieval system judges its content to be relevant to a given (partial) content query.

More precisely, we view the problem of answering an INEX XPath query as a multi-faceted problem, involving elements, as well as the documents containing those elements and the environment of those elements (i.e., other elements in the same document that may contain hints about their contents). For each of those aspects, we generate a separate retrieval run, thus obtaining, for each (target) element e, scores $score_{ele}(e)$, $score_{doc}(e)$, and $score_{env}(e)$, and these scores are then combined to produce a final score $score_{mix}(e)$ for every element e. The retrieval model used in all of our runs is the so-called multinomial language model [7].

We will now discuss the three separate retrieval runs. First, the element-based run is a content-based run, producing a ranked list of elements, where the target restriction is the only structural constraint which is satisfied. Scoring is based solely on the content of the target element. In the case of our running example (3) we can think of this as a run that evaluates the query

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

For a given query, elements are assigned their own retrieval score; that is, for an element e, the score obtained in the element-based run, is simply $score_{ele}(e) := score(e)$, where the latter denotes the so-called retrieval status value returned by the system for e.

Second, our document-based retrieval run is a content-based run, outputting a ranked list of elements, where, again, the target restriction is the only structural constraint that is enforced. An element receives the retrieval status value of the article that contains it.[3] In the case of our running example (3) think of this as a run that evaluates the query

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

In sum, an element is assigned the retrieval status value returned by the system for the document that contains the element: $score_{doc}(e) := score(d)$.

Third, in the environment-based run we consider, given a query and a document, all elements e' in the document that are contained in the node set of a path constraint in one of the about predicates in the query; those elements e', then, constitute the *environment*. The score of an element e that satisfies the target constraint is the sum of two things: first, e's score on the partial content query in the about restriction on the target;[4] second, the sum over all other about predicates in the query of the maximum score achieved by any element e' in the environment on the partial content query contained in that about statement. In our running example (3), `about(.//au//aff,'California')` identifies elements in the environment, and `about(.,'weather forecasting systems')` is the restriction on the target. More formally, we reorder the elements of the element based run by assigning them the score

$$score_{env}(e) := score_{\text{target }\texttt{about}}(e) + \sum_{a} \max_{e'} score(e'),$$

where "target about" is the target about restriction used to select e, and where a ranges over all other about functions in the query, and e' is in the environment.

Finally, we combine the scores defined so far. We perform a simple combination, where we re-order the elements of the element-based run, by assigning them a new score:

$$score_{mix}(e) = score_{ele}(e) + score_{doc}(e) + score_{env}(e).$$

Run	MAP		Recall
Conjunction run	0.1081		0.2461
Disjunction run	0.0243	-77.5%	0.0601

Table 1: Results for exact-match approaches, change is measured relative to the conjunction run

Run	MAP		Recall
Document-based run	0.2372		0.6094
Element-based run	0.3033	+27.9%	0.5966
Environment-based run	0.3090	+30.3%	0.5966
Mix-doc-ele-env	0.3330	+40.4%	0.5966

Table 2: Results for best-match approaches, change is measured relative to the document-based run

5. EXPERIMENTAL RESULTS

We evaluate the runs described in Section 4 using the standard recall and precision-based measures [6]. Table 1 shows the results for the exact-match approaches. The mean average precision (MAP) score of the conjunction run is much higher than the score of the disjunction run. This is not surprising, as the run using conjunctions is much more selective: it returns few, but mostly relevant elements. This leads to reasonable precision, but fairly low recall. Perhaps somewhat counterintuitively, the disjunction run does not "fix" the recall problem; the recall of this run is actually less than that to the conjunction run. This can be explained by the fact that an exact-match approach does not order the result set. Given that the retrieval task is to retrieve for each query maximally 1000 elements, this becomes a particularly limited strategy. The disjunction approach usually finds far more than 1000 elements and is forced to randomly choose 1000 elements from the result set.

Table 2 shows the results for the best-match approaches. The document-based run serves as a baseline for our best-

[3] We consider this to be our baseline run; it illustrates what can be achieved with a document retrieval system and simple path filtering.

[4] By the definition of the INEX query format, such about restrictions always exist.

