Structured Queries in XML Retrieval

Jaap Kamps[1,2] Maarten Marx[2] Maarten de Rijke[2] Börkur Sigurbjörnsson[2]

[1] Archives and Information Studies, University of Amsterdam, Amsterdam, The Netherlands
[2] Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
{kamps,marx,mdr,borkur}@science.uva.nl

ABSTRACT

Document-centric XML is a mixture of text and structure. With the increased availability of document-centric XML content comes a need for query facilities in which both structural constraints and constraints on the content of the documents can be expressed. How does the expressiveness of languages for querying XML documents help users to express their information needs? We address this question from both an experimental and a theoretical point of view. Our experimental analysis compares a structure-ignorant with a structure-aware retrieval approach using the test-suite of the 2004 edition of the INEX XML retrieval evaluation initiative. Theoretically, we create mathematical models of users' knowledge of a set of documents and define query languages which exactly fit these models. One of these languages corresponds to an XML version of fielded search, the other to the INEX query language.

Our main findings are: First, while structure is used in varying degrees of complexity, over half of the queries can be expressed in a fielded-search like format which does not use the hierarchical structure of the documents. Second, structure is used as a search hint, and not a strict requirement, when judged against the underlying information need. Third, the use of structure in queries functions as a precision enhancing device.

Categories and Subject Descriptors

H.2 [**Database Management**]: H.2.3 Languages—*Query Languages*; 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

Measurement, Performance, Experimentation

Keywords

Full-text XML querying, XPath, XML Retrieval

1. INTRODUCTION

There is an ever growing availability of semi-structured information, on the Web and in digital libraries. Increasingly, users, both expert and non-expert, have access to text documents, equipped with some semantic hints through XML-markup. How can we query such data? We could adopt a standard information retrieval (IR) 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 or XQuery queries. But here, recall is often too low.

Based on their content, XML documents may be categorized into two groups: *data-centric* and *document-centric*. The former contain highly structured data marked up with XML tags, an example being geographic data in XML [13]. Document-centric documents are loosely structured documents (often text) marked-up with XML, with electronic journals in XML providing important examples. Whereas emerging standards for querying XML, such as XPath and XQuery, can be very effective for querying data-centric XML, another approach seems to 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 IR (cf. [24, 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 elements match a given query. An XPath processor will return precisely these elements with no inherent ranking of results. In contrast, for querying text 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 results by their topical relevance to the user's query.

At INEX, the INitiative for the Evaluation of XML Retrieval (INEX) [9], the focus is on a *combined* approach to XML retrieval, featuring aspects of exact match and best-match retrieval. Free text search functionality is added to XPath, in the form of a new `about` function. With the same (standard) syntax as the standard `contains` function, the `about` function has two main features; it allows the user to (1) express information needs with a mixture of content and structure requirements; and (2) use best-match querying of document-centric XML. Although the `about` function has the same syntax as `contains`, its semantics is not strictly defined but left to relevance judgments by human assessors.

The aim of this paper is to understand the combined DB and IR approach to XML retrieval. The specific reearch problem we tackle is how the exact match and the best match approaches can be combined in an effective and useful manner. We address this problem by answering the following questions:

1. How do users exploit the additional expressive power of structural constraints in their queries?
2. What is the effect on retrieval performance of adding structural constraints to queries?
3. What is the appropriate query language for XML retrieval?

We will answer the first two questions by an analysis of the INEX data. For the third, such an analysis has been carried out in [15], resulting in a proposal for a query language based on their findings. We give a mathematical model of users' knowledge of an XML collection and link this to the appropriate expressive power of query languages. Our main results are:

- Structural constraints are mainly used as search hints, not as strict requirements. The hierarchical nature of the documents is used in less than half of the examined queries.
- Adding structural constraints has a positive effect on early precision and a negative effect on overall recall.
- A typology of different uses of content and structure queries.
- Intuitive mathematical models of users' knowledge of a set of XML documents and the formulation of query languages which exactly fit this knowledge.

The rest of this paper is organized as follows. In Section 2 we discuss the retrieval task at INEX and analyze the queries used, resulting in an answer to question 1. In Section 3 we report on experiments comparing the retrieval effectiveness of structured queries versus ordinary queries, thereby answering question 2. Section 4 describes content-oriented flavors of XPath and provides semantic characterizations of their expressive power. It gives a mathematical basis for answering the third question. We conclude in Section 5.

