INEX 2003 Workshop Proceedings

December 15-17, 2003
Schloss Dagstuhl
International Conference and Research
Center for Computer Science

http://inex.is.informatik.uni-duisburg.de:2003/

Preface

The aim of the workshop is to bring together researchers in the field of XML retrieval who participated in the Initiative for the Evaluation of XML retrieval (INEX) during 2003. The aim of the INEX initiative is to provide means, in the form of a large XML test collection and appropriate scoring methods, for the evaluation of XML retrieval systems. During the past year participating organisations contributed to the building of a large-scale XML test collection by creating topics, performing retrieval runs and providing relevance assessments along two relevance dimensions for XML components of varying granularity. The workshop concludes the results of this large-scale effort, summarises and addresses encountered issues and devises a workplan for the evaluation of XML retrieval systems.

The workshop is organised into presentation and workshop sessions. During the presentation sessions participants have the opportunity to present their approaches to XML indexing and retrieval. The workshops serve as discussion forums to review issues related to: the creation of the INEX topics; the definition of the two relevance dimensions; the use of the on-line assessment system; and the development of evaluation metrics. Finally, a track proposals workshop will identify new tasks for INEX'04.

Acknowledgement

We would like to thank the IEEE Computer Society for providing us the XML document collection. Special thanks go to Shlomo Geva for the set up of the WIKI server, Norbert Goevert for providing the evaluation metrics, Gabriella Kazai for helping with the various guideline documents, and Benjamin Piwowarski for providing the on-line assessment tool. Finally, we would like to thank the participating organisations and people for their contributions to the INEX test collection., as well as their contributions to evaluation methodologies for XML retrieval.

Norbert Fuhr, University of Duisburg-Essen
Mounia Lalmas, Queen Mary University of London
Saadia Malik, University of Duisburg-Essen

December 2003

Schloss Dagstuhl

Schloss Dagstuhl or Dagstuhl manor house was built in 1760 by the then reigning prince Count Anton von Öttingen-Soetern-Hohenbaldern. After the French Revolution and occupation by the French in 1794, Dagstuhl was temporarily in the possession of a Lorraine ironworks. In 1806 the manor house along with the accompanying lands was purchased by the French Baron Wilhelm de Lasalle von Louisenthal. In 1959 the House of Lasalle von Louisenthal died out, at which time the manor house was then taken over by an order of Franciscan nuns, who set up an old-age home there. In 1989 the Saarland government purchased the manor house for the purpose of setting up the International Conference and Research Center for Computer Science. The first seminar in Dagstuhl took place in August of 1990. Every year approximately 2,000 research scientists from all over the world attend the 30-35 Dagstuhl Seminars and an equal number of other events hosted at the center.

http://www.dagstuhl.de/

Table of Contents

Xpath Inverted File for Information Retrieval 1
Shlomo Geva, Murray Leo-Spork (*Queensland University of Technology*)

Distributed XML Information Retrieval 9
Wayne Kelly , Shlomo Geva , Tony Sahama, Wengkai Loke (*Queensland University of Technology*)

RMIT INEX experiments: XML Retrieval using Lucy/eXist 17
Jovan Pehcevski, James Thom (*RMIT University)*, Anne-Marie Vercoustre (*CSIRO-ICT Centre)*

An Evaluation of INEX 2003 Relevance Assessments 25
Kenji Hatano (*Nara Institute of Science and Technology*), Hiroko Kinutani (*Japan Science and Technology Agency*), Masahiro Watanabe (*The National Institute of Special Education*), Yasuhiro Mori, Masatoshi Yoshikawa (*Nogoya University),* Shunsuke Uemura (*Nara Institute of Science and Technology*)

XXL @ INEX 2003 33
Ralf Schenkel, Anja Theobald, Gerhard Weikum (*Max-Planck Institüt für Informatik*)

Applying the IRStream Retrieval Engine to INEX 2003 41
Andreas Henrich, Günter Robbert *(University of Bayreuth)* Volker Lüdecke *(University of Bamberg)*

HyREX at INEX 2003 49
Mohammad Abolhassani, Norbert Fuhr, Saadia Malik *(University of Duisburg-Essen)*

The SearX-Engine at INEX'03: XML enabled probabilistic retrieval 57
Holger Flörke (*doctronic GmbH)*

Retrieving the most relevant XML components 58
Yosi Mass, Matan Mandelbrod (*IBM Research Lab*)

Searching in an XML Corpus Using Content and Structure 65
Yiftah Ben-Aharon, Sara Cohen, Yael Grumbach, Yaron Kanza, Jonathan Mamou, Yehoshua Sagiv, Benjamin Sznajder, Efrat Twito (*The Hebrew University*)

The TIJAH XML-IR System at INEX 2003 (DRAFT) 72
J.A. List(*CWI*), V.Mihajlovic (*University of Twente*) ,A.P. De Vries(*CWI*), G. Ramírez (*CWI*), D.Hiemstra (*University of Twente*)

The University of Amsterdam at INEX 2003 80
Jaap Kamps, Maarten de Rijke, Börkur Sigurbjörnsson (*University of Amsterdam*)

Bayesian Networks and INEX'03 87
Benjamin Piwowarski, Huyen-Trang Vu, Patrick Gallinari *(LIP6)*

IRIT at INEX 2003 91
Karen Sauvagnat , Gilles Hubert, Mohand Boughanem, Josiane Mothe*(IRIT)*

Using Language Models for Flat Text Queries in XML Retrieval 97
Paul Ogilvie, Jamie Callan (*Carnegie Mellon University*)

Cheshire II at INEX'03: Component and Algorithm Fusion at XML Retrieval 104
Ray R. Larson (*University of California, Berkeley*)

Identifying and Ranking Relevant Document Elements 112
Andrew Trotman, Richard A. O'Keefe (*University of Otago*)

The Simplest Query Language That Could Possibly Work 117
Andrew Trotman, Richard A. O'Keefe (*University of Otago*)

Helsinki's EXTIRP @ INEX 125
Antoine Doucet, Lili Aunimo, Miro Lehtonen, Renaud Petit (*University of Helsinki*)

Using value-added document representations in INEX 132
Birger Larsen, Haakon Lund, Jacob K. Andresen and Peter Ingwersen (*Royal School of Library and Information Science*)

Adapting the Extended Vector Model for XML Retrieval 138
Carolyn J. Crouch, Sameer Apte (*University of Minnesota Duluth*), Harsh Bapat (*Persistent Systems Pvt. Ltd.*)

Cooperative XML (CoXML) Query Answering at INEX 03 142
Shaorong Liu, Wesley W. Chu (*UCLA Computer Science Department*)

Appendix

INEX'03 Guidelines for Topic Development 150

INEX'03 Retrieval Task and Run Submission Specification 158

INEX'03 Relevance Assessment Guide 162

Workshop Schedule 168

XPath Inverted File for Information Retrieval

Shlomo Geva
Centre for Information Technology Innovation
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434 Brisbane Q 4001 Australia
s.geva@qut.edu.au

Murray Leo-Spork
Centre for Information Technology Innovation
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434 Brisbane Q 4001 Australia
m.spork@qut.edu.au

ABSTRACT

In this paper we describe the implementation of a search engine for XML document collections. The system is keyword based and is built upon an XML inverted file system. We describe the approach that was adopted to meet the requirements of Strict Content and Structure queries (SCAS) and Vague Content and Structure queries (VCAS) in INEX 2003.

Keywords: Information Retrieval, Inverted File, XML, XPath, INEX, Assessment, Evaluation, Search Engine

1. Introduction

File Inversion is probably the most widely used technique in text retrieval systems. In an inverted file, for each term in the collection of documents, a list of occurrences is maintained. Information about each occurrence of a term includes the document-id and term position within the document. Maintaining a term position in the inverted lists allows for proximity searches, the identification of phrases, and other context-sensitive search operators. This simple structure, combined with basic operations such as set-union and set-intersect, support the implementation of rather powerful keyword based search engines.

XML documents contain rich information about document structure. The objective of the XML Information Retrieval System that we describe in this paper is to facilitate access to information that is based on both content and structural constraints. We extend the Inverted File scheme in a natural manner, to store XML context in the inverted lists.

2. XML File Inversion

In our scheme each term in an XML document is identified by 3 elements. File path, absolute XPath context, and erm position within the XPath context.

The file path identifies documents in the collection; for instance:

C:/INEX/ex/2001/x0321.xml

The absolute XPath expression identifies a leaf XML element within the document, relative to the file's root element:

/article[1]/bdy[1]/sec[5]/p[3]

Finally, term position identifies the ordinal position of the term within the XPath context.

One additional modification that we adopted allowed us to support queries on XML tag attributes. This is not a strictly content search feature, but rather structure oriented search feature. For instance, it allows us to query on the 2nd named author of an article by imposing the additional query constraint of looking for that qualification in the attribute element of the XML author element. The representation of attribute values is similar to normal text with a minor modification to the XPath context representation – the attribute name is appended to the absolute XPath expression. For instance:

article[1]/bdy[1]/sec[6]/p[6]/ref[1]/@rid[1]

Here the character '@' is used to flag the fact that "rid" is not an XML tag, but rather an attribute of the preceding tag <ref>.

An inverted list for a given term, omitting the File path and the Term position, may look something like this:

Context
XPath
article[1]/bdy[1]/sec[6]/p[6]/ref[1]
article[1]/bdy[1]/sec[6]/p[6]/ref[1]/@rid[1]
article[1]/bdy[1]/sec[6]/p[6]/ref[1]/@type[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[13]/pp[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[14]/pdt[1]/day[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[14]/pp[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[15]
article[1]/bm[1]/bib[1]/bibl[1]/bb[15]/@id[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[15]/ti[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[15]/obi[1]

In principle at least, a single table can hold the entire cross reference list (our inverted file). Suitable indexing of terms can support fast retrieval of term inverted lists. However, it is evident that there is extreme redundancy in the specification of partial absolute XPath expressions (substrings). There is also extreme redundancy in full absolute XPath expressions where multiple terms in the same document share the same leaf context (e.g. all terms in a paragraph). Furthermore, many XPath leaf contexts exist in almost every document (e.g. /article[1]/fm[1]/abs[1]).

We have chosen to work with certain imposed constraints. Specifically, we aimed at implementing the system on a PC and base it on the Microsoft Access database engine. This is a widely available off-the-shelf system and would allow the system to be used on virtually any PC running under any variant of the standard Microsoft Windows operating system. This choice implied a strict constraint on the size of the database – the total size of an Access database is limited to 2Gbyte. This constraint implied that a flat list structure was infeasible and we had to normalise the inverted list table to reduce redundancy.

3. Normalized Database Structure

The structure of the database used to store the inverted lists is depicted in Figure 1. It consists of 4 tables. The ***Terms*** table is the starting point of a query on a given term. Two columns in this table are indexed - The ***Term*** column and the ***Term_Stem*** column. The ***Term_Stem*** column holds the Porter stem of the original term. The ***List_Position*** is a foreign key from the ***Terms*** table into the ***List*** Table. It identifies the starting position in the inverted list for the corresponding term. The ***List_Length*** is the number of list entries corresponding to that term. The ***List*** table is (transparently) sorted by Term so that the inverted list for any given term is contiguous. As an aside, the maintenance of a sorted list in a dynamic database poses some problems, but these are not as serious as might seem at first, and although we have solved the problem it is outside the scope of this paper and is not discussed any further. A search proceeds as follows. Given a search term we obtain a starting position within the List table. We then retrieve the specified number of entries by reading sequentially.

The inverted list thus obtained is *Joined* (SQL) with the ***Document*** and ***Context*** tables to obtain the complete de-normalised inverted list for the term. The XPath context is then checked with a regular expression parser to ensure that it satisfies the topic's <Title> XPath constraints.

The retrieval by ***Term_Stem*** is similar. First we obtain the Porter stem of the search term.

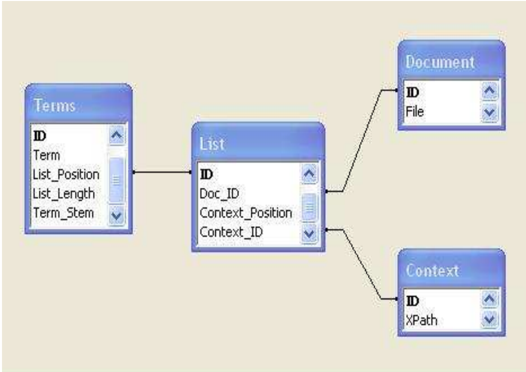


Figure 1: Database Schema for XML Inverted File.

Then we search the list by ***Term_Stem*** – usually getting duplicate matches. All the lists for the duplicate hits on the ***Terms*** table are then concatenated. Phrases and other proximity constraints can be easily evaluated by using the ***Context_Position*** of individual terms in the ***List*** table.

We have not compressed XPath expressions to minimise the extreme redundancy of XPath substrings in the ***Context*** table. With this normalization the database size was reduced to 1.6GByte and within the Microsoft Access limits.

5. The CASQuery Engine

Before discussing the implementation details of the CASQuery engine it is necessary to introduce some terminology. We then describe the implementation of the search engine.

5.1 Terminology

- **XPath Query**: An XPath Query is a query that meets the criteria of the INEX query specification. It can be considered a subset of the W3C's XPath language.
- **Step**: A Step is a component of an XPath query that specifies some Axis (child, descendant, descendant-or-self etc.) a NodeTest (e.g. a NameText that tests the name of an element) and optionally some Predicate
- **Path**: A Path is a sequential list of Steps
- **Predicate**: A predicate contains a filter that specifies some condition that a node must meet in-order to satisfy it. This filter may be an "about" function or an equality expression.
- **Context**: The context for an element is an absolute XPath expression denoted by a list of child steps with a numerical index e.g. "/article[1]/bdy[1]/sec[1]/p[4]"
- **ReturnElement**: A ReturnElement is an element (qualified by the document name and a context) that satisfies the full path expression of a query (or query fragment)

not including any path expression in a filter. The context of the ReturnElement is the one returned by the query engine to the user.

- **SupportElement**: A SupportElement is an element (qualified by the document name and a context) that satisfies the full path expression of a query (or query fragment) including any path expression in a filter. The context of the ReturnElement not returned to the user but can be used to "support" the validity of the ReturnElement (in other words: shows why the ReturnElement was in fact returned).

The search engine was designed to operate on the <Title> element of CAS topics. It operates in the same manner for both strict (SCAS) and vague (VCAS) interpretation of the queries. The only difference is in the definition of equivalence tags:

SCAS Equivalent tags:

- **Article,bdy**
- **p|p[1-3]|ip[1-5]|ilrj|item-none**
- **sec|ss[1-3]**
- **h|h[1-2]a?|h[3-4]**
- **l[1-9a-e]|dl|list|numeric-list|numeric-rbrace|bullet-list**

VCAS Equivalent tags:

- **Article,bdy,fm**
- **sec|ss[1-3]|p|p[1-3]|ip[1-5]|ilrj| item-none**
- **h|h[1-2]a?|h[3-4]**
- **yr|pdt**
- **snm|fnm|au**
- **bm|bibl|bib|bb**
- **l[1-9a-e]|dl|list|numeric-list|numeric-rbrace|bullet-list**

5.2 Parsing the Query

We used the Programmar[2] parser development toolkit to generate a parser for XPath[3] queries. Programmar accepts a Backus Naur Form (BNF) grammar as input and is able to generate a parser that can parse an instance of that query into a *parse tree*. The Programmar library then provides an API to access and walk the parse tree that it constructed.

We used the XPath BNF grammar as defined by the W3C as input to the Programmar IDE. Some small modification to the BNF syntax was made in order to make the task of walking the parse tree and gathering the required information simpler.

Our approach was to walk the parse tree and construct an abstract syntax tree, which represents that same query but at a higher level of abstraction than the parse tree generated by the Programmer toolkit. Representing the query at a higher level of abstraction meant that implementing the query engine that processes that query was made simpler.

5.3 The Abstract Syntax

The abstract syntax was contained within a separate module that is kept independent of the QueryEngine that processes it. Thus we allow for the possibility that the abstract syntax for XPath queries may be utilised in other applications. For example it would be possible to implement a more traditional XPath processor on top of this abstract syntax. Therefore there is a dependency from the Query Engine to the Abstract Syntax package but no reverse dependency.

