XXL @ INEX 2003

Ralf Schenkel
schenkel@mpi-sb.mpg.de

Anja Theobald
anja.theobald@mpi-sb.mpg.de

Gerhard Weikum
weikum@mpi-sb.mpg.de

Max–Planck Institut für Informatik
Saarbrücken, Germany

ABSTRACT

Information retrieval on XML combines retrieval on content data (element and attribute values) with retrieval on structural data (element and attribute names). Standard query languages for XML such as XPath or XQuery support Boolean retrieval: a query result is a (possibly restructured) subset of XML elements or entire documents that satisfy the search conditions of the query. Such search conditions consist of regular path expressions including wildcards for paths of arbitrary length and boolean content conditions.

We developed a flexible XML search language called XXL for probabilistic ranked retrieval on XML data. XXL offers a special operator '∼' for specifying semantic similarity search conditions on element names as well as element values. Ontological knowledge and appropriate index structures are necessary for semantic similarity search on XML data extracted from the Web, intranets or other document collections. The XXL Search Engine is a Java–based prototype implementation that support probabilistic ranked retrieval on a large corpus of XML data.

This paper outlines the architecture of the XXL system and discusses its performance in the INEX benchmark.

1. INTRODUCTION

The main goal of the initiative for the evaluation of XML retrieval (INEX) is to promote the evaluation of content–based and structure–based XML retrieval by providing a hugh test collection of scientific XML documents, uniform scoring procedures, and a forum for organisations to compare their results. For that purpose, the INEX committee provides about 12.000 IEEE journal articles with a rich XML structure. In cooperation with the participanting groups a set of content–only queries (CO) and a set of content–and–structure queries (CAS) was created. Each group evaluated these queries on the given data with their XML retrieval system and submitted a set of query results.

In this paper we describe the main aspects of our XXL search engine. First of all, we present our flexible XML search language *XXL*. In addition, we describe our ontology model which we use for semantic similarity search on structural data and content data of the XML data graph. Then we give a short overview how XXL queries are evaluated in the XXL Search Engine and which index structures used to support an efficient evaluation. Finally, we present the our results in the INEX 2003 benchmark.

2. XML DATA MODEL

In our model, a collection of XML documents is represented as a directed graph where the nodes represent elements, attributes and their values. For identification, each node is assigned a unique ID, the `oid`. There is an directed edge from a node x to a node y if

- y is a subelement of x,
- y is an attribute of x,
- y contains the value of element x or
- y contains the value of attribute x.

Additionally, we model an XLink [7] from one element to another by adding a special, directed edge between the corresponding nodes. We call the resulting graph the *XML data graph* for the collection.

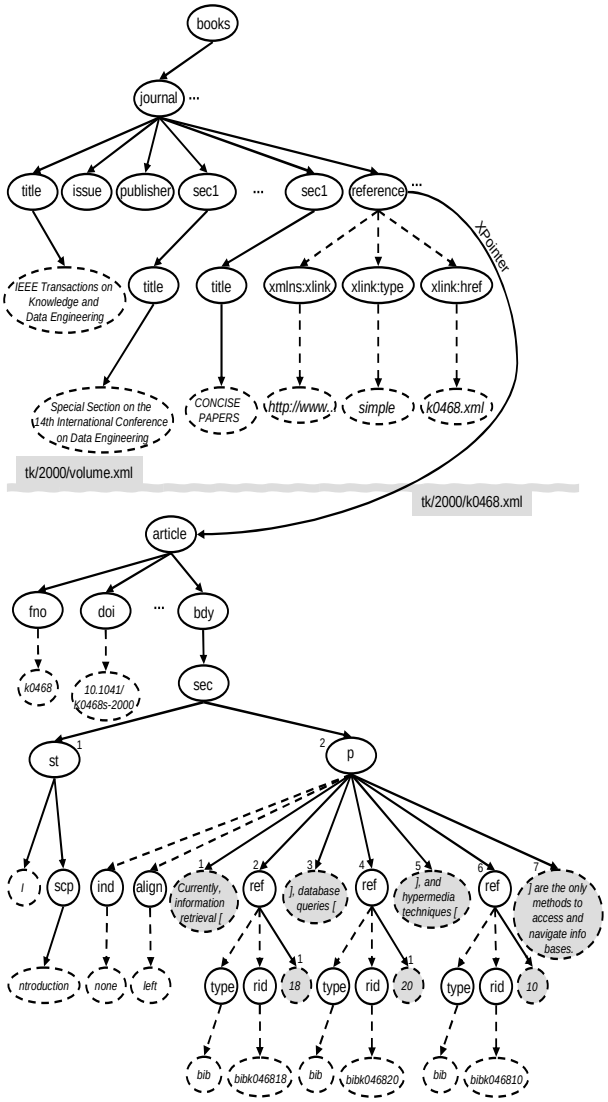


Figure 1: XML data graph

Figure 1 shows the XML data graph for a collection of two XML documents from the INEX collection (adapted as shown in Section 6): a journal document with an XLink pointing to an article document. Each node that contains an element or attribute name is called *n–node* (shown as normal nodes in Figure 1), and each node that contains an element or attribute value is called *c–node* (dashed nodes in Figure 1). To represent mixed content, we need a local order of the child nodes of a given element. In Figure 1 you can see a sentence which is partitioned into several shaded c–nodes.