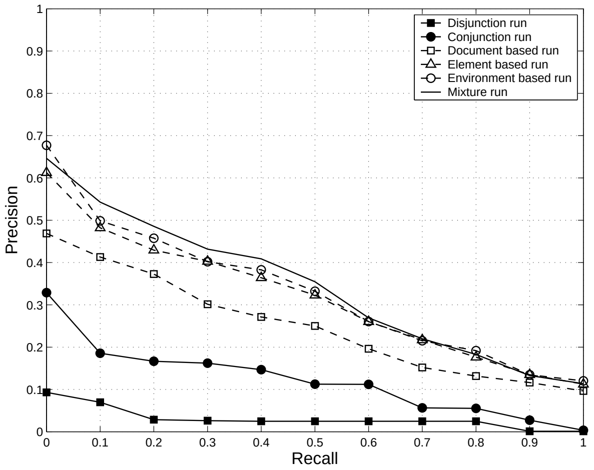

Figure 1: Precision-recall curves for all experiments

match approaches, i.e., as a lower bound on the effectiveness of a document-based retrieval system. Similarly, the element-based run serves as a lower bound on the effectiveness of element-based retrieval system. We see that the element-based approach scores significantly better than the document-based approach. The environment-based run faithfully implementing the queries leads to a mild further improvement. Figure 1 shows the precision-recall graphs for all runs. Although the environment-based run does not score significantly better than the element-based run, we can see that the run gives better performance at low recall levels. The combination of all three best-match approaches leads to further improvement of retrieval effectiveness.

The performance of the best-match approaches is much better than the exact-match approaches, both in terms of MAP and recall.

6. CONCLUSIONS

The widespread use of XML to mark-up textual documents prompts the need for appropriate query languages. Querying document-centric XML is the natural meeting place of two fields. The structure of the documents asks for approaches from the database field, whereas the textual content of the documents asks for approaches from the field of information retrieval. This suggests a query language that mixes constraints on the content with constraints on the structure. While it is unrealistic to assume that arbitrary users of a Web collection are fully aware of the precise XML-structure of the documents, we do want to allow users to make use of their (limited) knowledge of the structure.

We outlined two fragments of XPath tailored to users that have varying degrees of understanding of the used XML structure—one for "ignorant" users that only know (some of) the tags; and one for "semi-ignorant" users that only know (some of) the valid nestings between tags—and gave both syntactic and semantic characterizations of these fragments. As for the content part, we do not want to interpret textual queries literally, but as expressions of the underlying information needs. This is realized by adding an `about` function to XPath, having the exact same syntax as `contains`. The semantics of the `about` function is radically different, it is a best-match semantics based on the relevance of the document component for the expressed information need.

We evaluated querying document-centric XML collections from a user perspective using the INEX 2003 test-suite. The run combining evidence from all elements, the documents that contain them, and the surrounding elements (the environment) leads to the best performance; this can be interpreted as a sign that taking both content and structure into account helps performance. We showed that best-match approaches outperform exact-match approaches for evaluating content-and-structure queries. This suggests that users querying document-centric XML will likely be more satisfied using a best-match query language, rather than an exact match query language such as standard XPath.

7. ACKNOWLEDGMENTS

Jaap Kamps was supported by the Netherlands Organization for Scientific Research (NWO) under project number 612.066.302. Maarten Marx was supported by NWO, under project number 612.000.106. Maarten de Rijke was supported by 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. 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.

[2] M. Benedikt, W. Fan, and G. Kuper. Structural Properties of XPath Fragments. In *Proc. ICDT*, 2003.

[3] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[4] CLEF. Cross-Language Evaluation Forum, 2003. `http://www.clef-campaign.org`.

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

[6] D. Harman. Overview of the First Text REtrieval Conference (TREC-1). In *Proc. TREC-1*, 1993.

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

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

[9] N. Kurtonina and M. de Rijke. Expressiveness of concept expressions in first-order description logics. *Artificial Intelligence*, 107(2):303–333, 1999.

[10] W. May. Information extraction and integration with FLORID: The MONDIAL case study. Technical report, Universität Freiburg, Institut für Informatik, 1999.

[11] R. A. O'Keefe and A. Trotman. The Simplest Query Language That Could Possibly Work. In *Proceedings of the 2nd INEX Workshop*, 2004.

[12] T. Saracevic. Relevance: A review of and a framework for the thinking on the notion in information science. *JASIS*, 26:321–343, 1975.

[13] B. Sigurbjörnsson and A. Trotman. Queries, INEX 2003 working group report. In *Proceedings of the 2nd INEX Workshop*, 2004.

[14] V. Vianu. A Web odyssey: from Codd to XML. In *Proc. PODS*, pages 1–15. ACM Press, 2001.

[15] S. Wasserman and K. Faust. *Social Network Analysis*. Cambridge University Press, 1994.

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