The basic structure of an XPath query (in the abstract syntax) is that it consists of a Path that contains a list of Steps. This is consistent with the terminology used by the XPath standard. Steps must contain a node test – and may also contain zero to many filters (or predicates).

5.4 Evaluateable Fragments

Once the XPath parser has constructed the abstract syntax, the query engine performs one further transformation on the query before executing. The path, or list of steps, must be broken down into EvaluateablePathFragments. Each step in the query that contains an *EvaluatableExpression* will be treated as the last step in an EvaluateablePathFragment.

An *EvaluatableExpression* is a step filter that can be evaluated by the QueryEngine.

In our implementation we are using an index of inverted lists that map a *term* to a list of *contexts* (full absolute XPath path plus document name). Therefore, for a filter to be *evaluateable* it must filter based on some term that can be looked up in the index. For example the filter:

/article/bdy[count()=1]

would not be *evaluateable* in our system as no terms is given in the filter. However the filter:

/article//yr[. = "1999"]

is evaluateable as the term "1999" will be in the index.

As an example, the query:

//article[//yr='1999']//sec[about(./,'DEHOMAG')]

would be broken down into two fragments:

1. **//article[//yr = "1999")]**
2. **//article//sec[about(//p, 'DEHOMAG')]**

Notice that the second fragment contains the full path including the "article" step.

Next each EvaluatablePathFragment is evaluated – the eval() method will return a set of nodes whose *contexts* match the full path for that fragment. For example fragment 2 above may return a node with the context:

/article[1]/bdy[1]/sec[2]

5.5 Merging Fragments

After each fragment is evaluated independently, we will have a list of node sets (one for each fragment) that must be merged. For example when merging the two sets from the above fragments, we will wish to include only those elements returned from the first fragment if they also have a descendent node contained in the set returned from the second fragment. In fact, what we need to return are elements with a context that matches the full path of the last fragment (in the case above they must have a context that matches //article//sec – the last named element in the context must be "sec"). What is meant by "including elements from the first fragment" is that the *SupportElements* for those *ReturnElements* in the first set will be added to a descendant *ReturnElement* (if it exists) in the 2nd set.

For example: let us say that the first set contains a ReturnElement with the context "/article[1]" and that ReturnElement has an attached SupportElement of "/article[1]/fm[1]/yr[1]" (for the purposes of this example assume that all contexts are in the same document. Then let us say that the second set contains a ReturnElement of "/article[1]/bdy[1]/sec[2]". This element is supported by /article[1]/bdy[1]/sec[2]/p[3]. In this case the ReturnElement in the 2nd is a descendant of the ReturnElement in the 1st set – so we can merge the supports from the 1st ReturnElement into the supports of the 2nd and we will end up with a ReturnElement ("/article[1]/bdy[1]/sec[2]") that has 2 supports ("/article[1]/fm[1]/yr[1]" and "/article[1]/bdy[1]/sec[2]/p[3]").

When merging sets we must determine whether to do a strict merge or a union merge. For example if we need to merge the 2 fragments above, fragment 1 is "strict" – all elements that we merge from fragment 2 must also have an ancestor "article" element that contains a "yr" element for "1999".

The last fragment will always require a strict merge. This is because of the requirement stated above, that all elements returned by the query must have a context that satisfies the full path of the query.

However, a Union merge can be appropriate when we are merging two fragments where neither are the last fragment in the query, and both are non-strict (for example both only contain "about()" filters. In this case all ReturnElements will be retained, whether an element returned from the second fragment is a descendant of some element from the first fragment or not.

5.6 Support Elements

Support elements are elements that were found to contain at least one instance of a term that was specified in the filter. The element that contains this term must satisfy the full path for that filter including the context path.

In our example above the first filter (first fragment) looks for occurrences of the term "1999" in elements whose context matches the path "//article//yr". If we find that the term "1999" occurs in an element with the context "/article[1]/bdy[2]/sec[1]/p[1]" this is not a valid support for this filter. However, if we find a single occurrence of "1999" in the context "/article[1]/fm[1]/yr[1]" this would be a valid support.

Once we have removed all supports that do not represent valid supports (according to the filter), we then can create the return elements for this filter. In this case the return path is "//article" so the return element would have the context "/article[1]" with an attached support element with the context "/article[1]/fm[1]/yr[1]" and having one "hit" for the term "1999". It is possible that a return element contains more than one support element. For example, if within the same document we find another element with the context "/article[1]/fm[1]/yr[2]" that contains 2 hits on the term "1999" we would add another support element to the return element and record 2 hits on it. (This example is spurious as in the case of an equality constraint you actually only want to find one hit on the term. However it would make sense in the context of the "about()" filter).

5.7 Ranking

The approach we adopted in ranking was a multi-stage sorting process.

- First sort by f*ilter satisfaction*.
- For ReturnElements that satisfy the same number of filters - sort by number of distinct terms and phrases that were hit.
- For ReturnElements with the same number of filters satisfied and the same number of distinct terms - calculate a score based on total number of terms hit adjusted by a factor that penalises terms that are very common in the document collection.

5.7.1 Filter Satisfaction

A ReturnElement is considered to have satisfied a filter where it is a valid ReturnElement for that filter, and it has a least one SupportElement that has recorded a hit for at least one term in the filter. A valid ReturnElement is one whose context matches the path expression of the filter.

In its simplest form, the filter satisfaction algorithm will rank higher a ReturnElement that has satisfied a greater number of filters. There are a number of refinements to this rule:

- Where two filters appear as *Predicates* to different *Steps* in the query expression (e.g. //article[//yr = "1999"]

//sec[about(./, 'DEHOMAG')]), each one of these filters that is satisfied will count towards the overall *filter satisfaction count.*

- Where two filters appear in the same Predicate and they are and-ed together (e.g. //article//sec[[//yr = “1999”] **AND** about(./, 'DEHOMAG')]), each one of these filters that is satisfied will count towards the overall *filter satisfaction count.*
- Where two filters appear in the same Predicate and they are or-ed together (e.g. //article//sec[[//yr = “1999”] **OR** about(./, 'DEHOMAG')]), if both filters are satisfied only one will be counted towards the overall *filter satisfaction count.*
- If any unwanted terms are hit in a SupportElement for the ReturnElement, then the *filter satisfaction count* will be reduced by a count of 2.

5.7.2 Distinct terms and phrases

This algorithm is a second stage sort after the *filter satisfaction* sort. Where two ReturnElements have the same *filter satisfaction count*, the distinct terms algorithm is applied to determine their relative rank. Here we rank ReturnElements based on the number of distinct terms and phrases that they satisfy.

If a SupportElement has recorded hits for a particular term, its containing ReturnElement will have it’s *distinct terms and phrases count* incremented by one. Take for example the query:

//article[about(.//st,'+comparison') and

about(.//bib,'"machine learning"')]

Let us take the case where we have two ReturnElements that satisfy both filters. The first ReturnElement has supports that hit the terms “comparison” and “machine”. The second ReturnElement has supports that hit the terms “comparison”, “machine” and “learning”. In this case the second ReturnElement will be ranked higher. Note that it does not matter how many times each term is hit – it only matters if a term was hit at least once, or not at all.

The *distinct terms and phrases count* secondly takes into account the number of phrases that a ReturnElement has supports for. For example, take the query and the second ReturnElement we discussed above. If this ReturnElement also contained a support for the phrase “machine learning” - that is to say a context was found where the words “machine” and “learning” appear directly adjacent to each other – the *distinct terms and phrases count algorithm* will increment the count by one.

5.7.3 Scorer penalizes frequent terms

The final stage algorithm of the 3 stage sort is only invoked where two ReturnElements have the same *filter satisfaction count* and *distinct terms and phrases count*. This algorithm calculates a score based on the total number of instances that terms were hit by SupportElements. The total number of hits for a term is normalized based on heuristic that takes into account how frequently that term occurs in the entire documents collection. This normalization factor is calculated as follows:

- *Hits*: Total number of instances that this term appears in the ReturnElements supports.
- *TermFrequency*: Count of number of times this term appears in total document collection
- *TermFrequencyConstant*: A constant (determined using heuristics)
- *Score*: The ranking score for this ReturnElement
- *Terms*: The set of terms the score is based on
- i: Denotes the term
- *ScarcityMultiplier* =

1 + (*TermFrequency* / *TermFrequencyConstant*)

- *Score* = $\sum_{i \in Terms} (hits_i * (1 / ScarcityMultiplier_i))$

5.8 Discussion on Ranking

Our overall ranking strategy was based on a series of heuristics.

5.8.1 Filter Satisfaction

It is clear that our strategy places a high degree of importance to whether a particular collection of query terms are aggregated into one filter or if they are put in separate filters. For example, let us take the following two queries:

//article[about(.,'clustering distributed') and about(., 'java')]

//article[about(.,'clustering distributed java')]

Whilst these filters may appear logically equivalent, our filter satisfaction algorithm will mean that lists returned from each query formulation will vary significantly in how they are sorted. With the first query, the term “java” is raised to the same level of importance as that of both the other terms (“clustering” and “distributed”). By contrast, with the second query, a result that hits “clustering” and “distributed” (but not “java”) will rank equal to a result that hits “distributed” and “java” (but not “clustering”). However, if the first query formulation is used the second result would be ranked higher as it satisfies two filters whereas the first result only satisfies one.

We believe this ranking strategy works well due to the psychology involved in creating these two filters. It can be inferred that when a query writer aggregates terms into one filter he/she considers all terms so aggregated of equal importance. In contrast, where a query writer puts terms in separate filters they are indicating that whilst each filter should be treated of equal importance, terms contained in separate filters are not necessarily of equal importance.

The second thing worth discussing about the filter satisfaction algorithm is the way it treats or-ed filters versus the treatment for and-ed filters. Let us take another two filters by way of example:

//article[about(.,'clustering) and about(.,'distributed')]//sec[(about('java')]
//article[about(.,'clustering) or about(.,'distributed')]//sec[(about('java')]

Further, let us assume we have *returnElement1* that hits the terms "clustering" and "distributed" and *returnElement2* that hits the terms "clustering" and "java".

In this case query 1 will rank *returnElement1*and *returnElement2* equal (both with a *filter satisfaction count* of 2). However query 2 will treat these quite differently. The *returnElement2* will still have a *filter satisfaction count* of 2 but the *returnElement1* will have a *filter satisfaction count* of only one.

Again we believe this makes intuitive sense. The second query construction implies that the user wants one of "clustering" or "distributed" to be hit – they don't care which – and if they are both hit then this is not as important as if "java" is also hit. It is interesting to note that the following query would be equivalent to the second query:

//article[about(.,'clustering distributed')]//sec[(about('java')]

One final thing to note about this algorithm is how it treats unwanted terms (i.e. terms preceded by a minus sign). The algorithm is very harsh in how it treats the occurrence of such terms (by deducting 2 from the overall *filter satisfaction count*). However, We have found this works well in practice as the specification of such unwanted terms by the query writer appears to indicate a very strong aversion to that term.

5.8.2 Distinct Terms and Phrases

The *distinct terms and phrases* algorithm is important in two respects:

- It places a greater importance on the number of distinct terms hit, than on the total number of instances that a term or terms are hit. (This can also be said about the *filter satisfaction* algorithm).
- Phrases are given prominence by the fact that they in effect count as an additional distinct term.

Let us consider the consequences of first point above. Take as an example the filter "//article[about(.,'clustering distributed java')]". Let us say that a ReturnElement records hits on the term "clustering" and "distributed"; both terms with 100 instances of this term occurring in the return's supports – a total of 200 recorded hits. Then let us take another ReturnElement that records just the one instance of a hit on each of "clustering", "distributed" and "java". It may surprise that this second ReturnElement will be ranked higher when it only recorded 3 separate hits versus the 200 of the first ReturnElement.

However, we believe this strategy has worked quite well in reality. What we have found that this in effect gives a greater prominence to those terms that do not occur frequently – that is it weights infrequent terms more heavily than frequent terms. This makes intuitive sense as an infrequent term that appears in a query is more likely to aid the precision of the recall than frequent terms. The more frequent a term is in the overall document collection the less value it has to determining the requirements of the user.

As regards the 2nd point above about giving phrases prominence, this should be self explanatory. Phrases occur much less frequently than individual terms, so it makes sense to treat them with a level of importance equivalent to the individual terms.

5.8.3 Scorer penalizes frequent terms

Finally we discuss the algorithm that is invoked where the above two algorithms still cannot separate two equally ranked ReturnElements. It is only in this final stage algorithm that we take into account how "strong" the support is for a ReturnElement – that is how many instances of hits on terms have been recorded in a ReturnElement's SupportElements.

As per our discussion for the *distinct terms and phrases* algorithm, here we also wish to penalize infrequent terms. The algorithm we developed to do this was refined by running a series of experiments and running our own assessment on the results to see if the modified algorithm improved the results. The *TermFrequencyConstant* gives us the ability to adjust the *normalization factor* for penalizing frequent terms.

5.9 Exceptions

Some INEX topics included conditions that could not be easily evaluated in the absence of external knowledge. For instance, a conditions such as **about[.//yr,2000].** Such a condition can be easily evaluated if a user, or an external schema can be consulted, in which the meaning of “about” in relation to <yr> can be determined. Furthermore, in practical terms, the implementation must take account of the type of the element (e.g, is it numeric or alphanumeric?).

The treatment of equality functions involving years (i.e. a "yr" tag) is straightforward: a string comparison is made between the value in the tag and the constant. However, the treatment of inequality functions (i.e. those involving inequality operators "<", ">", "<=", ">=") is more complex. The greater than operator is undecideable as the upper range of year values to search the index for is unbounded. The less than operator may be decideable if we take the year 1 as the lower bound - but in this case the practical consequence of having to search the index for upwards of 1,990 terms is that we need to define a more reasonable lower bound. As such, we allowed for the lower and upper bound of year terms to be configured via our configuration file. This results in a managable range of year terms for which we have to search the index for any reasonable year based inequality predicate. The “about” was also defined in a configuration file (3 years either side of specified “about” year).

6. Experimental Results

The system was only designed for Content and Structure queries (CAS). Only the <Title> element was used in topic evaluation. The system was not designed to take advantage of information contained in the <Description> and <Keywords> elements of a Topic.

6.1 Strict Content and Structure (SCAS)

The best results were obtained with the SCAS query and strict quantization metric. The average precision was 0.26 (the submission was ranked **3rd**.) With the Generalized quantization metric the system was ranked 8th. These results are somewhat surprising given that we only used the <Title> element of a topic. One would have expected the use of additional keywords from the <Description> and <Keywords> elements to assist retrieval and ranking.

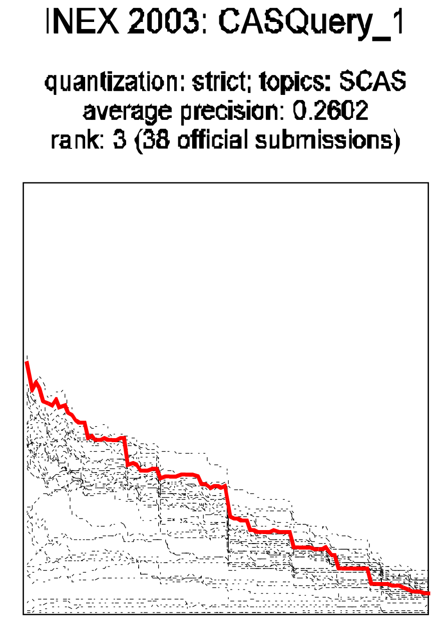


Figure 2: Retrieval results for Strict Content and Structure (SCAS) topics, quantization Strict

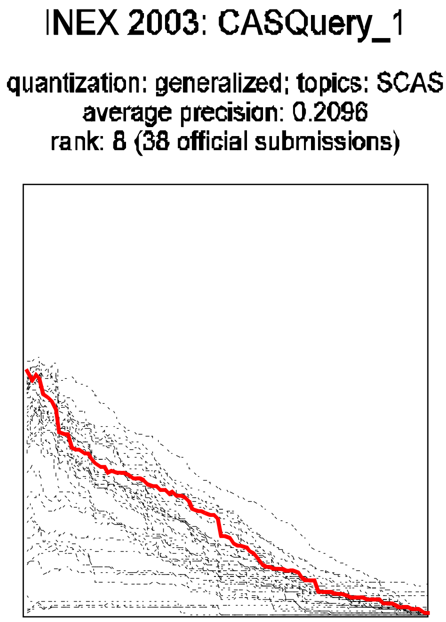


Figure 3: Retrieval results for Strict Content and Structure (SCAS) topics, quantization Generalized.