3. THE FLEXIBLE XML QUERY LANGUAGE XXL

The Flexible XML Search Language XXL has been designed to allow SQL-style queries on XML data. We have adopted several concepts from XML-QL [8], XQuery[3] and similar languages as the core, with certain simplifications and resulting restrictions, and have added capabilities for ranked retrieval and ontological similarity. As an example for an XXL query, consider the following query that searches for publications about astronomy:

```
SELECT $T              // output of the XXL query
FROM   INDEX           // search space
WHERE  ~article AS $A  // search condition
  AND  $A/~title AS $T
  AND  $A/#/~section ~ "star | planet"
```

The `SELECT` clause of an XXL query specifies the output of the query: all bindings of a set of element variables. The `FROM` clause defines the search space, which can be a set of URLs or the index structure that is maintained by the XXL engine. The `WHERE` clause specifies the search condition; it consists of the logical conjunction of *path expressions*, where a path expression is a regular expression over *elementary conditions* and an elementary condition refers to the name or content of a single element or attribute. Regular expressions are formed using standard operators like `'/'` for concatenation, `'|'` for union, and `'*'` for the Kleene star. The operator `'#'` stands for an arbitrary path of elements. Each path expression can be followed by the keyword `AS` and a variable name that binds the end node of a qualifying path (i.e., the last element on the path and its attributes) to the variable, that can be used later on within path expressions, with the meaning that its bound value is substituted in the expression.

In contrast to other XML query languages we introduce a new operator `'~'` to express semantic similarity search conditions on XML element (or attribute) names as well as on XML element (or attribute) contents.

The result of an XXL query is a subgraph of the XML data graph, where the nodes are annotated with local relevance probabilities called similarity scores for the elementary search conditions given by the query. These similarity scores are combined into a global similarity score for expressing the relevance of the entire result graph. Full details of the semantics of XXL and especially the probabilistic computation of similarity scores can be found in [17, 18].

4. ONTOLOGY–BASED SIMILARITY

Ontologies have been used as a means for storing and retrieving knowledge about the words used in natural language and relations between them.

In our approach we consider a term t as a pair $t = (w, s)$ where w is a word over an alphabet Σ and s is the word sense (short: sense) of w, e.g.

```
t1 = (star, a celestial body of hot gases)
t2 = (heavenly body, a celestial body of hot gases)
t3 = (star, a plane figure with 5 or more points)
```

In order to determine which terms are related, we introduce semantic relationships between terms that are derived from common sense. We say that a term t is a *hypernym* (*hyponym*) of a term t' if the sense of t is more general (more specific) than the sense of t'. We also consider holonyms and meronyms, i.e., t is a *holonym* (*meronym*) of t' if t' means something that is a part of something meant by t (vice versa for meronyms). Finally, two terms are called *synonyms* if there senses are identical, i.e., their meaning is the same.

Based on these definitions we now define the ontology graph $O = (V_O, E_O)$ which is a data structure to represent concepts and relationships between them. This graph has concepts as nodes and an edge between two concepts whenever there is a semantic relationship between them. In addition, we label each edge with a weight and the type of the underlying relationship. The weight expresses the semantic similarity of two connected concepts. Figure 2 shows an excerpt of an example ontology graph around the first sense for the word "star".

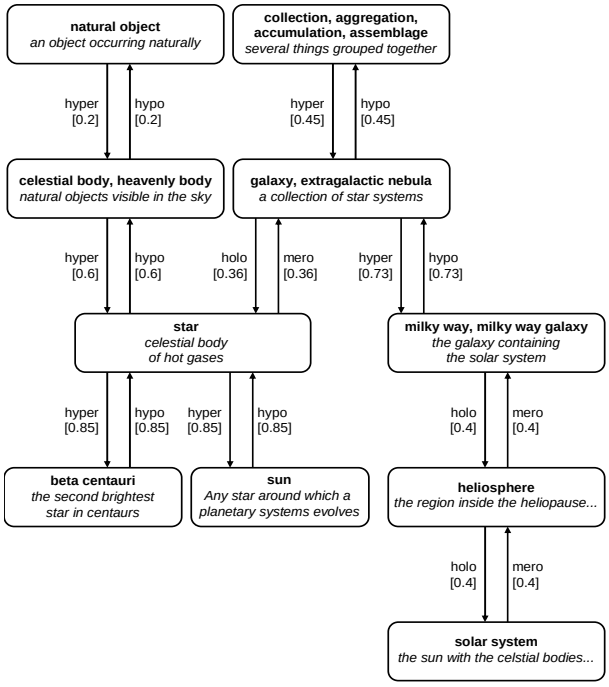


Figure 2: Excerpt of an ontology graph O with labeled edges

To fill our ontology with concepts and releationship we use the voluminous electronical thesaurus WordNet as backbone. WordNet organzies words in synsets and presents relationships between synsets without any quantification.