2. EXPRESSING INFORMATION NEEDS WITH CONTENT-AND-STRUCTURE

In this section we answer the first question from the introduction. We examine how users express their information needs in the XPath-like query language used at INEX 2004. We will see that about half of the queries do not use the hierarchical structure of the documents. They simply require that certain keywords occur in elements with a certain tag name.

In addition, we find that the requested elements in queries should be viewed as retrieval hints, not as strict requirements on the results: over half of the relevant elements has another tag name than the one specified in the query.

2.1 The INEX XML Document Collection

The queries we study are run against the document-centric XML collection that comes with the INitiative for the Evaluation of XML Retrieval (INEX) [9]. It contains over 12,000 articles from 21 IEEE Computer Society journals, marked up with XML tags. The DTD of the INEX XML document collection is extremely complex. There are 192 different content types, including 11 different tag names for representing paragraphs; about 170 tag names are actually used in the collection, including articles ⟨article⟩, sections ⟨sec⟩, author names ⟨au⟩, affiliations ⟨aff⟩, etc. On average an article contains 1,532 elements and the average element depth is 6.9.

The INEX setup is such that we should think of the INEX document collection as a forest of articles. These are XML documents whose roots have the tag name `article`. Because the actual storage of the documents may be different, most queries start with the prefix `//article`.[1] This is only an artefact of the representation and we will treat the tag name `article` as referring to the root of a document.

2.2 The INEX Topic Format

At INEX, two types of topic are used: Content-Only (CO) topics and Content-And-Structure (CAS) topics. All topics contain the same three fields as traditional IR topics [8]: title, description and narrative. The title is the actual query submitted to the retrieval system. The description and narrative describe the information need in natural language. The described information need is used to judge the relevancy of the retrieved answers to the queries. The difference between the CO and CAS topics lies in the topic title. In the case of the CO topics, the title describes the information need as a small list of keywords. In the case of CAS topics, the title describes the information need using (a flavor of) XPath extended with the `about` function discussed below. At INEX 2003, full XPath was allowed, and at INEX 2004 a restricted version of XPath was used [21, 23]. Here we analyze the title part of the CAS topics, which we simply call *queries* from now on.

2.3 INEX 2004 Queries

The specific instructions for topic development at INEX 2004 [20] stated that CAS queries

- should use only descendant axis (i.e., `//`),
- should use only boolean `and` and `or`,
- should contain at least one `about` statement, and
- the rightmost filter should be an `about` statement.

The `about` function is the IR counterpart of the familiar XPath `contains` function. Although they have the same syntax, 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>`. Unlike an XPath processor equipped only with `contains`, a human assessor will likely decide that `about(.//aff,'California')` returns true if that element is below the node of evaluation. For a more elaborate example, look at the following query (against a collection containing several articles):

[1]Only three queries (out of the 34 used at INEX 2004) do not start with this prefix; however, these queries are prefixed with either ⟨sec⟩ or ⟨abs⟩ tags that only occur in the context of an ⟨article⟩ tag anyway.

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 structural constraints, 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. In the spirit of IR, the ultimate decision of relevancy is in the hands of a human assessor, who may bring lots of context and world knowledge to her 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.

The resulting language is called NEXI (Narrowed Extended XPath I) [23]. For our analysis, we use the set of 34 CAS queries (version 2004-7) with query numbers 127–147, and 149–161 [for details, see 6].

Requested Elements. One of the main advantages of using CAS queries is that they allow the user to specify the types of elements that should be returned as answers. Table 1 lists which kind of elements were requested in the 34 CAS queries studied.

Element	Frequency	Percentage
sec *(section)*	16	47.06%
article	5	14.71%
p *(paragraph)*	4	11.76%
*	2	5.88%
abs *(abstract)*	2	5.88%
bb *(bibliography entry)*	1	2.94%
bdy *(body)*	1	2.94%
bib *(bibliography)*	1	2.94%
fig *(figure)*	1	2.94%
vt *(vita)*	1	2.94%

Table 1: Frequency of requested elements in the 34 CAS queries of INEX 2004.

Note that the requested element is not strictly enforced, but merely regarded as a retrieval hint.[2] For instance, paragraphs may be judged relevant answers to a query of the form `//sec[about(.,'xxx')]`. Hence, it is of interest to look at the tag names of elements that are judged relevant for the respective queries.