6.2 Vague Content and Structure (VCAS)

Results are not available at the time of writing this pre-workshop paper.

7. Discussion

There is no question that the formulation of the <Title> element of an XML topic at INEX 2003 is not end user oriented. However, it does allow for exact specification of structure and content constraints. We were able to implement a search engine that evaluates CAS <title> expressions with good accuracy and reasonable response time. Furthermore, we were able to construct the search engine on top of a generic XML inverted file system. This allows the application of the system to XML collections without explicit reference to the underlying XML Schema (or DTD). It seems however that in the definition of INEX CAS Topics the authors did not always specify the intent of the topic (as evident in the topic's narrative) in an accurate manner. This ultimately must have lead to low precision (across all submissions from all participants).

We were not able to solve the problem in a completely generic fashion because some topics' structural constraints could not be easily interpreted in a generic manner (e.g treatment of about conditions over <year>). This problem can be overcome to some extent with the use of an XML Schema in future evaluations at INEX.

REFERENCES

[1]Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), DELOS Network of Excellence on Digital Libraries, ERCIM-03-W03

[2]Programmar™ Developers Toolkit, NorKen Technologies, Inc., http://www.programmar.com/

[3]"XML Path Language (XPath) Version 1.0" http://www.w3.org/TR/xpath

Distributed XML Information Retrieval

Wayne Kelly
Centre for Information Technology Innovation
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434 Brisbane Q 4001 Australia
w.kelly@qut.edu.au

Shlomo Geva
Centre for Information Technology Innovation
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434 Brisbane Q 4001 Australia
s.geva@qut.edu.au

Tony Sahama
Centre for Information Technology Innovation
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434 Brisbane Q 4001 Australia
t.sahama@qut.edu.au

Wengkai Loke
Centre for Information Technology Innovation
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434 Brisbane Q 4001 Australia
w.loke@qut.edu.au

ABSTRACT

In this paper we describe the implementation of a distributed search engine for XML document collections. The system is based on a generic P2P collaborative computing framework. A central server coordinates query and search results distribution. The server holds no documents nor does it hold any indexes. The document collection is distributed amongst multiple PC based workstations, where it is also indexed and searched. The system is scalable to databases several orders of magnitude larger than the INEX collection, by using a system of standard networked PCs.

Keywords: P2P, INEX, XML, Distributed Database, Information, Retrieval, Inverted File, XPath, Assessment, Evaluation, Search Engine

1. Introduction

Web search engines such as Google are enormously valuable in allowing ordinary users to access information on a vast array of topics. The enormity of the information being searched and the massive number of clients wishing to make use of such search facilities means, however, that the search mechanisms are inherently constrained. The data being searched needs to be a priori indexed. Searching is limited to finding documents that contain at least one occurrence of a word from of a list of words somewhere within its body. The exact relationship of these words to one another cannot be specified. These limitations mean that it is often difficult to specify exactly what you want, consequently clients are overwhelmed by an avalanche of query results – if users don't find what they are looking for in the first couple of pages of results they are likely to give up.

XML documents contain rich structural information that can be used by information retrieval system to locate documents, and part thereof, with much greater precision than text retrieval systems can. However, systems capable of searching XML collections by content are typically resource hungry and are unlikely to be supported extensively on central public servers for some time to come, if at all.

Peer to Peer (P2P) file sharing systems such as KaZaA, Gnutella and Napster enable documents to be searched and accessed directly from end user's PCs, i.e., without needing to *publish* them on a web server, but again the indexing for retrieval is a priori. This is fine if you are searching based on well defined metadata keys such as song title or performer, but not if you are trying to search based on the content of the data.

The greatest degree of search specificity is achieved if the search engine can potentially access the content of the entire collection for each and every query. Obviously this is infeasible for huge document collections such as the entire WWW. If, however, we limit ourselves to smaller collections such as documents archived by a "community" of individuals that are collaborating on some project or share some common interest, then such a precise Information Retrieval paradigm is feasible and highly desirable.

The P2P framework that we propose is based on *search agents* that visit the workstations of participating individuals to perform custom searches. Individuals wishing to perform a search can choose from a library of "standard" search agents, or they can implement their own agent that implements an arbitrarily sophisticated search algorithm. The agents execute on the individual workstations within our P2P host environment that "sand-boxes" them, preventing them from doing "harm" to the workstations and allowing the workstation owners to control exactly which "resources" can be accessed. Resources potentially accessed include files, directories and databases. The key advantages of our system compared to web search engines such as Google are:

- Arbitrarily sophisticated algorithms can be used to perform highly selective searches, since the query is known before the actual document collection is scanned.
- The documents don't have to be explicitly published to a central server – they are accessed in place. This saves time and effort and means that working documents can be made immediately available from the time they are created, and work can continue on those documents locally while still being externally accessible.
- Volunteers have the option to only partially publish documents. This means they allow a client's search agent to examine their documents, but they limit the response that such search agents can return to the client. The response could be as limited as saying "Yes - I have a document that matches your query". In most cases, the agent will return some form of URL which uniquely identifies the matching document, but our framework doesn't in itself provide a mechanism for the client to retrieve that document from the volunteer. The exact mechanism by which such documents are retrieved is beyond the scope of this paper, but it could for example be a manual process, whereby the owner of the volunteer workstation will access each such client request based on the identity of the client and the document being retrieved. This might happen, for example, in a medical setting with doctors requesting patient records from other doctors, or in a law enforcement setting with police agencies requesting criminal histories from other jurisdictions.

The remainder of this paper is organized as follows. In section 2 we describe the system underlying the distributed search engine. In section 3 we describe the XML search engine that is distributed and executed by search agents on the distributed database. In section 4 we discuss the results of testing the systems against the INEX collection. In section 5 we discuss and summarize the lessons learnt from the INEX exercise.

2. System Architecture

Our system is termed P2P in that the actual searching is performed on peer nodes. The internal architecture of our system is, however, client/server based - for a number of reasons. The underlying architecture of our system is illustrated in Figure 1. The client PCs that make up the "leaves" of system belong to the individuals in the community and can play two distinct roles; they can be a *searcher* or they can *volunteer* to be searched. A *searcher* is a PC that submits queries to the system. The *volunteers* are the PCs on which the documents reside and on which the queries are processed. Individual PCs can play either or both of these roles at various points in time. PCs volunteer themselves to be searched typically only when they are otherwise idle. This is a form of cycle stealing, as the execution of the search agents may consume considerable CPU time and memory bandwidth of the machine while it is running.

The clients of the system - the searchers and the volunteers come and go over time; the *search server* is the only part of the system that remains constant. It acts as a central point of contact for searchers wishing to submit queries and for volunteers willing to be searched. It also acts as a repository for queries waiting to be processed and query results waiting to be retrieved. At the point in time when a searcher submits a query, there may be some volunteers "currently connected" to the server that would be willing to process that query immediately. In such a case some results may be able to be returned to the searcher almost immediately (allowing of course, for the time to perform the search on the volunteer machines - which can be arbitrarily long depending on the complexity of the search algorithm and the size of the document collection being searched on each PC).

Often, however, the relatively small set of set of volunteers that are currently connected, will either produce no results for the query, or at least will produce no results that are satisfactory to the searcher (note that this is made more probable by the high degree of query specificity that is possible with an agent based search framework). In such a case, we assume the searcher will often be willing to wait (minutes, hours, days or perhaps even weeks) for other volunteers to connect to the system and hopefully contribute interesting new results. This is the key difference between our distributed search engine and traditional cycle stealing systems. In a traditional cycle stealing system, all volunteers are considered equal – once a computational task has been carried out by one machine there is no point if having any other volunteer machine repeat that

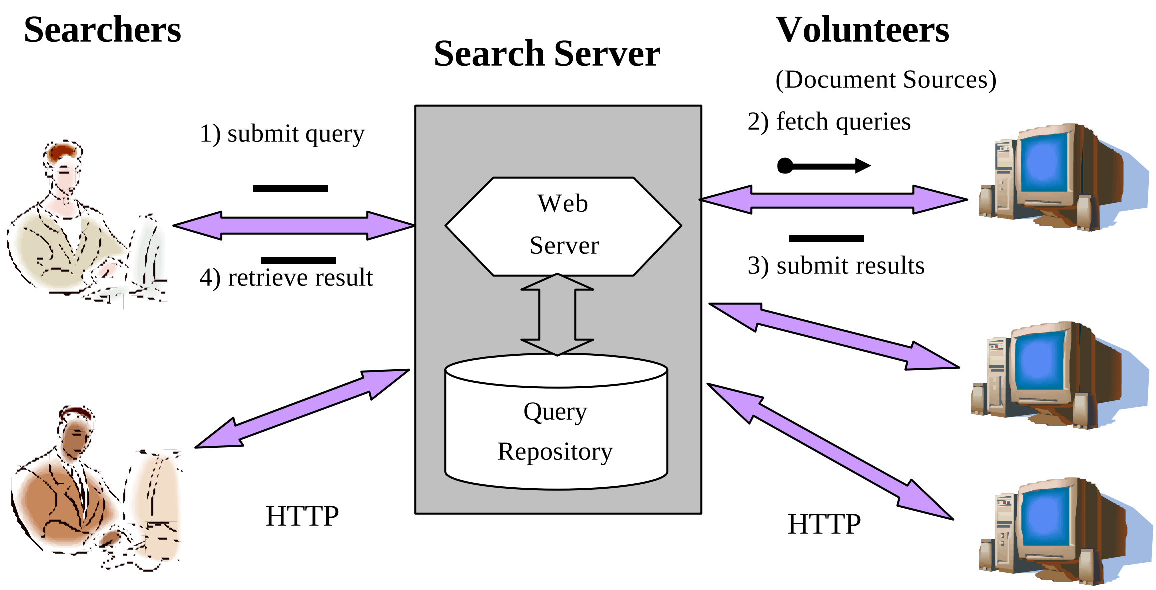


Figure 1: System Architecture

same computation. In our distributed search system, however, each PC is assumed to archive a

different set of documents – so even if a query has been processed on one volunteer, it still makes sense to keep that query around for other volunteers to process when they connect later.

Having a query and results *repository* allows the submission of queries and results to be separated in time from the fetching and processing of those queries and results. Having a central server means that once a client has submitted a query, it can disconnect from the system, and only reconnect much latter when it expects to find a significant collection of results. More importantly, the wide spread use of corporate firewalls will often mean that PC's performing searchers cannot directly communicate with many potential volunteers and vice versa. Having a central server that is able to receive HTTP requests from anywhere on the Internet has the effect of providing a gateway for searchers and volunteers to work together who would otherwise be unable to communicate. Note, installing a web server on all searcher and volunteer PC's would *not* achieve the same effect – a HTTP request message generally can not be sent to a machine behind a firewall, even if that machine hosts a web server.

The search server exposes interfaces to searchers and volunteers as SOAP web services transported using HTTP. Searchers can submit queries and fetch results and volunteers can fetch queries and submit results. All communication is initiated by either the searchers or the volunteers, and connections are not left open; i.e, the server can't push either queries or results to the searchers or the volunteers - they must request them. From the volunteer's perspective, the server is stateless. The server maintains *neither* a list of currently "connected" volunteers, nor a list of all potential volunteers. Anyone can volunteer at any time (subject to any authentication that the server may implement to ensure that the volunteer is a member of "the community"). When a volunteer connects to the server (after having been "disconnected" for a period of time) it receives a list of all queries that have been submitted to the server since that volunteer last connected. Each volunteer is responsible for keeping a "time-stamp" (in reality a sequence number allocated by the server) that represents the point in time at which that volunteer last requested queries from the server. In this way, the server is spared from maintaining information specific to each volunteer yet is able to respond to requests from individual volunteers in a personalized manner.

The time period that a query remains on the server is determined by a number of factors. Firstly, the searcher can specify a "time-to-live" when they submit the query. This may be overridden by the server which may dictate a system wide maximum "time-to-live" for all queries. Individual volunteers may also implement their own policies, such as refusing to process queries that are older than a certain date. Finally, the searcher can manually retract a query from the server as soon as they have received satisfactory result(s) to their query or if they realize that the query was incorrect or too inexact.

3. The XML Search Engine

The search engine is based on an XML inverted file system, and a heuristic approach to retrieval and ranking. These are discussed in the following sections.

3.1. The XML Inverted File

In our scheme each term in an XML document is identified by 3 elements. File path, absolute XPath context, and term position within the Xpath context.

The file path identifies documents in the collection ; for instance :

C :/INEX/ex/2001/x0321.xml

The absolute Xpath expression identifies a leaf XML element within the document, relative to the file's root element:

/article[1]/bdy[1]/sec[5]/p[3]

Finally, term position identifies the ordinal position of the term within the Xpath context.

One additional modification that we adopted allowed us to support queries on XML tag attributes. This is not a strictly content search feature, but rather structure oriented search feature. For instance, it allows us to query on the 2nd named author of an article by imposing the additional query constraint of looking for that qualification in the attribute element of the XML author element. The representation of attribute values is similar to normal text with a minor modification to the Xpath context representation – the attribute name is appended to the absolute Xpath expression. For instance:

article[1]/bdy[1]/sec[6]/p[6]/ref[1]/@rid[1]

Here the character '@' is used to flag the fact that "rid" is not an XML tag, but rather an attribute of the preceding tag <ref>. An inverted list for a given term, omitting the File path and the Term position, may look something like this:

Context
Xpath
article[1]/bdy[1]/sec[6]/p[6]/ref[1]
article[1]/bdy[1]/sec[6]/p[6]/ref[1]/@rid[1]
article[1]/bdy[1]/sec[6]/p[6]/ref[1]/@type[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[13]/pp[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[14]/pdt[1]/day[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[14]/pp[1]
article[1]/bm[1]/bib[1]/bibl[1]/bb[15]
article[1]/bm[1]/bib[1]/bibl[1]/bb[15]/@id[1]

In principle at least, a single table can hold the entire cross reference list (our inverted file). Suitable indexing of terms can support fast retrieval of term inverted lists. However, it is evident that there is extreme redundancy in the specification of partial absolute Xpath expressions (substrings). There is also extreme redundancy in full absolute Xpath expressions where multiple terms in the same document share the same leaf context (e.g. all terms in a paragraph). Furthermore, many Xpath leaf contexts exist in almost every document (e.g. /article[1]/fm[1]/abs[1]).

We have chosen to work with certain imposed constraints. Specifically, we aimed at implementing the system on a PC and base it on the Microsoft Access database engine. This is a widely available off-the-shelf system and would allow the system to be used on virtually any PC running under any variant of the standard Microsoft Windows operating system. This choice implied a strict constraint on the size of the database – the total size of an Access database is limited to 2Gbyte. This constraint implied that a flat list structure was infeasible and we had to normalise the inverted list table to reduce redundancy.

3.2 Normalized Database Structure

The structure of the database used to store the inverted lists is depicted in Figure 1. It consists of 4 tables. The ***Terms*** table is the starting point of a query on a given term. Two columns in this table are indexed - The ***Term*** column and the ***Term_Stem*** column. The ***Term_Stem*** column holds the Porter stem of the original term. The ***List_Position*** is a foreign key from the ***Terms*** table into the ***List*** Table. It identifies the starting position in the inverted list for the corresponding term. The ***List_Length*** is the number of list entries corresponding to that term. The ***List*** table is (transparently) sorted by Term so that the inverted list for any given term is contiguous. As an aside, the maintenance of a sorted list in a dynamic database poses some problems, but these are not as serious as might seem at first, and although we have solved the problem it is outside the scope of this paper and is not discussed any further. A search proceeds as follows. Given a search term we obtain a starting position within the List table. We then retrieve the specified number of entries by reading sequentially.