For quantification of relationships we consider frequency–based correlations of concepts using large web crawls. In our approach, we compute the similarity of two concepts using correlation coefficients from statistics, e.g. the Dice or Overlap coefficient [14].

For two arbitrary nodes u and v that are connected by a path $p = \langle u = n_0 \dots n_k = v \rangle$, we define the similarity $sim_p(u, v)$ of the start node u and the end node v along this path to be the product of the weights of the edges on the path:

$$sim_p(u,v) = \prod_{i=1}^{length(p)-1} weight(\langle n_i, n_{i+1}\rangle)$$

where $weight(\langle n_i, n_{i+1}\rangle)$ denotes the weight of the edge $e = (n_i, n_{i+1})$. The rationale for this formula is that the length of a path has direct influence on the similarity score. The similarity $sim(u,v)$ of two nodes u and v is then defined as the maximal similarity along any path between u and v:

$$sim(u,v) = \max\{sim_p(u,v) \mid p \text{ path from } u \text{ to } v\}$$

However, the shortest path (the path with the smallest number of edges) need not always be the path with the highest similarity, as the triangular inequation does not necessarily hold. Thus, we need an algorithm that takes into account all possible paths between two given concepts, calculates the similarity scores for all paths, and chooses the maximum of the scores for the similarity of these concepts. This is a variant of the single–source shortest path problem in a directed, weighted graph. A good algorithm to find the similar concepts to a given concept and their similarity scores is a variant of Dijkstra's algorihm [6] that takes into account that we multiply the edge weights on the path and search for the path with the maximal weight instead of minimal weight.

Furthermore, as words may have more than one sense, it is a priori not clear in which sense a word is used in a query or in a document. To find semantically similar words, it is fundamental to disambiguate the word, i.e., to find out its current sense. In our work we compute the correlation of a context of a given word and the context of a potential appropriate concept from the ontology using correlation coefficients as described above. Here, the context of a word are other words in the proximity of the words in the query or document, and the context of a concept is built from the words of the neighbor nodes of the concept. See [15] for more techical details on the disambiguation process.

5. THE XXL SEARCH ENGINE

5.1 Architecture of the XXL Search Engine

The XXL Search Engine is a client-server system with a Java-based GUI. Its architecture is depicted in Figure 3.

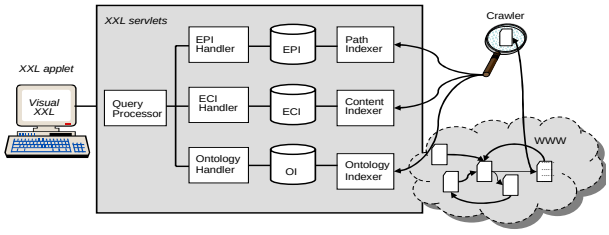


Figure 3: Architecture of the XXL search engine

The server consists of the following core components:

- *service components*: the crawler and the query processor, both Java servlets
- *algorithmic components*: parsing and indexing documents, parsing and checking XXL queries
- *data components*: data structures and their methods for storing various kinds of information like the element path index (EPI), the element content index (ECI), and the ontology index (OI).

The EPI contains the relevant information for evaluating simple path expressions that consist of the concatenation of one or more element names and path wildcards #. The ECI contains all terms that occur in the content of elements and attributes, together with their occurrences in documents; it corresponds to a standard text index with the units of indexing being elements rather than complete documents. The OI implements the ontology graph presented in Section 4.

5.2 Query Processing in the XXL Search Engine

The evaluation of the search conditions in the Where clause consists of the following two main steps:

- The XXL query is decomposed into subqueries. A global evaluation order for evaluating the various subqueries and a local evaluation order in which the components of each subquery are evaluated are chosen.
- For each subquery, subgraphs of the data graph that match the query graph are computed, exploiting the various indexes to the best possible extent. The subresults are then combined into the result for the original query.

5.2.1 Query Decomposition

As an example for an XXL query, consider the following XXL query where we are interested in scientific articles about information retrieval and databases:

```
SELECT $T
FROM   INDEX
WHERE ~article AS $A
  AND  $A/~title AS $T
  AND  $A/#/~section ~ "IR & database"
```

The Where clause of an XXL query consists of a conjunction `"W1 And ... And Wn"` of subqueries `Wi`, where each subquery has one of the following types:

- `Pi`
- `Pi AS $A`
- `Pi ~|LIKE|=|<>|<|> condition`

where each `Pi` is a regular path expression over elementary conditions, `$A` denotes a element variable to which the end node of a matching path is bound, and `condition` gives a content–based search condition using a binary operator. From the definitions of variables we derive the *variable dependency graph* that has an edge from `$V` to `$W` if the path bound to `$W` contains `$V`. We require the variable dependency graph of a valid XXL query to be acyclic.