Elements Judged Relevant. We use version 3.0 of the assessments, containing judgments for the 26 queries numbered 127–137, 139–145, 149–153, and 155–157. Moreover, we focus on elements rated as highly exhaustive and highly specific—also called strict or (3,3) assessments. For the 4 queries numbered 133, 140, 143, and 144, there are no elements judged as highly exhaustive and highly specific. Table 2 lists the frequencies of element-types judged relevant for the remaining 22 CAS queries. We collapse the tag equivalences for sections and paragraphs, as defined in [20]. We use sec+ and p+ to denote the equivalence classes of sections and paragraphs, respectively.

Element	Frequency	Percentage
p+	854	31.41%
vt	747	27.47%
sec+	262	9.64%
au	110	4.05%
bb	104	3.82%
fnm	104	3.82%
st	90	3.31%
article	73	2.68%
fig	53	1.95%
it	37	1.36%
bdy	36	1.32%
ref	34	1.25%
scp	32	1.18%
atl	23	0.85%
abs	13	0.48%
fm	11	0.40%
b	10	0.37%

Table 2: Frequency of elements judged relevant for all assessed CAS queries at INEX 2004. We only show tag names that occur at least 10 times.

Requested versus Relevant Elements. Next, we investigate how often the element that is judged relevant actually has the tag name specified by the query. Consider Table 3; the rows show the tag names of requested elements as stated in the query and the columns show the tag names of elements judged relevant. E.g., if we look at the assessments of topics requesting sections (sec), we see that 27.7% of the relevant elements are sections (sec+), 24.7% are paragraphs (p+), and 3.3% are articles.[3] We conclude that the element names as requested in the query can indeed only be considered as a retrieval *hint*, and not as a strict constraint on the output of a query. While not strictly enforced, however, there seems to be a preference for the type of XML elements satisfying

	article	sec+	p+	abs	vt
article (2)	10.8%	1.3%	1.6%	–	–
sec (10)	3.3%	27.7%	24.7%	0.9%	0.4%
p (4)	4.0%	26.0%	48.0%	–	–
abs (2)	16.0%	–	24.0%	24.0%	–
vt (1)	–	–	44.0%	–	52.0%

Table 3: Frequency of relevant elements (columns) for queries asking for elements with tag name (rows). The number of aggregated queries is indicated between brackets.

[2]At INEX, relevance is assessed on the basis of the narrative describing the underlying information need [10, p. 237]: "CAS queries are topic statements, which contain explicit references to the XML structure, and explicitly specify the contexts of the user's interest (e.g. target elements) and/or the contexts of certain search concepts (e.g. containment conditions). ... Although users may think they have a clear idea of the structural properties of the collection, there are likely to be aspects to which they are unaware. The idea ... is to allow the evaluation of XML retrieval systems ... where not only the content conditions within a user query are treated with uncertainty but also the expressed structural conditions. ... The path specifications should therefore be considered hints as to where to look."

[3]The surprising numbers for the article topics are due to strange assessments of one of the article topics, which is most likely due to a misinterpretation of the assessment guidelines.

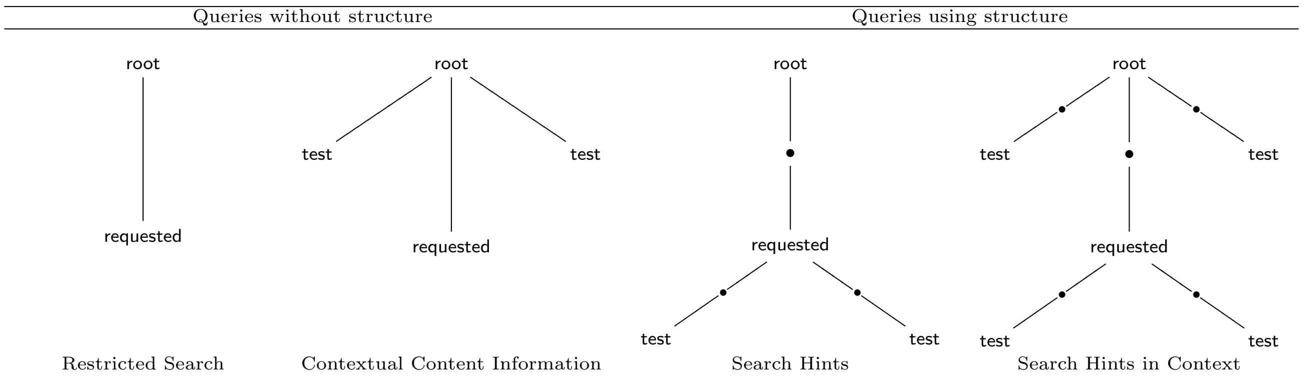

Figure 1: The tree shapes of the queries.

it. E.g., if people ask for sections, they are more likely to judge sections as relevant than any other kind of tag.[4]

2.4 How Structure is Used

To see how users use structure in their queries, we break down the set of queries by increasing complexity. This results in four categories, to be discussed below and graphically depicted in Figure 1; note that hierarchical information about the documents is only used in the last two categories.