The inverted list thus obtained is *Joined* (SQL) with the ***Document*** and ***Context*** tables to obtain the complete de-normalised inverted list for the term. The XPath context is then checked with a regular expression parser to ensure that it satisfies the topic's <Title> XPath constraints.

The retrieval by ***Term_Stem*** is similar. First we obtain the Porter stem of the search term.

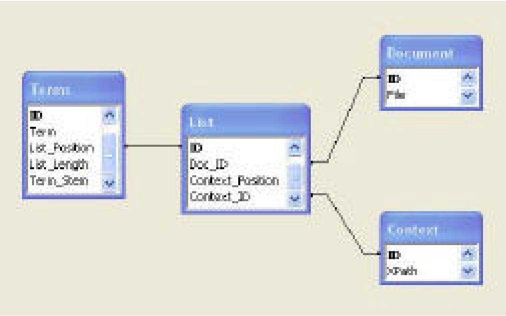


Figure 2: Database Schema for the XPath based Inverted XML File.

Then we search the list by ***Term_Stem*** – usually getting duplicate matches. All the lists for the duplicate hits on the ***Terms*** table are then concatenated. Phrases and other proximity constraints can be easily evaluated by using the ***Context_Position*** of individual terms in the ***List*** table.

With this normalization the database size was reduced to 1.6GByte and within the Microsoft Access limits. This is of course a trade of in performance since costly join operations may be necessary for the more frequent terms.

3.3 Searching the Database

The database structure enables the identification of inverted lists corresponding to individual terms. Each term that appears in a filter of an INEX <Title> element has an associated Xpath context. Terms that appear in a <keywords> element of a topic have the default context of /article. With simple SQL statements it is easy enough to retrieve inverted lists for terms that satisfy a filter.

3.3.1 SCAS topics

Our search strategy for SCAS topics consists of several steps, as follows.

We start by fragmenting the INEX <Title> element into several sub-queries, each corresponding to a filter on the path. So, for instance:

<title>//article[about(.//st,'+comparison')/ bm[about(.//bib,'machine learning')]</title>

is transformed to a set of 2 individual queries:

S|//article|//article//st|+comparison
R|//article/bm|//article//bm//bib|machine learning

This formulation identifies two sub-queries, each with 4 parts delimited by a '|'. The **S** denotes a support element and the **R** denotes a Returned Element. The support element has the Xpath signature **/article**. The return element has the XPath signature **/aticle/bm**. The support element filter looks for elements with the Xpath signature **//article//st,** containing the term "**comparison**". The returned element filter looks for elements with the Xpath signature **//article/bm//bib,** containing the phrase "**machine learning**".

Strict compliance to the XPath signature of the various elements is enforced. However, this is moderated by the use of equivalent tags.

SCAS Equivalent tags:

- **Article,bdy**
- **p|p[1-3]|ip[1-5]|ilrj|item-none**
- **sec|ss[1-3]**
- **h|h[1-2]a?|h[3-4]**
- **l[1-9a-e]|dl|list|numeric-list|numeric-rbrace|bullet-list**

Each of the elements is scored in the following way – we count the number of times that each term in the filter is found in the element. If more than one term is found then the term counts are multiplied together. This has the desired heuristic that elements containing many search terms are scored higher than elements having fewer search terms.

The score of a returned element is the sum of the scores of all its support elements. So in the example above, the score of a **//article/bm** element is the sum of all the corresponding **//article//st** elements (within the same <article>) and all **//article/bm//bib** elements (within the same <article> and same <bm>). At one extreme a returned element may be supported by numerous elements from all filters. At the other extreme it may only have support in one term of the returned element filter. We accept all such return elements as candidates for results. However, the returned elements are sorted first by the number of support filters that they satisfy and then by their score.

Topics that make use of AND clauses and OR clauses in the <Title> are handled by generating separate query for each clause. We do not distinguish between AND and OR and effectively allow ranking to take care of it. The heuristic justification is that if all terms appear then the score should be higher regardless of whether AND or OR were used. Also, if AND was specified, but only satisfied by some of the terms, we still want the partially matching elements as potentially valid results – after all, this may be the best that we can find.

The <Keywords> element of topic is also used – it defaults to a query on the entire <article> and considered a support to all returned elements within the same article.

3.3.2 VCAS topics

The VCAS queries were treated in exactly the same manner as SCAS queries, except that we expanded the equivalence tag interpretation.

VCAS Equivalent tags:

- **Article,bdy,fm**
- **sec|ss[1-3]|p|p[1-3]|ip[1-5]|ilrj|item-none**
- **h|h[1-2]a?|h[3-4]**
- **yr|pdt**
- **snm|fnm|au**
- **bm|bibl|bib|bb**
- **l[1-9a-e]|dl|list|numeric-list|numeric-rbrace|bullet-list**

3.3.2 CO Topics

The CO topics were handled in the same manner as CAS topics. However, all terms from both the <Title> and <Keywords> elements of the CO topic were combined to form a single query – after removing duplicate terms. The return element was assigned the default XPath signature //* which means that any element in the article was returnable (subject to support). For instance, topic 91 –

<title>Internet traffic</title>
<keywords>internet, web, traffic, measurement, congestion </keywords>

is transformed to the following query:

R|//*|//article|Internet,traffic,web,measurement, congestion

Every element with the context of //article (this includes descendents) and which contains at least one of the terms in the query is suitable for return. However, since only leaf nodes in the XML tree contain terms (with very few exceptions) there is a need to associate a score with other non-leaf elements in the tree in order to qualify them for selection. The search engine propagates the score of matching elements upwards, recursively, to ancestor nodes, in the following manner. If an ancestor has a single child it receives half the child's score. If it has multiple children it receives the sum of their scores. In this manner, for instance, a section with multiple scoring paragraphs receives a score higher than any of its paragraphs and will be ranked higher. A section having only one scoring paragraph will be ranked lower.

3.3.4 Selection by Year

Selection by year was treated as an exception. The search engine expands conditions with respect to years to allow for a range of years. It allows up to 5 years below for a Less Than condition, up to 5 years above for a Greater Than condition, and 2 years either side for an about condition. Equality is treated strictly. This is necessary for two reasons. The inverted list structure does not support range queries so it is necessary to translate such conditions to explicit values that can be searched. It is also not possible to interpret the *about* condition over <year> without some pre-conceived idea of what might be a reasonable year range.

3.3.4 Term expansion

The search engine can optionally expand keywords in one of two ways. It can perform plural and singular expansion, or it can use the full porter stem (pre-stored in the database). In the case of phrases, the program also attempts to construct an acronym. So for instance, the phrase "Information Retrieval" generates the additional term "IR". A common writing technique is to introduce an acronym for a phrase and thereafter use the acronym for brevity. For instance, at INEX, we defined "Strict Content and Structure" as "VCAS". Subsequent references are to VCAS only. So the idea here is to try and guess acronyms. We use several simple rules that attempt to manipulate the phrase initials to construct a few acronyms. If an acronym thus generated is found in the inverted list it is used as an additional term.

4. Results

Two aspects of the system were tested. The precision/recall values were measured through the standard INEX evaluation process. The performance of the distributed search engine was also tested on a distributed database.

4.1 Performance

The system was tested as a stand alone search engine in a single PC and on a distributed configuration. On a single PC (Pentium 4, 1.6GHz, 500MB RAM) the search times for topics varied between 5 seconds and one minute, depending on the number of terms and their frequency in the database.

The database can be distributed in a logical manner by placing each of the 18 journals on a different PC. Each search engine was set to return the N best results. We used a threshold N=100, but this is a run-time argument. The communications overhead of the system is about 5 seconds (pretty much fixed, given a reasonably fast connection.) The search over a single journal is very quick and takes less than 3 seconds. The INEX collection can thus be searched in less than 10 seconds even for the most elaborate topics. The total search time is pretty much upper limited by the longest search time on any of the distributed components. Nevertheless, results arrive asynchronously, so the user can view early results before the entire distributed search is complete.

The system scales up well. If the full database is duplicated on several PCs the search time is virtually constant – as long as the number of results returned is reasonably capped.

Results are ranked independently by each distributed search component. Consequently, the results can be displayed in order, either globally, or within each Journal. A difference between the single complete database and the distributed database results can arise if there are useful results in one journal that are ranked below the allowed threshold N. However, this difference will only affect the lower end of the ranked list and in any case this problem can be easily circumvented. An obvious variation is to determine the return threshold by rank rather than by count. In this manner poor results can be avoided while better results are allowed to arrive in larger numbers from fruitful searches of distributed database compartments.

5.2 Precision/Recall

The better results were obtained in the SCAS track with plural/singular term expansion. It scored an average precision (generalized) of 0.195 (rank 12/38). The Porter stemming expansion of terms produced somewhat lesser results with an average precision of 0.186. Without term expansion the results had an even lower score with an average precision of 0.174.

VCAS results are not available at the time of writing this paper.

In the CO track results were similar. The better results were obtained with full Porter stemming, with an average precision (generalized) of 0.0525 (rank 14/56). Somewhat lesser, but essentially similar results were obtained with plural/singular expansion with an average precision of 0.0519. Without term expansion the average precision was 0.0505.

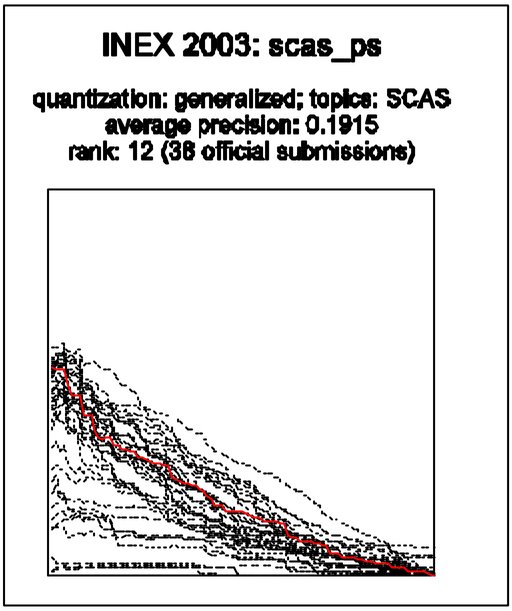


Figure 3: Plural/Singular expansion

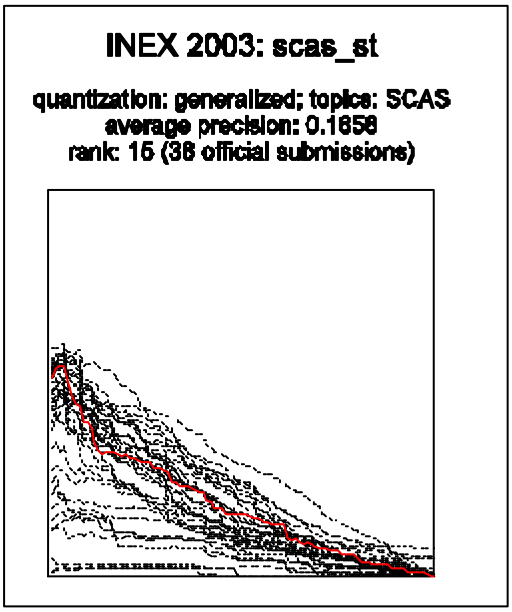


Figure 4: Full Porter stemming

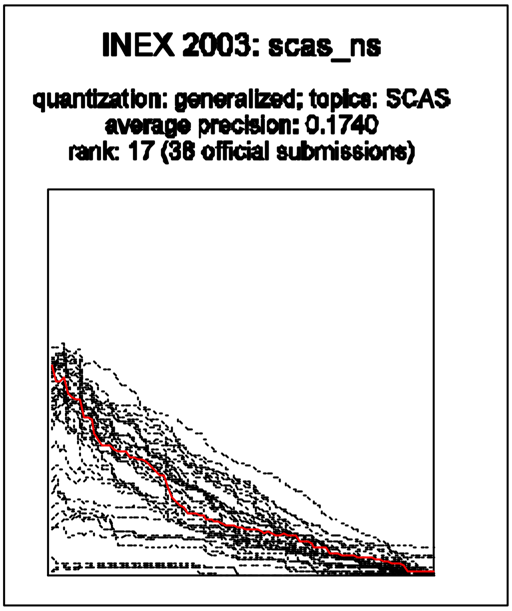


Figure 5: Without Term expansion

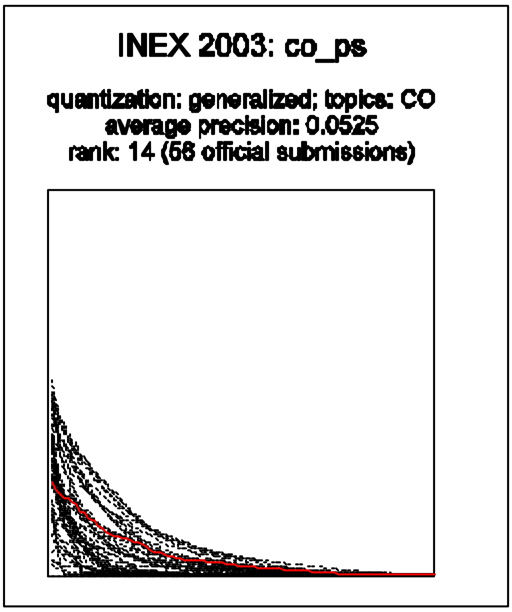


Figure 6: Full Porter stemming

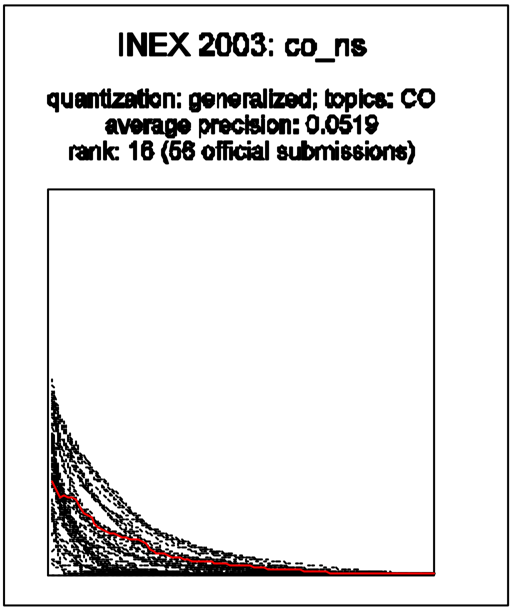


Figure 7: Plural/Singular expansion

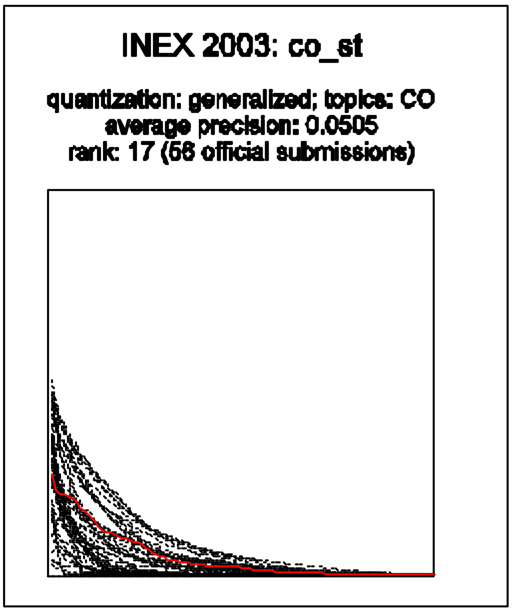


Figure 8: Without Term expansion

5. Discussion

The search engine that was developed and tested performs reasonably well in terms of precision/recall. It performs very well in terms of speed, and scales almost linearly.

Inspection of our results suggests that while the system was able to retrieve the most significant <article> elements, it fell short in terms of ranking the various descendents. With CAS queries the loose interpretation of AND, OR, and equality constraint might have contributed to violations of topic <title> XPath constraints leading to selection of undesirable elements. With CO queries the ranking heuristics that we used were generic. We only took account of abstract tree structure considerations. It might have been advantageous to also apply heuristics that are specific to the INEX collection and perceived intent of topic authors (in general, not specifically). For instance, paragraphs might be better units of retrieval than sections. More analysis and experimentation with ranking is required.