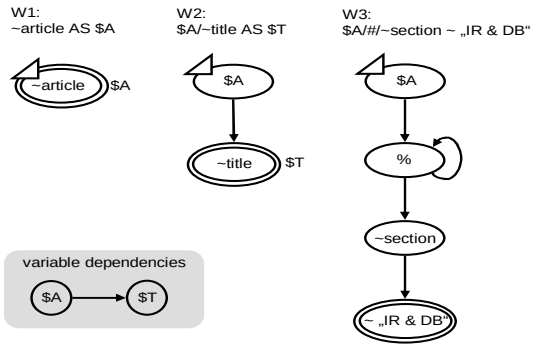


Figure 4: XXL search graphs for each subquery of the given XXL query

Each subquery corresponds to a regular expression over elementary conditions which can be described by an equivalent non-deterministic finite state automaton (NFSA). Figure 4 shows the search graphs of the example query together with the variable dependency graph.

5.2.2 Query Evaluation

To evaluate an XXL query, we first choose an order in which its subqueries are evaluated. This order must respect the variable dependency graph, i.e., before a subquery that defines a variable is evaluated, all subqueries that define variables used in this subquery must be evaluated. As this may still leaves us some choices how to order subqueries, we estimate the selectivity of each subquery using simple statistics about the frequency of element names and search terms that appear as constants in the subquery. Then we choose to evaluate subqueries and bind the corresponding variables in ascending order of selectivity (i.e., estimated size of the intermediate result).

Each subquery is mapped to its corresponding NFSA. A result for a single subquery, i.e. a *relevant path*, is a path of the XML data graph that matches a state sequence in the NFSA from an intial state to a final state. For such a result, the relevance score is computed by multiplying the local relevance scores of all nodes of the path. In addition, all variables that occur in the subquery are assigned to one node of the relevant path.

A result for the query is then constructed from a consistent union of the variable assignments and a set of relevant paths (one from each subquery) that satisfies the variable assignments. The global relevance for such a result is computed by multiplying the local relevances of the subresults.

The local evaluation order for a subquery specifies the order in which states of the subquery's NFSA are matched with elements in the XML data graph. The XXL prototype supports two alternative strategies: in top-down order the matching begins with the start state of the NFSA and then proceeds towards the final state(s); in bottom-up order the matching begins with the final state(s) and then proceeds towards the start state.

As an example, we show how the NFSA shown in Figure 5 is evaluated in top-down order on the data shown in that figure.

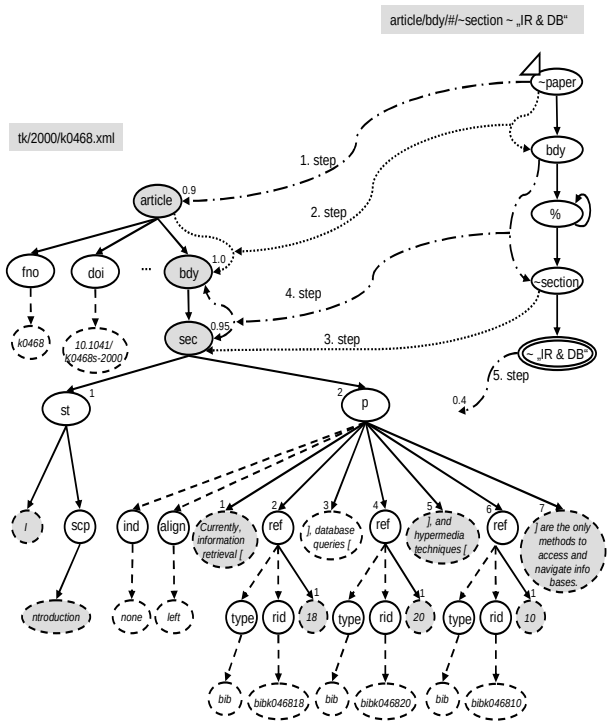


Figure 5: Evaluation of a XXL search graph in top–down manner

Step 1: The first elementary search condition contains a semantic similarity search condition on an element name. Thus, we consult the ontology index to get words which are similar to `paper`, yielding the word `article` with *sim(article, paper) = 0.9*. The first part of our result graph is therefore a n–node of the data graph named `article`, and it is assigned a local relevance score of 0.9.

Step 2: To be relevant for the query, a node from the result set of Step 1 must also have a child node with name `bdy`. As a result of Step 2, we consider result graphs formed by such nodes and their respective child.

Step 3: The next state in the NFSA corresponds to a wildcard for an arbitrary path in the data graph. Explicitly evaluating this condition at this stage would require an enumeration of the (possibly numerous) descendants of candidate results found so far, out of which only a few may satisfy the following conditions. We therefore proceed with the next condition in the NFSA and postpone evaluating the path wildcard to the next step. The following condition is again a semantic similarity condition, so we consult the ontology index to get words which are similar to `section`. Assume that the ontology index returns the word `sec` with a similarity score of 0.95. There are no n-nodes in the data that are named `section`, but we can add n–nodes named `sec` to our preliminary result with a local relevance score of 0.95.

Step 4: In this step we combine the results from steps 2 and 3 by combining n-nodes that are connected through an arbitrary path.