Restricted Search. This category has queries in which structure is only used as a constraint on the returned elements. The query is an ordinary content-only query, but the search is restricted to particular XML elements. A typical example of such a query is to restrict the search to sections:

```
//sec[about(.,'xxx')].
```

In general, such queries have the form `//tag[P]` with `P` a positive boolean combination of functions `about(.,'xxx')`.

Contextual Content Information. This category is similar to the *Restricted Search* category, but additionally we may put content restrictions on the environment in which the requested element occurs. A typical example looks like:

```
//sec[about(.,'yyy') and about(//abs,'xxx')].
```

This query asks for sections about `yyy` in documents which contain an abstract (`abs`) about `xxx`. In general, such queries have the form `//tag[P]`, with `P` a positive boolean combination of functions `about(.,'xxx')` and `about(//tag,'xxx')`. Note that `about(//abs, 'xxx')` expresses that somewhere below the root of the document there is an abstract (`abs`) which is about `xxx`.

Search Hints. This category is again similar to the *Restricted Search* category, but additionally we may put content restrictions on subelements of the requested element, and we may use the hierarchical nature of the documents. These extra restrictions can be viewed as search hints or retrieval cues to the system. A typical example is a query which asks for sections about `xxx` containing a theorem about `yyy`:

```
//sec[about(.,'xxx') and about(.//thm,'yyy')].
```

[4]Note, especially for the granularity constraints `abs` and `vt`, that we do not distinguish between paragraphs appearing within or outside the element matching the granularity constraint.

The general form of such queries is `path[P]` with `P` a positive boolean combination of functions `about(.,'xxx')` and `about(.path ,'xxx')`, and `path` a location path sequence of the form `//tag`$_1$`//...//tag`$_n$.

Search Hints in Context. This category combines the *Search Hints* with the *Contextual Context Information* categories. An example is a query which asks for sections about `xxx` containing a theorem about `yyy`, in documents which contain an abstract (`abs`) about `zzz`:

```
//sec[about(.,'xxx') and about(.//thm,'yyy')
          and about(//abs,'zzz')].
```

The general form of queries of this category is `path[P]` with `P` a positive boolean combination of `about(.,'xxx')`, `about(.path,'xxx')` and `about(path ,'xxx')`, and `path` a location path sequence of the form `//tag`$_1$`//...//tag`$_n$.

To help situate the query categories just introduced, we recall the XML fragments proposed by Carmel et al. [3, 4] as a simple alternative to XPath for content and structure queries. Using the intuitive query-by-example underlying XML Fragments, only the *Restricted Search* and *Search Hint* categories can be expressed. For capturing queries in the other categories, a syntactic device is introduced [3]. Our approach differs from XML fragments in our focus on the descendant axis instead of the child axis, and our distinction between users having varying degrees of knowledge about valid tag nestings. E.g., *Contextual Content Information* can only be correctly specified in XML fragments using additional knowledge of the DTD.

Returning to our 34 CAS queries, we provide a classification in terms of our four categories in Table 4. We based this classification not on the actual syntactic shape of the queries, but on the fact whether they could equivalently be expressed in the query format of the category.

Category	Fraction	Query number
Restricted Search	15%	127,136,142,143,152
Contextual Content Information	47%	128, 129, 130, 131, 132, 134, 135, 137, 138, 141, 144, 145, 151, 158, 159, 160
Search Hints	6%	147, 153
Search Hints in Context	32%	133, 139, 140, 146, 149,150 154, 155, 156, 157, 161

Table 4: Classification of CAS queries.

As to the first research question in the introduction (How do users exploit the additional expressive power of structural constraints in their queries?), our main finding is that 62% of the 34 CAS queries does *not* use the hierarchical structure of the documents. The remaining 38% only uses the NEXI query language in a rather restricted form. That is, the hierarchical nature of the documents is used in less than half of the queries we examined. However, we also see that no less than 79% of the queries use contraints on the context of the elements to be returned. These contextual constraints cannot be captured by ordinary keyword queries.