6. REFERENCES

[1] Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), DELOS Network of Excellence on Digital Libraries, ERCIM-03-W03

[2] "XML Path Language (XPath) Version 1.0" http://www.w3.org/TR/xpath

RMIT INEX experiments: XML Retrieval using Lucy/eXist

Jovan Pehcevski
School of CS and IT
RMIT University
Melbourne, Australia 3000
jovanp@cs.rmit.edu.au

James Thom
School of CS and IT
RMIT University
Melbourne, Australia 3000
jat@cs.rmit.edu.au

Anne-Marie Vercoustre
CSIRO-ICT Centre
Melbourne, Australia 3000
Anne-Marie.Vercoustre@csiro.au

ABSTRACT

This paper reports on the RMIT group's approach to XML retrieval while participating in INEX 2003. We indexed XML documents using Lucy, a compact and fast text search engine designed and written by the Search Engine Group at RMIT University. For each INEX topic, up to 1000 highly ranked documents were then loaded and indexed by eXist, an open source native XML database. A query translator converts the INEX topics into corresponding Lucy and eXist query expressions, respectively. These query expressions may represent traditional information retrieval tasks (unconstrained, CO topics), or may focus on retrieving and ranking specific document components (constrained, CAS topics). With respect to both these expression types, we used eXist to extract final answers (either full documents or document components) from those documents that were judged highly relevant by Lucy. Several extraction strategies were used that differently influenced the ranking order of the final answers. The final INEX results show that our choice for a translation method and an extraction strategy leads to a very effective XML retrieval for the CAS topics. We observed a system limitation for the CO topics resulting in the same or similar choice to have little or no impact on the retrieval performance.

Keywords

XML Search & Retrieval, eXist, Lucy, INEX

1. INTRODUCTION

During INEX 2002, different participants used different approaches to XML retrieval. These approaches were classified into three categories [1]: extending well known full-text *information retrieval* (IR) models to handle XML retrieval; extending *database management systems* to deal with XML data; and *XML-specific*, which use native XML databases that usually incorporate existing XML standards (such as XPath, XSL or XQuery). Our modular system utilises a combined approach using traditional information retrieval features with well-known XML technologies found in most native XML databases.

Lucy[1] is RMIT's fast and scalable open source full-text search engine. Lucy follows the content-based information retrieval approach and supports Boolean, ranked and phrase queries. However, Lucy's smallest unit of retrieval is a whole document, thus ignoring the structure specified using the document schema as in the XML retrieval approach. Indeed, when dealing with information retrieval from a large XML document collection, sections that belong to a document, or even smaller document components such as paragraphs, may be regarded as appropriate units of retrieval. Accordingly, it is important to have an IR-oriented XML retrieval system that will be able to identify and rank these units of retrieval.

eXist[2], an open source XML database, follows the XML-specific retrieval approach. It is the XML-specific approach that deals with both the content and the structure of underlying XML documents and incorporates keyword, Boolean and proximity search. Most of the retrieval systems that follow this approach use databases specifically built for XML. These databases are often called *native XML databases*. However, most of these systems do not support any kind of ranking of the final answers, which suggests a need of applying an appropriate retrieval strategy to determine the relevance of the answers to a given retrieval topic.

The XML retrieval approach we consider at INEX 2003 is that for many retrieval topics it appears the only way to obtain satisfactory answers is to use either *proximity* or *phrase search* support in XML retrieval systems. That is, a final answer is likely to be relevant if it contains (almost) all of the query terms, preferably in a desired order. The native XML databases, as explained above, provide all the required support to enable this functionality. However, when a native XML database needs to load and index a large XML collection, the time required to extract the most relevant answers for a given query is likely to increase significantly. Moreover, the XML database needs to determine a way to somehow assign relevance values to the final answers. Accordingly, it would be more efficient if the XML database has to index and search a smaller set of XML documents that may have previously been determined *relevant* for a particular retrieval topic. The database would then need to decide upon the most effective strategy for extracting and ranking the final answers. We have therefore decided to build a system that uses a combined IR/XML-specific retrieval approach. Our modular system effectively utilises Lucy's integrated ranking mechanism with eXist's powerful keyword search extensions. The INEX results show that our system produces effective XML retrieval for the content-and-structure (CAS) INEX topics.

[1] http://www.seg.rmit.edu.au/lucy/

[2] http://exist-db.org/

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="117" query_type="CO" ct_no="98">

<title>
Patricia Tries
</title>

<description>
Find documents/elements that describe
Patricia tries and their use.
</description>

<narrative>
To be relevant, a document/element
must deal with the use of Patricia Tries
for text search. Description of the standard
algorithm, optimised implementation and use
in Information retrieval applications are all
relevant.
</narrative>

<keywords>
Patricia tries, tries, text search,
string search algorithm,
string pattern matching
</keywords>

</inex_topic>
```

Figure 1: INEX Topic 117

2. INEX TOPICS

As in the previous year, INEX 2003 has used the same set of XML documents that comprises 12107 IEEE Computer Society articles published within the period 1997-2002 and stored in XML format. INEX 2003 also introduced a new set of ad-hoc retrieval topics which in contrast to the previous year were differently formulated. Revised relevance dimensions, *exhaustivity* and *specificity*, for assessing the relevance of the retrieval topics were also introduced.

Two types of XML retrieval topics are explored in INEX: content-only (CO) topics and content-and-structure (CAS) topics. A CO topic does not refer to the existing document structure. When dealing with CO topics, an XML retrieval system should follow certain rules that will influence the size and the granularity of a resulting document component. Not every document component can be regarded as a meaningful answer for a given query. Some of them are too short to act as meaningful answers while some of them are too broad. Thus, if an XML retrieval system shows poor performance (in terms of its effectiveness), the rules that decide upon the answer size and granularity should be changed accordingly.

A CAS topic, unlike a CO topic, enforces restrictions with respect to the existing document structure by explicitly specifying the type of the unit of retrieval (section, paragraph, or other). When dealing with CAS topics, an XML retrieval system should (in most cases) follow the structural constraints described in the topic, which will result in answers having the desired (or similar) structure. In this case, the size and the granularity of a final answer are determined in advance.

The rest of this section describes INEX topics 117 and 86, which are respectively the CO and CAS topics proposed and assessed by our group. Some issues were observed during our relevance assessments for these topics. Our final results at INEX 2003 show that these issues, when addressed correctly, significantly improve the performance of an XML retrieval system. We also discuss the implications of these INEX topics for using the combined Lucy/eXist retrieval system and report other comments and suggestions.

2.1 INEX Topic 117

Figure 1 shows the INEX CO topic 117. This topic searches for documents or document components focusing on algorithms that use Patricia tries for text search. A document or document component is considered relevant if it provides description of the standard/optimised algorithm implementation or discusses its usage in information retrieval applications.

Our first observation is that this topic (unintentionally) turned out to be a difficult one, since:

- *Patricia* (usually) represents a person's first name, rather than a data structure;
- *tries* is a verbal form, and
- keywords like *text*, *string*, and *search* appear almost everywhere in the INEX IEEE XML document collection.

The relevance assessments were long and difficult, mainly because there were too many answers (due to *Patricia* and *tries*), there were not many highly relevant answers, and the few somewhat relevant answers were hard to evaluate consistently both for exhaustivity and specificity.

For this and similar topics, it appears that the only way to obtain satisfactory results is to use either *proximity* operators or *phrase search* support in full text retrieval systems. In the context of XML, an interesting question is whether the granularity of XML document components can be used as the proximity constraint. For example, it is more likely that paragraphs containing few of the query keywords will be regarded more relevant than a document that contains all keywords in different sections. On the other side, since users expect meaningful answers for their queries, the answers are expected to be rather broad, so retrieved document components should at least constitute a section, possibly a whole document. Accordingly, an XML retrieval system should follow an effective *extraction strategy* capable of producing more relevant answers.

2.2 INEX Topic 86

Figure 2 shows the INEX CAS topic 86. This topic searches for document components (sections) focusing on electronic

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="86" query_type="CAS" ct_no="107">

<title>
//sec[about(.,'mobile electronic payment system')]
</title>

<description>
Find sections that describe technologies
for wireless mobile electronic payment systems
at consumer level.
</description>

<narrative>
To be relevant, a section must describe
security-related technologies that exist
in electronic payment systems that can be
implemented in hardware devices.
The main interests are systems that can be
used by mobile or handheld devices.
A section should be considered irrelevant
if it describes systems that are designed
to be used in a PC or laptop.
</narrative>

<keywords>
mobile, electronic payment system,
electronic wallets, e-payment, e-cash,
wireless, m-commerce, security
</keywords>

</inex_topic>
```

Figure 2: INEX Topic 86

payment technologies implemented in mobile computing devices, such as mobile phones or handheld devices. A section will be considered highly relevant if it describes technologies that can be used to securely process electronic payments in the mobile computing devices.

In order to consistently assess the relevance of the resulting document components (for this topic, most of these components were sections), two assessment rules were applied: document components focusing *only* on mobile computing devices were considered irrelevant, and document components focusing on security issues *in general* were also considered irrelevant.

It is evident from the above rules that for a document component to be considered marginally, fairly or highly relevant, it should *at least* contain a combination of some important words or phrases, such as *mobile*, *security*, *electronic payment system*, *e-payment*, and so on. In this sense, the issues encountered while assessing INEX CAS topic 86 were very similar with the ones discussed earlier for INEX CO topic 117. The only difference is that for this topic, the unit of retrieval is known in advance (`<sec>` identifies the type of document component to be retrieved), although by no means this should be regarded as a mandatory constraint, since the INEX DTD specifies different types of document components that may be regarded as sections (`sec`, `sec1`, `bm`, or even `bdy`). It is therefore reasonable to expect that the extraction strategy previously applied to the CO topics would lead to more effective results for the CAS topics. The final INEX results for the CAS topics shown later in Figure 5 confirm this expectation.

2.3 Implications of INEX topics

It is evident from the previous observations that using either Lucy or eXist will partially satisfy the information need expressed with both the CO and the CAS topics. Lucy supports phrase search and ranking, however proximity support is limited, and the unit of retrieval is a whole document. eXist supports proximity operators and phrase search, and additionally allows final answers containing any of the query terms. However, it does not rank the final answers, and unless explicitly specified in the query, it does not impose additional constraints on the granularities of the returned answers. We identify later that this missing feature represents a serious system limitation for the CO topics. Accordingly, we decided to take into account the positive aspects of both systems and build a modular system that incorporates a combined approach to XML retrieval. Section 3 describes our approach in detail.

2.4 Other comments and suggestions

As a result of our active INEX participation this year, particularly while creating the INEX topics 86 and 117 and assessing the relevance of corresponding documents and document components, we observed some additional issues.

- In proposing a retrieval topic, should a participant make a statement about what XML retrieval feature he/she is trying to evaluate?
- Should the INEX initiative start making a classification of these various features? The features that we refer here might include, for example, usefulness of existing links and references in XML documents, proximity search, selection criteria, granularity of answers, and so on.

Although the INEX 2003 assessment tool was much better than the one used in 2002, the assessment task is still very time consuming. We suggest whether less answers could be pooled for assessment and whether the assessment tool could be furthermore improved to reduce some interaction required by users. The last suggestion might for example include less required "clicks" and the ability to select a group of answers as irrelevant (regardless whether they represent documents or document components).

3. MODULAR SYSTEM ARCHITECTURE

For INEX 2003, we decided to build a modular system that uses a combined approach to XML retrieval, comprising two modules: the Lucy full-text search engine and the eXist native XML database. Before we explain our approach in detail, we briefly summarise the most important features of both modules.

3.1 *Lucy* search engine

Lucy is a compact and fast text search engine designed and written by the Search Engine Group at RMIT University. Although Lucy primarily allows users to index and search HTML[3] (or TREC[4]) collections, we have successfully managed to index and search the entire INEX IEEE collection of XML documents. However, Lucy's primary unit of retrieval is a whole document and currently it is not capable of indexing particular document components, such as `<author>`, `<sec>`, and `<p>`. Lucy has been designed for simplicity as well as speed and flexibility, and its primary feature, which is also evident in our case, is the ability to handle a large amount of text. It implements an inverted index structure, a search structure well researched and implemented in many existing information retrieval systems. Witten et al. [8] provide a detailed explanation for efficient construction of an inverted index structure such as implemented in Lucy.

Lucy is a fast and scalable search engine, and incorporates some important features such as support for Boolean, ranked and phrase querying, a modular C language API for inclusion in other projects and native support for TREC experiments. It has been developed and tested under the Linux operating system on an Intel-based platform, and is licensed under the GNU Public License.

3.2 *eXist*: a native XML database

Since January 2001, when eXist [3] started as an open source project, developers are actively using this software for various purposes and in different application scenarios. We use eXist as a central part of our modular XML retrieval system. eXist incorporates most of the basic and advanced native XML database features, such as full and partial keyword text searches, search patterns based on regular expressions, query terms proximity functions and similar features. Two of eXist's unique features are efficient index-based query processing and XPath extensions for full-text search.

Index-based query processing. For the purpose of evaluating XPath expressions in user queries, conventional native XML database systems generally implement top-down or bottom-up traversals of the XML document tree. However, these approaches are memory-intensive, resulting in slow query processing. In order to decrease the time needed for processing the queries, eXist uses an inverted index structure that incorporates numerical indexing scheme for identifying the XML nodes in the index. This feature enables eXist's query engine to use fast path join algorithms for evaluating XPath expressions. Meier [3] provides detailed technical explanation of this efficient index-based query processing implementation in eXist.

XPath extensions for full-text searching. Standard XPath implementations do not provide very good support for querying document-centric XML documents. Document-centric documents, as oppose to data-centric ones that usually contain machine-readable data, typically include mixed content and longer sections of text. eXist implements a number of XPath extensions to efficiently support document-centric queries, which overcome the inability of standard XPath functions (such as `contains()`) to produce satisfactory results. For example, the `&=` operator selects document components containing *all* of the space-separated terms on the right-hand side of the argument. `|=` operator is similar, except it selects document components containing *any* of the query terms. In the next section we provide examples of the way we used these operators in the INEX topic translation phase.

eXist is a lightweight database, completely written in Java and may be easily deployed in several ways. It may run either as a stand-alone server process, or inside a servlet-engine, or may be directly embedded into an existing application.

3.3 A combined approach to XML retrieval

Section 2 observes the implications of the INEX topics that influenced our choice for a combined approach to XML retrieval. However, due to the advanced retrieval features described previously it becomes evident that using eXist alone should suffice in satisfying the XML retrieval needs. Indeed, some applications have shown that eXist is already able to address real industrial needs [3]. Despite all these advantages, we were not able to use eXist as the *only* XML retrieval system for two main reasons: first, we were using eXist version 0.9.1, which did not manage to load and index the entire IEEE XML document collection needed for INEX, and second, although we could retrieve relevant pieces of information from parts of the IEEE document collection, eXist does not assign relevance values to the retrieved answers. Accordingly, since *ranking* of the retrieved answers is not supported, we decided to undertake a combined XML retrieval approach that utilises different *extraction strategies* to rank the answers. With respect to a specific extraction strategy, a document component may represent a highly ranked answer if it belongs to a document that has previously been determined relevant for a particular retrieval topic.

Figure 3 shows our combined approach to XML retrieval. The system has a modular architecture, comprising two modules: Lucy and eXist. We use INEX topic 86, as shown in Figure 2, to explain the flow of events.

First, the INEX topic is translated into corresponding queries understandable by Lucy and eXist, respectively. Depending on the type of the retrieval topic (CO or CAS), the topic translation utility follows different rules. For the INEX CO topics, such as topic 117 shown in Figure 1, queries that are sent to both Lucy and eXist include only terms that appear in the `<Keywords>` part of the INEX topics. For the INEX CAS topics, as shown in Figure 3, query terms that appear in both `<Title>` and `<Keywords>` parts of the INEX topics were used.