Step 5: The final state of the NFSA contains a content-based semantic similarity search condition which must be satisfied by the content of a `sec`-element in the result set of Step 4. We first decompose the search condition that may consist of a conjunction of search terms into the atomic formulas (i.e., single terms). For each atomic formula we consult the ontology index for similar words and combine them in a disjunctive manner. We then use a text search engine to evaluate the relevance of each element's content which is expressed through an tf/idf-based relevance score. This score is combined with the ontology-based similarity score to the relevance score of the atomic formula. Finally, we multiply the relevance scores for each formula to get the relevance score for the similarity condition.

In our example, the shaded nodes in Figure 5 form a relevant path for the given NFSA.

5.2.3 Index Structures

The XXL Search Engine provides appropriate index structures, namely the element path index (EPI), the element content index (ECI), and the ontology index (OI), that support the evaluation process described in the previous subsection.

The OI supports finding words that are semantically related to a given word, using the techniques presented in Section 4.

The ECI supports the evaluation of complex logical search conditions using an inverted file and a B+–tree over element names. Given an atomic formula, the ECI returns elements whose content is relevant with respect to that atomic formula and the tf/idf–based relevance score.

The EPI provides efficient methods to find children, parents, descendants and ascendants of a given node, and to test if two arbitrary nodes are connected. When the XML data graph forms a tree, we use the well-known pre- and postorder scheme by Grust et al. [10, 11] for this purpose. However, if the XML documents contain links, this scheme

can no longer be applied. For such settings that occur frequently with documents from the Web, the XXL Search Engine provides the *HOPI* index [16] that utilizes the concept of a 2–hop cover of a graph. This is a compact representation of connections in the graph developed by Cohen et al. [4]. It maintains, for each node v of the graph, two sets $L_{in}(v)$ and $L_{out}(v)$ which contain appropriately choosen subsets of the transitive predecessors and successors of v. For each connection (u, v) in the XML data graph G, we choose a node w on a path from u to v as a *center node* and add w to $L_{out}(u)$ and to $L_{in}(v)$. We can efficiently test if two nodes u and v are connected by checking $L_{out}(u)$ and $L_{in}(v)$: there is a path from u to v iff $L_{out}(u) \cap L_{in}(v) \neq \emptyset$. The path from u to v can be separated into a first hop from u to some $w \in L_{out}(u) \cap L_{in}(v)$ and a second hop from w to v, hence the name of the method.

More technical details how we improved the theoretical concept of a 2–hop–cover can be found in [16] which covers both the efficient creation of the index using a divide-and-conquer algorithm and the incremental maintenance of the index.

5.3 Implementation Issues

In our prototype implementation we store XML data in an Oracle 9i database with the following relational database schema:

- URLS (urlid, url, lastmodified),
- NAMES(nid, name),
- NODES(oid, urlid, nid, pre, post),
- EDGES(oid1, oid2),
- LINKS(oid1, oid2),
- CONTENTS(oid, urlid, nid, content),
- LIN (oid1, oid2) and
- LOUT(oid1, oid2).

Here, NODES, EDGES and CONTENTS store the actual XML data, URLS contains the urls of all XML documents known to the system, and LINKS holds the links between XML documents. LIN and LOUT store the L_{in} and L_{out} sets used by the HOPI index. The ECI makes use of Oracle's text search engine.

The OI is represented by the following three tables:

- CONCEPTS (cid, concept, description, freq),
- WORDS (cid, word) and
- RELATIONSHIPS(cid1, cid2, type, freq, weight).

The entries in the ontology index are extracted from the well–known electronic thesaurus WordNet [9]. Frequencies and weights are computed as shown in Section 4.

Both the crawler used to parse and index XML documents from the Web and from local directories and the query processor of the XXL search engine used to evaluate XXL queries are implemented using Java.

6. XXL AND THE INEX BENCHMARK

6.1 The INEX Data

The INEX document collection consists of eighteen IEEE Computer Society journal publications with all volumes since 1995. Each journal is stored in its own directory. For each journal, the volumes are organized in subdirectories per year. Each volume consists of a main XML file `volume.xml` that includes the XML files for the articles in this volume using XML entities. Thus, importing all volumes using a standard XML parser yields 125 single documents.

This organization of the data appears somewhat artificial and is unsuitable for answering INEX queries, as these queries typically ask for URLs of articles, not volumes. Having only volumes available as separate XML files, the path to the originating article for a hit has to be reconstructed from metadata in the XML files (the `fno` entries) which unfortunately is not always correct.

To overcome this problem, we adapted the INEX data in the following way. We replaced each entity in the volume files by an XLink pointing to the root element of the corresponding article. This modification keeps the original semantics of the data, but allows us to return the correct URLs of results in all cases. Additionally, such an organization is much closer to what one would expect from data available on the Web or in digital libraries. After this modification, importing all documents yielded 125 journal volumes and 12,117 journal articles.