3. THE EFFECT OF STRUCTURE ON RETRIEVAL EFFECTIVENESS

In this section we answer the second question from the introduction: What is the effect on retrieval performance of adding structural constraints to queries?

3.1 Experimental Setup

We base our experimental evidence on the INEX test-suite for the vague content and structure task. The difference with earlier results [19] is that none of the structural constraints was enforced, they were treated as search hints. We treat only highly specific and highly exhaustive elements as relevant (i.e., the so-called strict assessments) and use `trec_eval` for computing the scores.

Our aim is to contrast structured queries with keyword queries. That is, from a *structured query* like

```
//article[about(.,'sorting')]//sec[about(.,'heap sort')]
```

we simply collect all the content terms

```
sorting heap sort.
```

We refer to the latter type of representation of the information need as *full content query.* We create two runs:

Content oriented This run is based on the full-content query, and uses a language model approach together with query expansion [16]; we refer to [18] for details.

XPath oriented This run is based on the structured query, and target constraints are interpreted as strict; we refer to [19] for details.

3.2 Results

Average Precision. We first consider the results in terms of mean average precision (MAP). Table 5 shows the respective

	CO	XPath	Change
MAP	0.1377	0.0811	−41.1%

Table 5: Mean average precision.

scores. The content oriented (CO) run is clearly superior. This is, indeed, a disappointing result because the poorer scoring XPath-oriented run uses a more articulate query. Let us take a closer look and zoom in on the performance at different recall levels. Figure 2 shows the interpolated precision at the eleven standard recall levels. We see an interesting phenomenon. While the content-oriented run clearly outperforms the XPath-oriented run on higher recall levels, the XPath-oriented run outperforms the content oriented run on lower recall levels.

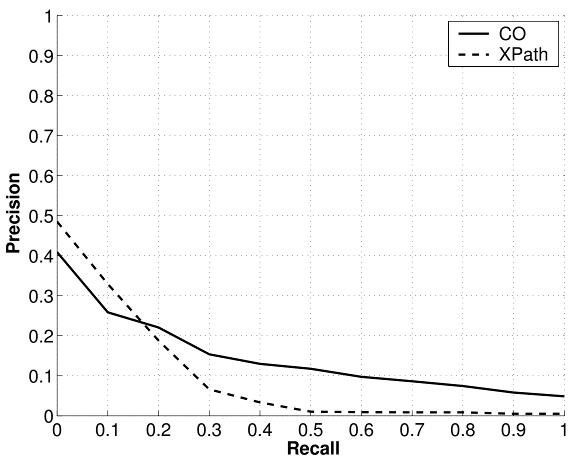

Figure 2: Interpolated precision at standard recall levels.

Initial Precision. Let us zoom in further and look explicitly at the performance on the initially retrieved elements. Table 6 shows the mean precision at ranks 5, 10, 20, and 30. Here, we see a complete reversal from the picture in Table 5: now, the XPath-oriented runs are superior. We zoom

Precision	CO	XPath	Change
@ 5	0.2091	0.3091	+47.8%
@ 10	0.1818	0.2591	+42.5%
@ 20	0.1614	0.1977	+22.5%
@ 30	0.1439	0.1788	+24.3%

Table 6: Mean precision at rank 5, 10, 20, and 30.

in even further and look solely at the first relevant element retrieved. Table 7 shows the mean reciprocal rank (MRR) of the first found relevant element. The outcome confirms the early precision results: the XPath-oriented run is superior to the content-oriented run.

	CO	XPath	Change
MRR	0.3778	0.4667	+23.5%

Table 7: Mean reciprocal rank scores.

Success. Next, we shift our focus to topics for which we do not score well. Does XPath-oriented querying help us score well on a larger set of topics, or does it simply improve our score where we did well already? Table 8 shows the mean

Success	CO	XPath	Change
@ 1	0.3182	0.3182	0.0%
@ 5	0.4091	0.5909	+44.4%
@ 10	0.5455	0.7273	+33.3%
@ 20	0.6364	0.7273	+14.3%

Table 8: Mean success at rank 1, 5, 10, and 20.

success—the fraction of topics for which at least one relevant element is found—at ranks 1, 5, 10, and 20. Again, the results are in favor of the XPath-oriented run. At rank 10, we fail to retrieve a relevant element for 27% of the topics.