For example, we use the query terms from the `<Keywords>` part of the INEX topic 86 to formulate the Lucy query:

```
.listdoc
'mobile "electronic payment system"
"electronic wallets" e-payment e-cash wireless
 m-commerce security'
```

[3]http://www.w3.org/MarkUp/
[4]http://trec.nist.gov/

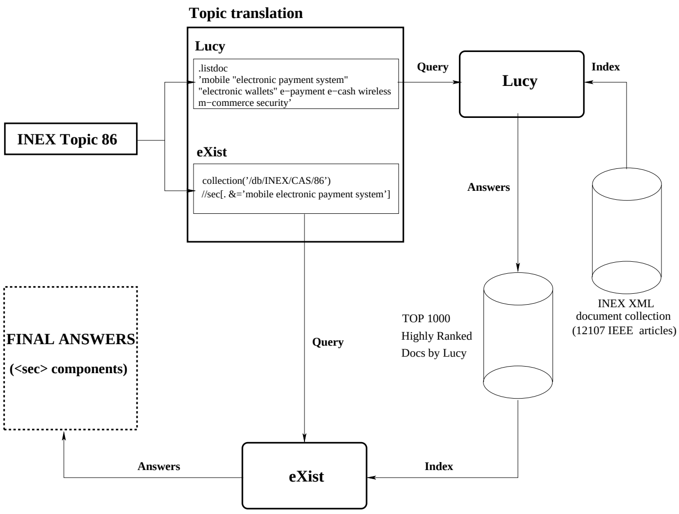


Figure 3: A modular system architecture.

However, before submitting a query to the system, the INEX document collection needs to be indexed. We use Lucy to create an inverted index from all the documents in the large IEEE XML collection. We then search this indexed data by entering the queries derived from the translation rules, as explained above. For the purpose of ranking its answers for a given query, Lucy uses a variant of the Okapi BM25 [5] probabilistic ranking formula. Okapi BM25 is one of the most widely used ranking formula in information retrieval systems. It is thus expected that, for a given INEX topic, Lucy will be able to retrieve highly relevant XML documents early in the ranking. Therefore, for each INEX topic, we retrieve (up to) 1000 highest ranked XML documents by Lucy. It is our belief that the information contained in these documents is sufficient to satisfy the information need expressed in the corresponding INEX topic. However, at this phase of development, Lucy's only unit of retrieval is a whole document. Accordingly, for a particular INEX topic, we still have to extract the relevant parts of these highly ranked documents. Wilkinson [7] shows that simply extracting components from highly relevant documents leads to poor system performance. Indeed, there may be cases when a section belonging to highly ranked document is irrelevant as opposed to a relevant section belonging to lowly ranked document. However, we believe that the retrieval performance of a given system may be improved using a suitable *extraction strategy*. We implemented several extraction strategies using eXist's XPath extensions. We provide examples how we use these XPath extensions while translating INEX topic 86 as follows.

For INEX CAS topics in general, and INEX topic 86 in particular, the terms that appear in the `<Title>` part are used to formulate eXist queries. However, since a document component is likely to be relevant if it contains *all* or *most of* the query terms that appear in the `<Title>`, we undertake several extraction strategies as described in detail in Section 4, where we explain how we construct our INEX runs. In general, these strategies depend on the combined usage of Boolean `AND` and `OR` operators, implemented using the `&=` and `|=` operators in eXist, respectively. Accordingly, the INEX topic 86 may be translated either as:

```
collection('/db/INEX/CAS/86')
//sec[. &='mobile electronic payment system']
```

if one wants *all* query terms to appear in the resulting section, or:

```
collection('/db/INEX/CAS/86')
//sec[. |='mobile electronic payment system']
```

if one wants *any* of the query term to appear in the resulting section.

We follow the first translation rule for our example in Figure 3. Final answers will thus constitute `<sec>` document components (if any) that contain all the query terms. By following this rule, we reasonably expect these document components to represent relevant answers for the INEX topic 86. On the other hand, it is clear that if the second translation rule is applied for the same topic, it may produce very many irrelevant answers as well as some further relevant answers. Accordingly, it is very important to decide upon the extraction strategy that will yield in highly relevant answers for a given INEX topic. We discuss the results for different extraction strategies in the following section.

4. INEX RUNS AND RESULTS

The retrieval task performed by the participating groups at INEX 2003 was defined as ad-hoc retrieval of XML documents. In information retrieval literature this type of retrieval involves searching a static set of documents using a new set of topics, which represents an activity very commonly used in library systems.

Within the ad-hoc retrieval task, INEX 2003 defines additional sub-tasks. These represent a *CO* sub-task, which involves content-only (CO) topics and a *CAS* sub-task, which involves content-and-structure (CAS) topics. The CAS sub-task comprises a *SCAS* sub-task and a *VCAS* sub-task. The SCAS sub-task requests that the structural constraints in a query must be *strictly* matched, while *VCAS* allows the structural constraints in a query to be treated as *vague* conditions.

At INEX 2003, for each topic belonging to a particular sub-task up to 1500 answers (full documents or document components) were required to be retrieved by the participating groups. In order to assess the relevance of the retrieved answers, the revised relevance dimensions (exhaustivity and specificity) need to be quantized in a single relevance value. INEX uses two quantization functions: *strict* and *generalised*. The strict function can be used to evaluate whether a given retrieval method is capable of retrieving highly relevant and highly focused document components, while the generalised function credits document components according to their *degree of relevance* (by combining the two relevance dimensions, exhaustivity and specificity).

Our group submitted 6 official runs to INEX 2003, 3 for each CO and SCAS sub-task, respectively. Figures 4 and 5 show

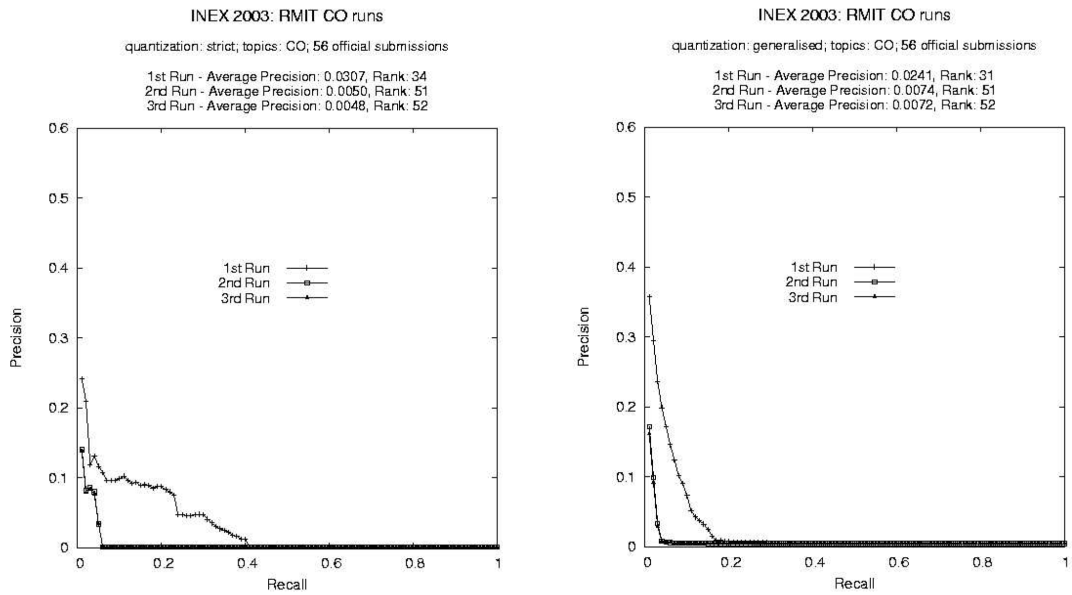


Figure 4: Results for the RMIT CO runs using both strict and generalised quantization functions

the results for both the CO and SCAS runs when both strict and generalised quantization functions are used. The rankings of the runs are determined according to the average precision over 100 recall points considering each corresponding INEX topic. Two of our three runs for each sub-task were automatically constructed while one was manually. The automatic runs were constructed using the translation rules explained in the previous section. We manually constructed the other runs in order to produce more meaningful queries for each INEX topic. Each run was constructed by using elements in the following answer lists: [A] that uses eXist's `&=` (logical `AND`) operator and enforces strict satisfaction of logical query conditions (the elements that belong to the answer list [A] will therefore represent document components containing *all* the query terms or phrases); [B] that uses the `|=` (logical `OR`) operator, "relaxes" the query conditions and allows for document components containing *any* of the query terms or phrases; and a combined answer list that contains the elements in the answer list [A] followed by the elements in the answer list [B–A].

Three retrieval runs were submitted for the CO sub-task. We constructed the first CO run by retrieving the 1500 highest ranked documents for each INEX topic. As described in the previous section, the `<Keywords>` part of each INEX topic was automatically translated as an input query to the Lucy search engine. The final rank of a document was then determined by its similarity with the given query as calculated by Lucy using a variant of Okapi BM25. As shown in Figure 4 this run performed better than the other two CO runs in both cases when strict and generalised quantization functions are used, which suggests that a whole document is often likely to be considered a preferable answer for an INEX CO topic.

For the other two runs, for each INEX CO topic we first used Lucy to extract (up to) the 1000 highest ranked documents. Then we used eXist to index and retrieve the final answers from these documents. We reasonably expected that the most relevant document components required to be retrieved for each INEX topic were very likely to appear within the 1000 highest ranked documents. Since the CO topics do not impose constraints over the structure of resulting documents or document components, we used the `//**` eXist construct in our queries. The "`**`" operator in eXist uses a heuristic that retrieves answers with different sizes and granularities. For our second CO run, the `<Keywords>` part of each topic was automatically translated as an input query to the eXist database, and its final answer list includes only elements from the answer list [B]. We used the manual translation process for our third run, where the final answer list includes the elements in the answer list [A] followed by the elements in the answer list [B–A]. Although we expected the third run to perform better than the second, Figure 4 shows that both these runs performed poorly in both cases when strict and generalised quantization functions are used, regardless of choices for the translation method and the extraction strategy. At this phase of development, the heuristic implemented in the "`**`" operator in eXist is not able to determine the most meaningful units of retrieval nor influence the desired answer granularity for a particular CO topic. Next we show that this is not the case for the CAS topics, where the type of the unit of retrieval is determined in advance and the choices for the translation method and the extraction strategy have a significant impact on the system's performance.

Three runs were submitted for the SCAS sub-task. As discussed previously, both `<Keywords>` and `<Title>` parts from INEX CAS topics were used to generate the input queries for Lucy and eXist, respectively. Our first SCAS run was automatic and its final answer list includes the elements in the answer list [A] followed by the elements in the answer list [B-A]. The queries for the second SCAS run were manually constructed and its final answer list includes the elements from the same answer lists as for the first run.

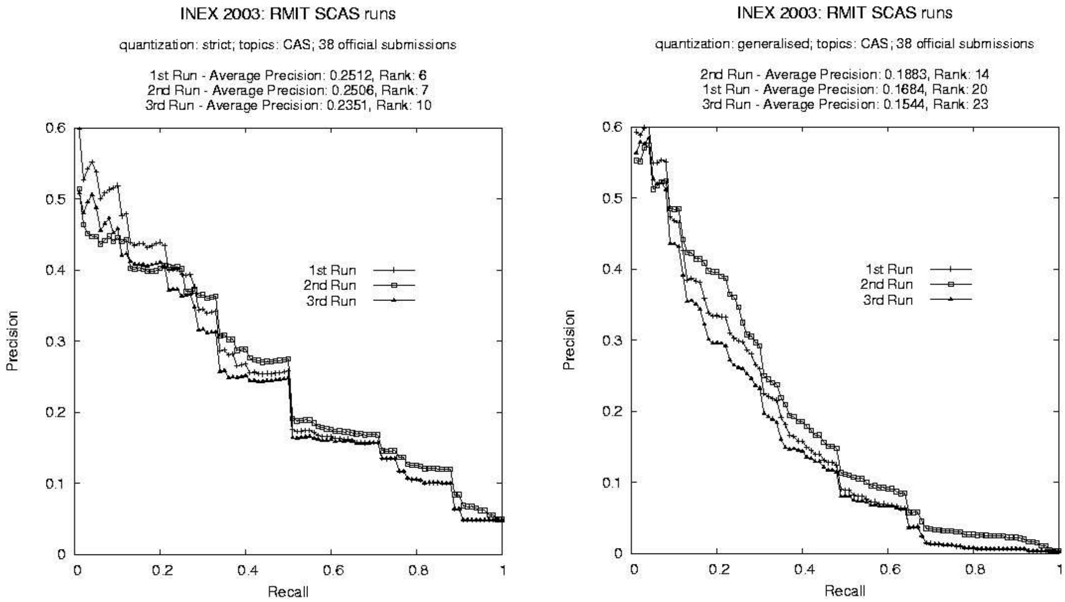


Figure 5: Results for the RMIT SCAS runs using both strict and generalised quantization functions

Figure 5 shows that these runs performed relatively better when using a strict quantization function compared with the runs from other participating groups at INEX 2003. Since the type of the unit of retrieval is determined in advance for the SCAS runs, the choice of the extraction strategy implemented in both runs appears to be very effective for retrieving highly exhaustive and highly specific document components. It can be observed that our system performs slightly more effective for the first than for the second run (6th compared to 7th out of 38 systems), and the first run performs better for recall values lower than 0.2. However, the choice of the translation method has an effect on the system's performance for recall values greater than 0.3, where the second run performs better than the first run. Figure 5 also shows that the choice of the extraction strategy is not as effective when using a generalised quantization function, where marginally/fairly exhaustive or marginally/fairly specific document components are regarded as partly relevant answers. Indeed, the ranks for both runs when evaluated using the generalised quantization function are not among the ten highest ranked INEX runs. In this case, the choice of the translation method results in second run performing better than the first run overall.

The third SCAS run was automatic, however its final answer list includes only the elements from the answer list [B]. By choosing this strategy we reasonably expected some irrelevant answers in the final answer list, but we hoped to find more relevant components in highly ranked documents. Indeed, as Figure 5 shows, irrespective of whether a strict or a generalised quantization function is used, our retrieval system is ranked lower for the third SCAS run compared to the previous two runs.

5. LIMITATIONS OF OUR SYSTEM

Previous sections describe the XML retrieval approach that we implemented while participating in INEX 2003. However, during different phases of our INEX involvement, particularly while constructing the INEX runs and assessing the relevance of retrieved results, we observed several system limitations. Although they can and should be considered as a weakness of our approach, the fact that we are able to identify them influences our future research directions. Some of these limitations include the following.

No IR ranking of the final answers. The choice of implementing an extraction strategy that may influence the rank of a final answer suggests that our system does not consider an IR ranking score for a particular answer. Although for a given INEX topic Lucy ranks the XML documents in a descending order of their query similarity, the unit of retrieval represents a whole document, and there is no support for existing XML technologies. eXist, on the other hand, has a tight integration with existing XML development tools and technologies, but does not rank the final answers according to their query similarity. We have thus decided that a particular extraction strategy should influence the final ranking score for a resulting document or document component. We have decided upon different extraction strategies while we constructed our INEX runs, and have shown that for the CAS topics some of them have a significant impact on the retrieval performance of our modular system.

Complex usage. Since our system has a modular architecture that incorporates a combined IR/XML-specific oriented approach to XML retrieval, its usage is very complex. It comprises two different retrieval modules (Lucy and eXist), each having different internal architectures and rules of use. Instead, it would be preferable to have only one system that incorporates the best features from the above modules.

Significant space overhead. The size of the INEX IEEE XML document collection takes around 500MB disk space. The inverted index file maintained by Lucy additionally takes

20% of that space. For each topic, (up to) 1000 XML documents are indexed by eXist, which adds up to approximately 12% of the space for the INEX collection. Although both Lucy and eXist implement efficient retrieval approaches, it becomes evident that their combination leads to significant disk space overhead. As for the previous limitation, one system that can deal with the above issues would also be preferable.

6. RELATED WORK

Even before INEX, the need for information retrieval from XML document collections had been identified in the XML research community. As large XML document collections become available on the Web and elsewhere, there is a real need for having an XML retrieval system that will efficiently and effectively retrieve information residing in these collections. This retrieval system will need to utilise some form of an XML-search query language in order to meet the growing user demand for information retrieval. Thus, the needs and requirements for such a query language have to be carefully identified and appropriately addressed [4].