The following table shows the number of records of each table after crawling and indexing the slightly modified INEX document collection.

table	number of records
URLS	12.232
NAMES	215
NODES	12.061.220
EDGES	12.048.987
LINKS	407.960
CONTENTS	11.779.730
LIN	28.776.664
LOUT	4.924.420

In addition to this structural problem, the INEX collection has some other properties that makes retrieval based on semantic similarities difficult, if not infeasible:

- Most element and attribute names are, even though they are derived from natural language, no existing words. As an example, the element name `sbt` stands for "'subtitle"'. However, the ontology used by XXL does not contain such abbrevations, so it had to be manually adapted if it was to be used for the INEX queries.
- Some element names are used only for formatting and do not carry any semantics at all. As an example, elements with name `scp` contain textual content that should be typeset small caps font.
- Each journal article has a rich structure with possibly long paths (which XXL supports with its highly efficient path index structures). However, as all articles are conforming to the same DTD, they share the same structure, which renders structural similarity search obsolete.
- The queries mostly contain keywords that are not well represented in WordNet, yielding ontology lookups useless in most cases. For some keywords, we manually enhanced the ontology, but this was far less complete than the information usually available with WordNet.

As a preliminary conclusion, the INEX collection is inappropriate for exploiting and stress–testing similarity search features as provided by our query language XXL and also other approaches along these lines [1, 5, 12].

6.2 The INEX Topics

The INEX benchmark consists of a set of content–only queries (CO) and content–and–structure queries (CAS) given in a predefined XML format. Each *topic* (INEX query) consists of a short description and a longer description of the topic of request and a set of keywords, and CAS queries also contain an XPath expression. For example, consider the CO–topic

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="98" query_type="CO" ct_no="26">
  <title>
    "Information Exchange", +"XML", "Information
     Integration"
  </title>
  <description>
    How to use XML to solve the information exchange
    (information integration) problem, especially in
    heterogeneous data sources?
  </description>
  <narrative>
    Relevant documents/components must talk about
    techniques of using XML to solve information
    exchange (information integration) among
    heterogeneous data sources where the structures
    of participating data sources are different
    although they might use the same ontologies
    about the same content.
  </narrative>
  <keywords>
    information exchange, XML, information integration,
    heterogeneous data sources
  </keywords>
</inex_topic>
```

To automatically transform a CO–topic into an XXL query we consider the keywords within the XML element `<title>` given for the query. As there is no way to automatically decice how to combine these keywords (conjunctively, disjunctively or mixed) in an optimal manner, we chose to combine them conjunctively. To get also results that are semantically similar to the keywords, we also add our similarity operator $\sim$. For the CO–topic 98 this process yields the following XXL query:

```
SELECT *
FROM INDEX
WHERE article/# ~  "(information exchange)
                    & XML
                    & (information integration)"
```

For CAS queries, we map the given XPath expression in a straightforward way to a corresponding XXL expression, adding semantic similarity conditions to all element names and keywords that appear in the XPath expression. However, as there are sometimes differences between the XPath expression and the natural language–based description of a query, this automatic transformation does not always yield optimal results. For the CAS–topic 63 this process yields the following XXL query:

```
SELECT $A
FROM INDEX
WHERE article AS $A
  AND $A ~ "digital library"
  AND $A/#/p ~ "authorization & (access control) &
                security"
```

6.3 The INEX Result Evaluation

For each topic the results of all participants are collected into a result pool for this topic. Then the potentially relevant components from each pool are assessed by a human who assigns an *exhaustivity* value and a *specificity* value. Exhaustivity describes the extent to which the component discusses the topic of request, specificity describes the extent to which the component focusses on the topic of request. Each parameter can accept four values:

0 not exhaustive/specific
1 marginally exhaustive/specific
2 fairly exhaustive/specific
3 highly exhaustive/specific

To assess the quality of a set of search results a metric based on the traditional recall/precision metrics is applied. In order to apply this metric, the assessors' judgements have to be quantised onto a single relevance value. Two different quantisation functions have been used:

1. *Strict* quantisation is used to evaluate whether a given retrieval approach is capable of retrieving highly exhaustive and highly specific document components.

$$f_{strict}(ex, spec) = \begin{cases} 1 & ex{=}3,\ spec{=}3\ (short:\ 3/3) \\ 0 & otherwise \end{cases}$$

2. In order to credit document components according to their degree of relevance (generalised recall/precision), a *generalized* quantisation has been used.

$$f_{generalized}(ex, spec) = \begin{cases} 1 & 3/3 \\ 0.75 & 2/3, 3/2, 3/1 \\ 0.5 & 1/3, 2/2, 2/1 \\ 0.25 & 1/1, 1/2 \\ 0 & 0/0 \end{cases}$$

Given the type of quantisation described above, each document component in a result set is assigned a single relevance value using the human–based relevance assessment.

Now, the precision and recall for a submitted result can be calculated using strict quantisation or generalized quantisation.

6.4 The XXL Experiments

We submitted runs with and without enabling lookups in the ontology index. With the OI enabled, each keyword in the query is replaced by the disjunction of itself and all its related terms.

6.4.1 CO–Topics

For the first experiment we evaluate CO–topics with and without ontology support. This scenario is used to compare the precision and recall of the following two runs:

1. *CO:Init* ... for this run we do not use the ontology index for query evaluation.
2. *CO:Onto* ... for this run we use the ontology index for query expansion.

For example consider the CO–topic 98 with the keywords:

```
"Information Exchange" +"XML" "Information Integration"
```

The corresponding XXL query for the first run *CO:Init* without enabling lookups in the ontology index has the following where clause:

```
"information exchange" & "XML" & "information integration"
```

The corresponding XXL query for the second run *CO:Onto* using ontology–based query expansion has the following where clause:

```
("information exchange" | "data exchange" |
 "heterogeneous data") &
("XML" | "semistructured data") &
("information integration" | "information sharing")
```

For the first XXL query we obtain 7 results with an average precision of 0.0002 for the strict quantisation and with an average precision of 0.0043 for the generalized quantisation. For the second XXL query we obtain 28 results with an average precision of 0.0002 for the strict quantisation and with an average precision of 0.0065 for the generalized quantisation.

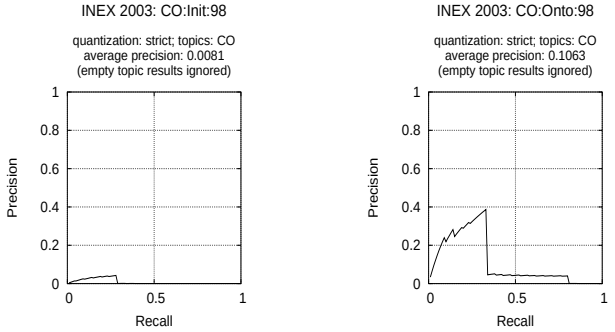
However, if we carefully look at the given topic, it turns out that a reformulation like the following could return better results. Thus, for the first XXL query we take:

```
("information exchange" | "information integration") &
"XML"
```

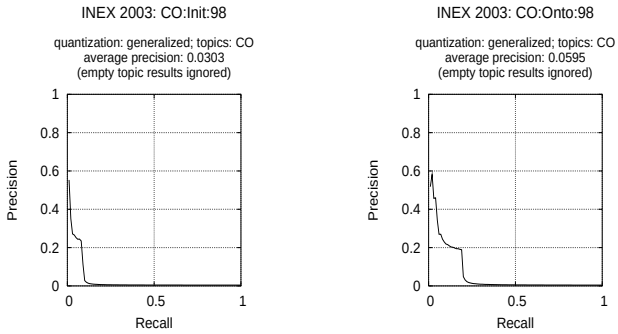
The second expanded XXL query has following structure:

```
(("information exchange" | "data exchange" |
  "heterogeneous data") |
 ("information integration" | "information sharing")) &
("XML" | "semistructured data") &
```

The following figure shows the average precision of the strict evaluation approach for the CO–topic 98 from the first run *CO:Init* (left) and from the second run *CO:Onto* (right).



The next diagrams show the average precision of the generalized evaluation approach for the CO–topic 98 from the first run *CO:Init* (left) and from the second run *CO:Onto* (right).



As the INEX runs had to use automatically generated queries such an optimization could not be applied. It turns out that this reformulation in fact yields even better results, even though the ontology–enabled results include some non–relevant results.

For the complete run of all 36 CO–topics submitted after official INEX deadline we obtain following results. The next two figures show the average precision with strict quantisation.

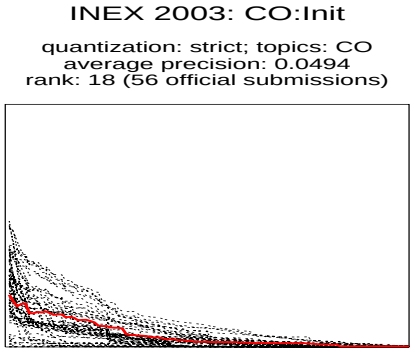


Figure 6: 36 CO–topics: XXL without OI (strict)

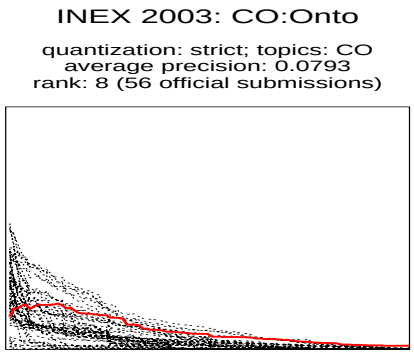


Figure 7: 36 CO–topics: XXL with OI (strict)

In the following two figures we see the average precision with generalized quantisation for the complete CO run.

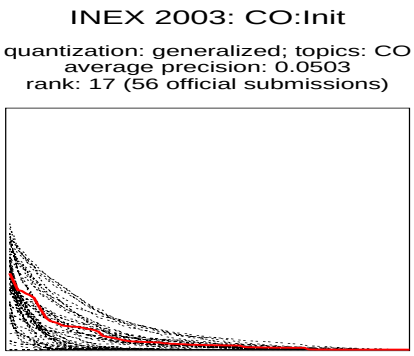


Figure 8: 36 CO–topics: XXL without OI (generalized)

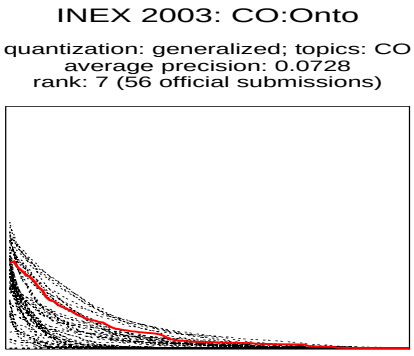


Figure 9: 36 CO–topics: XXL with OI (generalized)

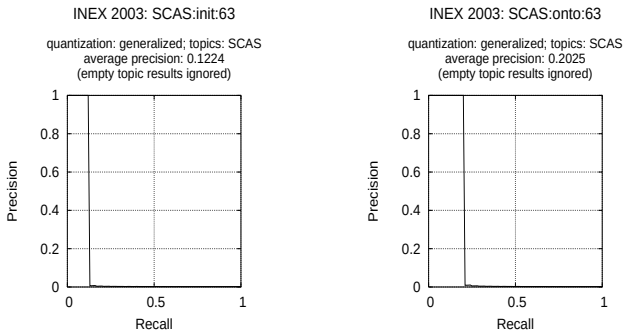
This experiment shows that ontology–based query expansion for keyword–based XML retrieval provides much better average precesion and better recall for each CO–topic.

6.4.2 CAS–Topics

For the second experiment we evaluate CAS–topics with and without ontology support. This scenario is used to compare the precision and recall of the following two runs:

1. *SCAS:Init* . . . for this run we evaluate the structural conditions exactly, but we do not use the ontology index for query evaluation.
2. *SCAS:Onto* . . . for this run we evaluate the structural conditions exactly and we use the ontology index for query expansion.

Because of some technical problems, we did not run all 30 CAS–topics. As an example for the ontology–based query evaluation on CAS–topics we present the generalized results for the CAS–topic 63. The strict evaluation of the first and the second run provides an average precision of 1.0.



This experiments shows that the XXL search engine is able to evaluate conditons on XML structure as well as conditions on XML contents. In addition, the ontology–based query expansion for the content condition provides much better average precision and better recall.

7. CONCLUSIONS

The results obtained for our XXL Search Engine in the INEX benchmark clearly indicate that exploiting semantic similarity generally increases the quality of search results. Given the regular structure of the INEX data, we could not make use of the features for structural similarity provided by XXL.

To further extend the result quality, we plan to add a relevance feedback step to incrementally increase the quality. Additionally, we will integrate information from other, existing ontologies into our ontology and extend the ontology to capture more kinds of relationships (e.g., instance-of relationships).

For future INEX benchmarks we would appreciate to have data that has a more heterogenous structure. The INEX data that is currently available is well suited for exact structural search with long paths, but not for search engines that exploit structural diversity.

8. REFERENCES

[1] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In Jensen et al. [13], pages 496–513.

[2] H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors. *Intelligent Search on XML Data*, volume 2818 of *Lecture Notes in Computer Science*. Springer, Sept. 2003.

[3] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. W3C recommendation, World Wide Web Consortium, 2002. `http://www.w3.org/TR/xquery`.

[4] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In D. Eppstein, editor, *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 937–946. ACM Press, 2002.

[5] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic search engine for XML. In *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*. Morgan Kaufmann, 2003.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[7] S. DeRose, E. Maler, and D. Orchard. XML linking language (XLink), version 1.0. W3C recommendation, 2001. `http://www.w3.org/TR/xlink/`.

[8] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. XML-QL. In *QL '98, The Query Languages Workshop, W3C Workshop*, Boston, Massachussets, USA, Dec. 1998.

[9] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[10] T. Grust. Accelerating XPath location steps. In M. J. Franklin, B. Moon, and A. Ailamaki, editors, *Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 2002, Madison, Wisconsin*, pages 109–120. ACM Press, New York, NY USA, 2002.

[11] T. Grust and M. van Keulen. Tree awareness for relational DBMS kernels: Staircase join. In Blanken et al. [2].

[12] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: ranked keyword search over XML documents. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 16–27. ACM Press, 2003.

[13] C. S. Jensen, K. G. Jeffery, J. Pokorný, S. Saltenis, E. Bertino, K. Böhm, and M. Jarke, editors. *Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, March 25-27, Proceedings*, volume 2287 of *Lecture Notes in Computer Science*. Springer, 2002.

[14] C. D. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[15] R. Schenkel, A. Theobald, and G. Weikum. Ontology-enabled XML search. In Blanken et al. [2].

[16] R. Schenkel, A. Theobald, and G. Weikum. HOPI: An efficient connection index for complex XML document collections. In *9th International Conference on Extending Database Technology (EDBT), Heraklion – Crete, Greece, March 14-18, 2004*, 2004.

[17] A. Theobald and G. Weikum. Adding Relevance to XML. In D. Suciu and G. Vossen, editors, *The World Wide Web and Databases: Third International Workshop WebDB 2000, Dallas, Texas, USA, May 18-19, 2000*, volume 1997 of *Lecture Notes in Computer Science*, pages 105–124, Berlin, Heidelberg, 2000. Springer.

[18] A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In Jensen et al. [13], pages 477–495.