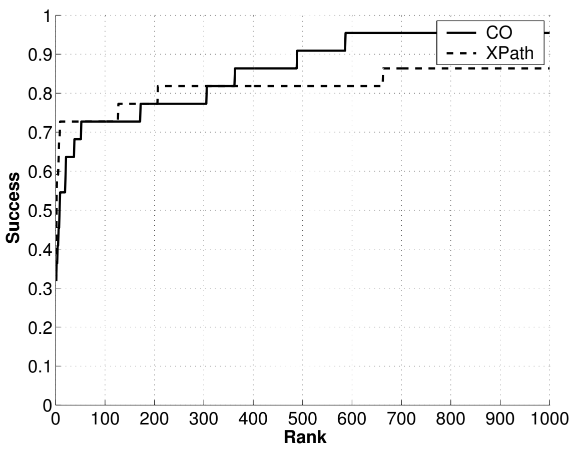

Figure 3: Success at n.

Recall that the task at hand is XML element retrieval, where each of the millions of XML elements may be relevant.

There is no such thing as mean average success (since it would trivially approximate the best success score at any rank if we average over an infinite number of ranks), but we can investigate the increase of success over ranks. Figure 3 shows the success over the first 1000 ranks. We see a picture similar to the interpolated precision curves in Figure 2: at lower ranks, the XPath-oriented run finds a relevant element for a larger fraction of topics, but at higher ranks the content-oriented run wins.

In sum, our results show that structured queries do *not* lead to improved mean average precision scores; in fact, we see a substantial drop in mean average precision. However, this can be attributed completely to poor scoring at higher recall levels. If we zoom in on the initially retrieved elements, or on the first found relevant element, the outcome is reversed: structured queries lead to substantially better initial precision scores. The experimental evidence indicates that structured queries function as a precision enhancing device: useful for promoting the precision of initially retrieved documents, possibly reducing fall-out but also reducing recall.

4. QUERY LANGUAGES FOR CONTENT AND STRUCTURE QUERIES

As stated previously, the NEXI query language is an extension of a subset of XPath. The motivation for restricting XPath is that users find it hard to state their information need in XPath and tend to make semantic mistakes in their query formulations. We analyze why users make such mistakes and build a user model. Then we show that the NEXI query language is a perfect fit for this user model: on the one hand, users cannot make the found semantic mistakes because the language is restricted (the language is *safe*); on the other, they can express every information need belonging to this user model (the language is *complete*).

4.1 Less Power is Better

At INEX, the focus is on retrieving sets of elements from document-centric XML using information about the content of the elements and their location in the documents. For this reason, it was decided to restrict the query language to the navigational part of XPath 1.0; in [7] this language is defined as Core XPath. The only objects which are manipulated in this language are sets of nodes (i.e., there are no arithmetical or string operations). Besides these restrictions, the full power of location paths is supported (except for namespace and attribute axis), including filter expressions being closed under the boolean operators. At INEX 2003, Core XPath expanded with the `about` function was used as a query language. The results were disappointing: many queries did not match the information need as described in the narrative and description part; often, the information need was much broader than the XPath expression [15]. The most typical mistake was the use of / (child axis) where // (descendant) was intended. These semantic mistakes can likely be attributed to the fact that users have no, or at best incomplete, knowledge of the structure of documents, that is, of the DTD (see Section 2.1 for details on the INEX collection). To reduce the chance of making such semantic mistakes, O'Keefe and Trotman [15] argued that apart from the descendant axis no other axis relations should be used in queries. This recommendation was implemented in the INEX 2004 NEXI query language (described in Section 2). We will provide a theoretical basis for this recommendation by giving a mathematical model of a user's knowledge of a DTD and relating the expressive power of the NEXI query language to this model.

A user's knowledge about a set of XML documents can be naturally formalized in terms of an *indiscernibility relation* between documents. For instance, most INEX users will not distinguish the two documents in Figure 4 solely based on the tag names. For a user, two indiscernible documents are the same, and a query should return the same answers from both documents. But there are XPath queries which return different answers on these two documents. This is the reason for considering weaker fragments of XPath, those for which indiscernible documents yield identical answers. Naturally, given an indiscernibility relation, we would like those fragments to be as large as possible.

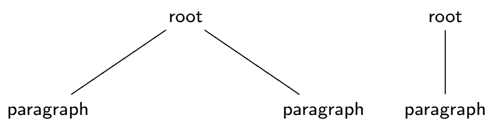

Figure 4: Two simple XML document trees.