At INEX 2002 the CSIRO group proposed a similar approach to XML retrieval. Their XML retrieval system uses a combination of a selection and a post-processing module. Queries are sent to PADRE, the core of CSIRO's Panoptic Enterprise Search Engine[5], which then ranks the documents and document components on the basis of their query similarity. In contrast to Lucy, whose primary unit of retrieval is a whole document, PADRE combines full-text and metadata indexing and retrieval and is capable of indexing particular document components, such as `<author>`, `<sec>` and `<p>`. Different "mapping rules" determine what metadata field is used to index the content of a particular document component. A post processing module was then used to extract and re-rank the final answers from documents and document components returned by PADRE [6].

In an effort to reduce the number of document components in an XML document that may represent possible answers for a given query, Hatano et al. [2] propose a method for determining the preferable units of retrieval from XML documents. We consider investigating these and similar methods for improving the effectiveness of our system for the CO topics.

7. CONCLUSION AND FUTURE WORK

We have described our combined approach to XML retrieval that we used during the INEX 2003 participation. Our retrieval system implements a modular architecture, comprising two modules: Lucy and eXist. For each INEX topic, we used Lucy, a full-text search engine designed by the Search Engine Group at RMIT, to index the IEEE XML document collection and retrieve the top 1000 highly ranked XML documents. We then indexed those documents with eXist, and implemented different topic translation methods and extraction strategies in our INEX runs. The INEX results show that these methods and strategies result in an effective XML retrieval for the CAS topics. Since our system is not yet able to identify the preferred granularities for the final answers, the methods and strategies are not as effective for the CO topics. Further investigations need to be done in order to improve this functionality.

We have also observed several limitations of our modular system. In order to overcome these limitations, we intend to implement a *full* XML information retrieval system that will incorporate the most advanced features of Lucy and eXist. We believe the resulting system will lead to a more accurate and interactive XML retrieval.

[5]http://www.panopticsearch.com

Acknowledgements

We would like to thank Wolfgang Meier for providing an assistance with using eXist and to Falk Scholer, Nick Lester, Hugh Williams and other members of the Search Engine Group at RMIT for their useful suggestions and support with using Lucy.

8. REFERENCES

[1] N. Govert and G. Kazai. Overview of the Initiative for the Evaluation of XML retrieval (INEX) 2002. In *Proceedings of the First INitiative for the Evaluation of XML Retrieval (INEX) Workshop*, Dagstuhl, Germany, December 2002.

[2] K. Hatano, H. Kinutani, M. Watanabe, M. Yoshikawa, and S. Uemura. Determining the Unit of Retrieval Results for XML Documents. In *Proceedings of the First INitiative for the Evaluation of XML Retrieval (INEX) Workshop*, Dagstuhl, Germany, December 2002.

[3] W. Meier. eXist: An Open Source Native XML Database. In *A. B. Chaudri, M. Jeckle, E. Rahm, R. Unland (editors): Web, Web-Services, and Database Systems. NODe 2002 Web- and Database-Related Workshops*, Erfurt, Germany, October 2002.

[4] J. Pehcevski, J. Thom, and A.-M. Vercoustre. XML-Search Query Language: Needs and Requirements. In *Proceedings of the AUSWeb 2003 conference*, Gold Coast, Australia, July 2002. `http://ausweb.scu.edu.au/aw03/papers/thom/`.

[5] S. Robertson and S. Walker. Okapi at TREC-8. *NIST Special Publication 500-246: The Eight Text REtrieval Conference (TREC-8)*, November 1999.

[6] A.-M. Vercoustre, J. A. Thom, A. Krumpholz, I. Mathieson, P. Wilkins, M. Wu, N. Craswell, and D. Hawking. CSIRO INEX experiments: XML Search using PADRE. In *Proceedings of the First INitiative for the Evaluation of XML Retrieval (INEX) Workshop*, Dagstuhl, Germany, December 2002.

[7] R. Wilkinson. Effective Retrieval of Structured Documents. In *W.B. Croft and C.J. van Rijsbergen (editors): Proceedings of the 17th Annual International Conference on Research and Development in Information Retrieval*, Dublin, Ireland, July 1994.

[8] I. Witten, A. Moffat, and T.C.Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 2nd edition, 1999.

An Evaluation of INEX 2003 Relevance Assessments

Kenji Hatano†, Hiroko Kinutani‡, Masahiro Watanabe§
Yasuhiro Mori♮, Masatoshi Yoshikawa♮, Shunsuke Uemura†

† Nara Institute of Science and Technology, Japan
{hatano,uemura}@is.aist-nara.ac.jp
‡ Japan Science and Technology Agency, Japan
kinutani@dblab.is.ocha.ac.jp
§ The National Institute of Special Education, Japan, masahiro@nise.go.jp
♮ Nagoya University, Japan
mori@dl.itc.nagoya-u.ac.jp, yosikawa@itc.nagoya-u.ac.jp

ABSTRACT

We have developed a keyword-based XML portion retrieval system based on statistics of XML documents to enhance overall performance. Currently, relevance assessments for keyword-based XML portion retrieval systems are provided only by INEX project; thus we evaluate our system utilizing CO topics of the INEX 2003 relevance assessments. However, in the case of using some CO topics, our system performed poorly in it's retrieval accuracy; consequently average precision of our system was not promising. In this paper, we analyze CO topics of the INEX 2003 relevance assessments based on statistics of answer XML portions and report our experimental results and requirements to the relevance assessments.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance evaluation (efficiency and effectiveness)*

General Terms

Information retrieval, Performance evaluation

Keywords

Keyword-based XML portion retrieval, Performance evaluation, Analysis of relevance assessments

1. INTRODUCTION

Extensible Markup Language (XML) [5] is becoming widely used as a standard document format in many application domains. In the near future, we believe that a great variety of documents will be produced in XML; therefore, in a similar way to developing Web search engines, XML information retrieval systems will become very important tools for users wishing to explore XML documents on the Internet.

XQuery [4], which is proposed by World Wide Web Consortium (W3C), is known as a standard query language for retrieving portions of XML documents. Using XQuery, users can issue a flexible query consisting of both some keywords and XPath notations[1]. If users have already known knowledge of the structure of XML documents, users can issue XQuery-style queries. However, there are a lot of XML documents whose XML schemas are different to each other; thus nobody can issue such a formulated query into information retrieval systems. As a result, we believe that XML retrieval systems should employ a much simpler form of query such as keyword search services. Keyword search services enable information retrieval by providing users with a simple interface. It is the most popular information retrieval method since users need to know neither a query language nor the structure of XML.

From the aforementioned background on the XML retrieval, close attention has recently been paid to a keyword-based XML portion retrieval system. Some keyword-based XML portion retrieval systems have already been available. They assume the existence of Document Type Definition (DTD) or XML schema of XML documents, so that they can deal with only one type of XML documents. It is true that DTD and XML schema facilitate enhancing retrieval accuracy and retrieval speed of keyword-based XML portion retrieval systems. However, XML documents on the Internet do not always include DTD or XML schema; thus they cannot deal with multiple types of XML documents whose structures are different to each other. XML documents on the Internet feature many types of document structures; consequently, a next generation of information retrieval system has to treat XML documents whose structures are different.

To cope with the problems described above, we have developed a keyword-based XML portions retrieval system using statistics of XML documents [13]. In XML portion retrieval, we assume that users can explicitly specify query keywords; thus, we believe that the size of retrieval results become small compared to document retrieval. Moreover, retrieval results of XML portion retrieval system should be semantically consolidated granularity of XML documents. In short, we believe that extremely small XML portions are not semantically consolidated. Therefore, we designed our XML portion retrieval system that can return small and semantically consolidated XML portions as retrieval results. However, our system performed poorly in its retrieval accuracy based on INEX 2003 relevance assessments; consequently average precision of our system was not promising

[1] Currently, XML Query working group is just starting to develop full-text search functions [2, 6].

score. According to [15], the INEX 2002 relevance assessments tended to regard large-size XML portions as correct retrieval results. This fact does not meet the purpose of our XML portion retrieval; thus it could be that retrieval accuracy of our system performed poorly if the INEX 2003 relevance assessments have the characteristics similar to the previous one. Therefore, we have to evaluate validity of the INEX 2003 relevance assessments.

In this paper, we analyze the INEX 2003 relevance assessments based on their statistics. We believe that the analyses of the relevance assessments show their own characteristics, and also help to construct a next version of the relevance assessments.

The remainder of this paper is organized as follows. First, we describe our keyword-based XML portion retrieval system in Section 2. Then, we report analyses of the INEX 2003 relevance assessments in Section 3 and discuss the validity of them in Section 4. Finally, we conclude this paper in Section 5.

2. OUR XML PORTION RETRIEVAL SYSTEM

In this section, we introduce retrieval model and purpose of our keyword-based XML portion retrieval system, and also explain our observation of XML portion retrieval.

2.1 Data Model and Retrieval Model

The data model of our system is similar to the XPath data model [7] for the sake of simplicity because XML is modeled as a hierarchical tree. The only difference between the XPath data model and ours is that attribute node is regarded as a child of element node[2].

In the meanwhile, the retrieval model of our system bears a resemblance to the proximal nodes model [18] for the sake of easy understanding. In the simplest terms, our logical model of an XML portion is a sub-tree whose root node is an element node. We can identify XML portions by their reference numbers derived from document order; therefore, users can obtain retrieval results selected from XML portions, which are identified by their reference number.

2.2 The Purpose of Our System

We can identify two types of keyword-based XML portion retrieval systems. In this paper, we call these two types of keyword-based XML portion retrieval systems *XML retrieval systems* and *XML search engines* for the sake of convenience. The former is based on structured or semi-structured database systems with keyword proximity search functions that are modeled as labeled graphs, where the edges correspond to the relationship between an element and a sub-element and to `IDREF` pointers [1, 12, 14]. Dealing with XML documents as XML graphs facilitates developing keyword-based information retrieval systems which are able to do retrieval processing efficiently. On the other hand, the latter has been developed in the research field of information retrieval [8, 10], and enables us to retrieve XML portions without indicating element name of XML documents. The large difference between the XML retrieval systems and the XML search engines is data characteristics of their retrieval targets. In short, we think that the former focuses mainly on *data-centric* XML documents, whereas the latter deals with *document-centric* ones[3]. In the meanwhile, both the XML retrieval systems and the XML search engines assume the existence of DTD or XML schema of XML documents in either research. It is a fact that DTD and XML schema facilitate enhancing retrieval accuracy and retrieval speed of their systems. However, there are some problems of searching XML portions on the Internet described in Section 1; thus other types of XML retrieval systems are required. Consequently, the XML retrieval systems in the future have to deal with XML documents whose structures are different.

In order to meet the needs of new architecture of XML retrieval systems, we have developed a keyword-based XML portions retrieval system using statistics of XML documents [13]. Our system focuses on retrieval of document-centric XML documents rather than that of data-centric ones. Our system does not utilize any information in relation to element name of XML documents, whereas the systems introduced above take advantage of the information for querying and indexing of XML documents. In our approach, XML documents must be divided into portions in order to develop a keyword-based XML retrieval system. Because XML is a markup language, XML documents can be automatically divided into their portions using their markup [16]; however, the problem which the number of the portions becomes huge is caused. In other words, it takes very long time to retrieve XML portions related to a keyword-based query using our approach. For this reason, we have to determine semantically consolidated granularity of XML documents as retrieval targets using the size of XML portions, and have to reduce the number of XML portions indexed by our XML retrieval system.

2.3 Evaluating Our System based on INEX 2003 Relevance Assessments

In this section, we report the retrieval accuracy of our keyword-based XML portion retrieval system based on INEX 2003 relevance assessments. The relevance assessments defined two metrics, strict and generalized; thus we performed experimental evaluations based on both metrics. The metrics have two criteria, “exhaustiveness” and “specificity,” for quality metrics of IR applications. The way how recall and precision is computed is described in a report [9][4]. Based on the metrics, we drew recall-precision curves for evaluation of XML portion retrieval system. Figure 1 and 2 show recall-precision curves of our system based on INEX 2003 relevance assessments. In these figures, n means the minimum number of tokens which is defined to eliminate extremely small XML portions from retrieval targets[5]. In short, as n becomes larger, the retrieval accuracy of our system becomes higher as well as the retrieval speed of our system becomes faster.

[2]If the element node has some attribute nodes that have brotherhood ties, they are owned by the element node.

[3]There is a data-centric and a document-centric view of XML described in [3].

[4]Another way is also available described in a technical report [11]; however, we did not apply it in this paper.

[5]The size of XML portions is proportional to the number of tokens contained in the XML portions.

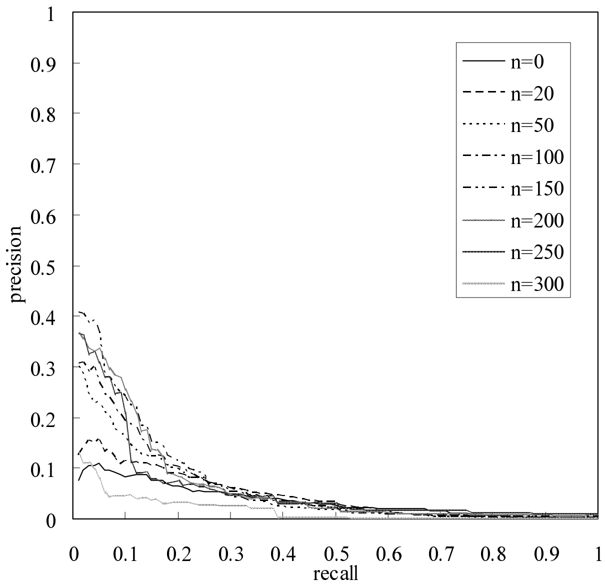


Figure 1: Evaluation of our system based on INEX 2003 relevance assessments (strict).

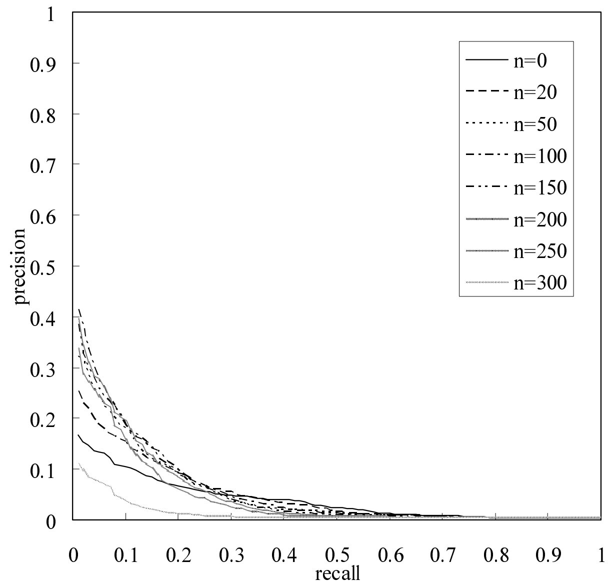


Figure 2: Evaluation of our system based on INEX 2003 relevance assessments (generalized).

As shown in these figures, our keyword-based XML portion retrieval system may be performing poorly, because the level of the recall-precision curves are relatively low among the participants of INEX project. Although we recognize the problems inherent in our system[6], it is thought that the problems may be not only in our system, but also in the relevance assessments. If the relevance assessments tend to regard large-size XML portions as correct retrieval results, our system will be a poorly-performing XML portion retrieval system, because our system tends to retrieve small XML portions except extremely small ones as retrieval results described in Section 1. As a matter of fact, in the case of threshold of the number of tokens is between 100 and 150, our system works properly (see Table 1). From our viewpoint, we think that this number of tokens is very large for XML portion retrieval. This is because the number of tokens is comparable to XML portions whose root node is `ss1`, `ss2`, or `ss3` as shown in Table 2. We have designed our XML portion retrieval to enable to retrieve XML portions corresponding to XML portions whose size is less than (sub)sections of INEX document collection as retrieval results. Therefore, retrieval results of our system will be the XML portions smaller than answer XML portions[7].