In sum, given an indiscernibility relation, there are two competing forces: *safety*, which reduces expressive power, and *completeness*, which asks for as much expressivity as possible. In a safe query language, users cannot write queries which return different answers from documents which these users consider to be the same. A safe language is designed to avoid making semantic mistakes by forbidding the user to pose such queries. We will shortly see that the NEXI language is an example of a safe and complete query language.

4.2 User Profiles

Below, we define two user profiles in terms of indiscernability relations, 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 and complete for these profiles.

Ignorant Users. Users formulating queries at INEX did not have a clear idea of the DTD of the collection [15]. Typically, they browsed the documents and picked up some knowledge about the available tags in this manner. Their queries can be viewed as an XML version of fielded search. 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. The typical queries of an ignorant user are the *Restricted Search* and *Contextual Content Information* queries from Section 2.4.

The following syntax, which we call *structure unaware XPath*, allows us to pose these queries. 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` and `self::tag`. Note that when `//tag` is used in a filter it means "there exists a descendant of the root labeled `tag`". I.e., `//tag` simply says that somewhere in the document there is a `tag` element. `self::tag` expresses that the current node is labeled by `tag`.

We turn to a semantic characterization of this fragment. In social network theory [25] 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 binary relation between the elements of D and D' connecting the roots. We call B a *structure unaware bisimulation* if, whenever xBy holds for two elements x, y in D, 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 $x'By'$; and
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, the relation which connects the roots and the paragraph elements in Figure 4 to each other is a bisimulation. Secondly, first-order formulas in one free variable can be seen as an alternative stronger query language than XPath (for the relative expressive power of the two cf., [12]). Thirdly, 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. *Let D, D' be documents and B a structure unaware bisimulation. Then any structure unaware XPath expression yields the same answer set on D and D'.*
2. *For every first-order formula that is invariant under structure unaware bisimulations there exists an equivalent structure unaware 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.*

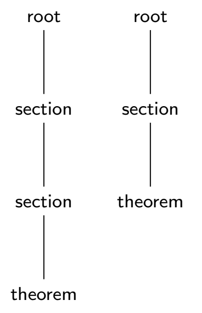

Figure 5: Document trees that do not bisimulate.

Semi-ignorant Users. Semi-ignorant users have some clue about the hierarchical structure of the documents. E.g., they know that paragraphs are below sections, but need not know that there *can* be elements in between [15]. For this reason, O'Keefe and Trotman [15] proposed 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`". Note that all types of queries discussed in Section 2 can be formulated in this fragment. As this XPath fragment does not contain negation, bisimulation is too strong a notion [11]. 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 3. Let D, D' be documents and B a binary relation between the elements of D and D' connecting the roots. We call B a *vertical 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 $x'By'$; and
3. similarly when $x' < x$.

Vertical 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 4. *Let X be a set of elements. The following are equivalent:*

1. *X is definable by a first-order formula in one free variable which is preserved under vertical simulations.*
2. *X is definable as the answer set of a union of Positive Descendant XPath formulas.*

The proof uses ideas from modal logic [2, Theorem 2.78] together with ideas from [1, Theorem 3.2].

Descendant or Descendant-or-self?. Positive Descendant XPath has great syntactic appeal because the only operator is //. It is a natural fragment because it corresponds exactly to the semi-ignorant users. Still, one could argue that it is too expressive for these users. Consider the two document trees in Figure 5. There are no vertical simulations between these two. But, according to the data and the arguments in

[15], INEX users consider them to be the same. We can easily adjust our notion of simulation to cater for this: instead of simulating the descendant relation $<$, only simulate the descendant-or-self relation $\leq$. Then, these two documents bisimulate! Unfortunately, there is no appealing abbreviated syntax for the corresponding query language ("Positive Descendant-or-Self XPath"), but probably this language can further reduce the number of semantic mistakes.

5. DISCUSSION AND CONCLUSIONS

We can identify a number of important lessons for future work in information retrieval from document-centric XML collections. Simply combining powerful XML query languages with IR-style retrieval and ranking of results does not work. The addition of structure to queries is not a simple recipe for improving results. This is in line with earlier work: the use of structure in queries has been studied extensively; prominent examples include booleans, proximity and phrase operators. In early publications, the usage of phrases and proximity operators—as well as a careful usage of boolean operators—showed improved retrieval results but rarely anything substantial [e.g., 5]. As retrieval models became more advanced, the usage of query operators was questioned. E.g., Mitra et al. [14] conclude that when using a good ranking algorithm, phrases have no effect on high precision retrieval (and sometimes a negative effect due to topic drift). Rasolofo and Savoy [17] combine term-proximity heuristics with an Okapi model, obtaining 3%–8% improvements for Precision@5, 10, 20, with hardly observable impact on the MAP scores.

For XML retrieval we draw the following conclusions. First, as observed in [15], less expressivity is better in that it reduces the chance of making semantic mistakes. We have shown that the proposed NEXI query language [15] is not ad hoc, but has a precise mathematical chracterization in terms of an intuitive user model. Secondly, users tend not to use structure in their queries. Although structure is used in varying degrees of complexity, over half of the queries can be expressed in a the very restrictive *ignorant user* language. We also found that structure is used as a search hint, and not as a strict search requirement, when judged against the underlying information need. As a consequence, we hypothesized that the use of structure in queries functions as a precision enhancing device.

To test this hypothesis we conducted a set of experiments. The outcomes confirm that structured queries function as a precision enhancing device: useful for promoting the precision of initially retrieved documents, possibly reducing fallout but also reducing recall. Structured queries can be a powerful tool, catering for the typical web searcher who is interested solely in the precision of the first handful of documents. As the INEX Interactive Track revealed, users rarely look beyond the first handful of returned elements [22].

Acknowledgments. We thank our referees and the participants of the Topic Format Working Groups at INEX 2002, 2003, and 2004.

This research was supported by the Netherlands Organization for Scientific Research (NWO) under project numbers 017.001.190, 220-80-001, 264-70-050, 365-20-005, 612.-000.106, 612.000.207, 612.013.001, 612.066.302, 612.069.006, and 640.001.501.

REFERENCES

[1] M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. In *Proc. ICDT*, 2003.

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

[3] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML documents via XML fragments. In *Proc. SIGIR*, pages 151–158, 2003.

[4] D. Carmel, Y. S. Maarek, Y. Mass, N. Efraty, and G. M. Landau. An extension of the vector space model for querying XML documents via XML fragments. In *Proceedings SIGIR 2002 Workshop on XML and Information Retrieval*, pages 14–25, 2002.

[5] J. Fagan. Experiments in automatic phrase indexing for document retrieval: A comparison of syntactic and non-syntactic methods. Technical report, Cornell University, 1987.

[6] N. Fuhr, M. Lalmas, S. Malik, and Z. Szlávik, editors. *INEX 2004 Workshop Pre-Proceedings*, 2004.

[7] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *VLDB'02*, 2002.

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

[9] INEX. INitiative for the Evaluation of XML Retrieval, 2004. `http://inex.is.informatik.uni-duisburg.de:2004/`.

[10] G. Kazai, M. Lalmas, and B. Piwowarski. INEX 2004 relevance assessment guide. In Fuhr et al. [6], pages 241–248.

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

[12] M. Marx and M. de Rijke. Semantic Characterizations of Navigational XPath. *ACM SIGMOD Record*, 34(2):41–46, 2005.

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

[14] M. Mitra, C. Buckley, A. Singhal, and C. Cardie. An analysis of statistical and syntactic phrases. In *Proc. RIAO-97*, 1997.

[15] R. A. O'Keefe and A. Trotman. The simplest query language that could possibly work. In *Proceedings of the 2nd INEX Workshop*, 2004.

[16] J. Ponte. Language models for relevance feedback. In W. Croft, editor, *Advances in Information Retrieval*, chapter 3, pages 73–96. Kluwer, 2000.

[17] Y. Rasolofo and J. Savoy. Term proximity scoring for keyword-based retrieval systems. In *Proc. ECIR 2003)*, pages 207–218, 2003.

[18] B. Sigurbjörnsson, J. Kamps, and M. de Rijke. The University of Amsterdam at INEX 2004. In Fuhr et al. [6], pages 104–109.

[19] B. Sigurbjörnsson, J. Kamps, and M. de Rijke. Processing content-oriented XPath queries. In *Proc. CIKM 2004*, pages 371–380. ACM Press, 2004.

[20] B. Sigurbjörnsson, B. Larsen, M. Lalmas, and S. Maalik. INEX04 guidelines for topic development. In Fuhr et al. [6], pages 219–236.

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

[22] A. Tombros, B. Larsen, and S. Malik. The interactive track at INEX 2004. In Fuhr et al. [6], pages 24–29.

[23] A. Trotman and B. Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In Fuhr et al. [6], pages 219–236.

[24] V. Vianu. A Web odyssey: from Codd to XML. In *Proc. PODS*, pages 1–15. ACM Press, 2001. ISBN 1-58113-361-8.

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