Table 1: Average precision of our system.

n	strict	generalized
0	0.0356	0.0390
20	0.0436	0.0476
50	0.0502	0.0505
100	0.0568	<u>0.0568</u>
150	<u>0.0669</u>	0.0525
200	0.0630	0.0503
250	0.0572	0.0416
300	0.0163	0.0130

From the aforementioned points, we choose the relevance assessments suitable for our system, and reevaluate retrieval accuracy of our system based on revised version of the relevance assessments.

3. ANALYSES OF INEX RELEVANCE ASSESSMENTS

3.1 Analyses of the Relevance Assessments

As we described in previous section, we think that the INEX 2003 relevance assessments may work against XML portion retrieval systems which tend to regard small-size XML portions as correct retrieval results. Consequently, we analyze statistics of answer XML portions of the relevance assessments. In this section, we define the answer XML portions as the XML portions whose exhaustiveness and specificity are 3. Our system can deal with only content-only (CO) topics of the relevance assessments; thus answer XML portions of CO topics are analyzed.

Figure 3 shows analyses of CO topics of the INEX 2003 relevance assessments. Area charts mean maximum, average, and minimum number of tokens of answer XML portions, and a line chart means the number of answer XML portions. As shown in Figure 3, we firstly found that five CO topics of the relevance assessments (whose topic are #92, #100, #102, #115, and #121) did not have answer XML portions whose exhaustiveness and specificity are 3. It is

[6] Our system cannot calculate similarities between a query and XML portions using both contents and structures of XML documents.

[7] Of cause, this is our opinions. In [17], the authors claimed that 500 words is valid for answer XML portions. The proper size of answer XML portions depends on retrieval purposes.

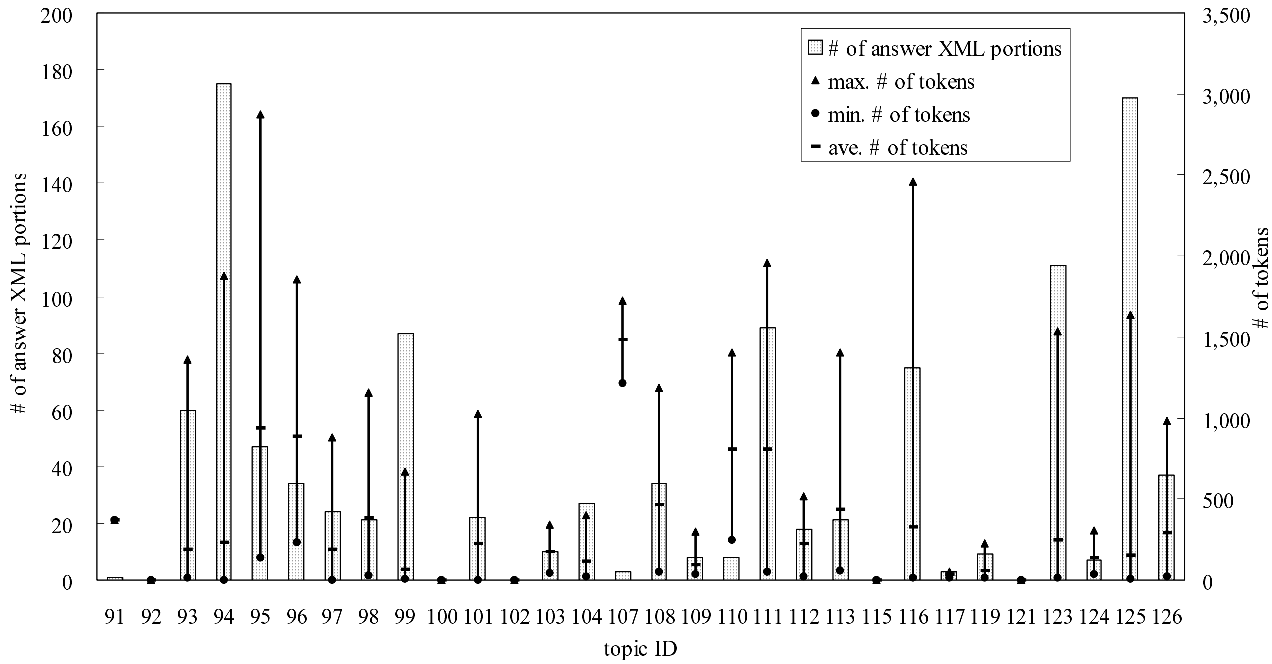


Figure 3: Analyses of the INEX 2003 relevance assessments.

doubtful that these topics were adopted as the CO topics of the relevance assessments. Moreover, we also found that average number of tokens of almost all CO topics was more than 100. Especially, average number of tokens of the CO topics whose IDs are #95, #96, #107, #110, and #111 was 500 and above; thus these CO topics distinctly work against our system. Furthermore, the number of answer XML portions substantially differs with each CO topic. We think that the CO topics with few answer XML portions are not inappropriate for the relevance assessments.

From the aforementioned points, we choose 14 CO topics (#93, #94, #97, #98, #99, #101, #104, #108, #112, #113, #116, #123, #125, #126) as the topics suitable for our system, and reevaluate retrieval accuracy of our system based on revised version of the relevance assessments.

3.2 Reevaluation of Our System

Figure 4 and 5 show recall-precision curves of our system based on revised versions of INEX 2003 relevance assessments. Moreover, Table 3 shows average precisions of each recall-precision curve. As compared with previous evaluations described in Section 2.3, retrieval accuracy of our system is improved about 3.55% on this experiment. This fact indicates that the INEX 2003 relevance assessments also tend to regard large-size XML portions as correct retrieval results; thus our system cannot return the XML portions relevant to CO topics of the relevance assessments. Needless to say, we do not know that retrieval accuracy of our system is better than that of other INEX participants' systems; however, we can confirm controversial points of the relevance assessments for our system.

To reduce the scope of such arguments, we have to clarify what XML portion retrieval is. It is difficult to define the granularity of XML documents; however, we think that determining such a retrieval unit for XML portions is the most important task for INEX project. Of cause, the retrieval unit for XML portions is differ for retrieval purposes of INEX participants; thus, we should determine the unit of XML portions for each retrieval purpose. In the case of our keyword-based XML portion retrieval system, small and semantically consolidated XML portions are defined as correct retrieval results; thus, we think that we are just forced to use the revised relevance assessments.

4. DISCUSSION

As we described in previous section, retrieval targets of INEX document collection depend on retrieval purpose of XML portion retrieval systems. In the case of our system, small and semantically consolidated XML portions are defined as the retrieval targets. On the other hand, Kamps et al. concluded in [15] that users and assessors of the INEX 2002 relevance assessments regard the XML portions whose root node is `article` as retrieval targets of INEX document collections; thus we think that the systems which tend to regard large-size XML portions as retrieval results gain the upper hand in retrieval accuracy. That is to say, nobody may be able to evaluate XML portion retrieval systems accurately using the relevance assessments.

We think that the INEX 2003 relevance assessments also contain the problem which the INEX 2002 relevance assessments have. In this section, we make specific mention of the issues of the INEX 2003 relevance assessments.

4.1 Two Dimensional Evaluation Measure

There are two dimensions of relevant evaluation measures, *exhaustivity* and *specificity* in the INEX 2003 relevance assessments. Exhaustivity describes the extent to which the

Table 2: Statistical analysis of XML portions

element	# of XML portions	# of tokens		
		average	maximum	minimum
book	3,612,202	28,897	64,181	6,341
journal	6,314,623	7,342	14,903	3,982
article	11,801,575	974	4,727	29
bdy	9,271,423	765	3,943	11
index	72,993	623	1,593	230
bm	3,125,254	310	2,863	2
dialog	41,317	212	906	19
sec	14,078,415	201	2,613	1
bib	1,662,190	194	1,959	8
bibl	1,662,640	194	1,959	8
app	812,923	138	1,353	2
ss1	7,854,413	127	2,109	1
ss2	1,509,337	92	1,261	1
ss3	11,642	91	325	9
fm	797,123	65	289	9
tgroup	363,102	62	401	2
proof	229,144	60	801	5
vt	1,021,500	55	235	2
dl	18,670	52	745	5
edintro	28,923	50	272	4

Table 3: Average precision of our system based on revised INEX 2003 relevance assessments.

n	strict	generalized
0	0.0564	0.0630
20	0.0666	0.0697
50	0.0689	0.0667
100	0.0777	0.0774
150	0.0866	0.0731
200	0.0769	0.0695
250	0.0611	0.0645
300	0.0253	0.0217

XML portion discussed the topic of request: (0: not exhaustive, 1: marginally exhaustive, 2: fairly exhaustive, 3: highly exhaustive). On the other hand, specificity describes the extent to which the XML portion focuses on the topic of request: (0: not specific, 1: marginally specific, 2: fairly specific, 3: highly specific).

Analyzing the relevance assessments, we examined XML portions which are not only highly exhaustive but also highly specific to each topic, namely $(exhaustivity(E), specificity(S))$ equals to $(3, 3)$. Table 4 is an example (topic #125) of our examinations. As we see in Figure 4, we found that there are some nested relationships among the XML portions. At this time, we did not understand how we could interpret the optimal XML portions. We think that the optimal XML portions depend on the purpose of XML portion retrieval systems; therefore there are a lot of interpretations related to the optimal XML portions. However, criterion for the optimal XML portions, such as the maximal or the minimal XML portions, should be defined.

4.2 Two Types of CO Topics

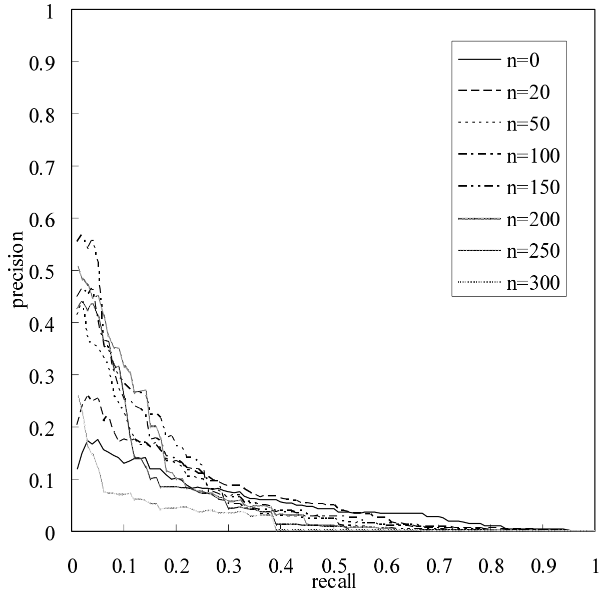


Figure 4: Evaluation of our system based on revised INEX 2003 relevance assessments (strict).

As we described in Section 3.1, the size and the number of answer XML portions related to each CO topic are vary (see Figure 3). Therefore, we notice that there may be two types of retrieval purpose in XML portion retrieval. In short, we think that CO topics of the relevance assessments consist of the topics for searching specific XML portions (SCO) and for searching aggregated XML portions (ACO).

- SCO
  When a topic has many small-size answer XML portions, we think these XML portions are specific; thus they will be semantically consolidated. Table 5 shows the topics of the INEX 2003 relevance assessments that have less than 500 tokens as the average number of tokens of answer XML portions whose (E, S) equal to $(3, 3)$. As for these topics, we think that required XML portions are more specific because keywords of the topics consist of some proper nouns, such as "Charles Babbage," "XML" and "Markv." Moreover, the average number of tokens of answer XML portions assessed as $(E, S) = (3, 1)$ or $(3, 2)$ is larger than assessed as $(E, S) = (3, 3)$.
- ACO
  When the topic has many large-size answer XML portions, we think XML portions whose root node is `article` are suitable for XML portion retrieval. Table 6 shows the topics of the INEX 2003 relevance assessments that have more than 500 tokens as the average number of tokens of answer XML portions whose (E, S) equal to $(3, 3)$. We think that these topics are exhaustive because there is no specific keyword, and we expect that the answer XML portion should cover information on the contents of the topic. As a result, the XML portion which was assessed as $(E, S) = (3, 3)$ became aggregated XML portions with comparatively

Table 4: XML portions evaluated $(E, S) = (3, 3)$ of topic #125

file	path	# of tokens
co/1999/r1057	/article[1]	1128
co/1999/r1057	/article[1]/bdy[1]	863
co/1999/r1057	/article[1]/bdy[1]/sec[3]	215
co/1999/r1057	/article[1]/bdy[1]/sec[3]/fig[1]	37
co/1999/r1057	/article[1]/bdy[1]/sec[3]/fig[1]/art[1]	11
co/1999/r1057	/article[1]/bdy[1]/sec[3]/fig[1]/fgc[1]	24
co/1999/r1057	/article[1]/bdy[1]/sec[3]/p[1]	63
co/1999/r1057	/article[1]/bdy[1]/sec[3]/p[2]	49
co/1999/r1057	/article[1]/bdy[1]/sec[3]/p[3]	32
co/1999/r1057	/article[1]/bdy[1]/sec[3]/p[4]	33
co/1999/r1057	/article[1]/bdy[1]/sec[3]/p[5]	82
co/1999/r1057	/article[1]/bdy[1]/sec[5]	343
co/1999/r1057	/article[1]/bdy[1]/sec[6]	308
co/1999/r1057	/article[1]/bm[1]/app[1]/p[1]	65
co/1999/r1057	/article[1]/bm[1]/app[1]/p[2]	68
co/1999/r1057	/article[1]/bm[1]/app[3]/p[1]	34
co/1999/r1057	/article[1]/bm[1]/app[3]/p[2]	54
co/1999/r1057	/article[1]/bm[1]/app[3]/p[3]	25

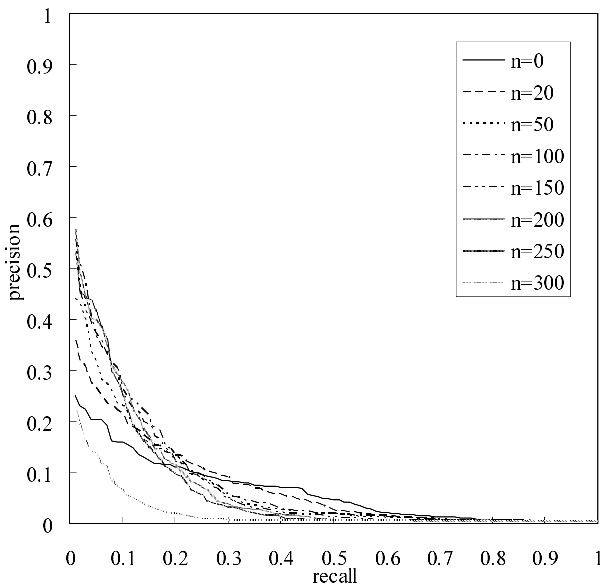


Figure 5: Evaluation of our system based on revised INEX 2003 relevance assessments (generalized).

large granularity. Moreover, the average number of tokens of answer XML portions assessed as $(E, S) = (3, 1)$, $(3, 2)$, or $(2, 3)$ is almost smaller than assessed as $(E, S) = (3, 3)$.

4.3 Our classification of INEX topics

We think that it is important to refine the INEX test collection year by year; thus excellent topics should be selected from the INEX 2002/2003 relevance assessments, and should be reused in INEX 2004. We attempt to classify the INEX topics based on the average number of tokens of answer XML portions.

Figure 6 shows analyses of CO topics of the INEX 2002 relevance assessments with the same way as Figure 3. In INEX 2002 relevance assessments, there are two dimensions of relevant evaluation measures, *relevance* and *coverage*; thus we translated $(relevance, coverage) = (3, E)$ into $(E, S) = (3, 3)$. Compared with these two figures, the INEX 2002 relevance assessments are similar to the INEX 2003 ones. Therefore, we will be able to classify CO topics into the following three categories.

1. The topics inappropriate for test collection
   The topic which does not have answer XML portions at all or has a few answer XML portions is inadequate. To evaluate a performance of the retrieval system, the test collection should guarantee to have certain degree of the number of the answer XML portions in order to emerge into the search results. Therefore, the topics (whose topic IDs are #31, #33, #49, #91, #92, #100, #102, #103, #107, #109, #110, #115, #117, #119, #121 and #124) would be inappropriate for the INEX test collection.

2. The topics for searching spcific XML portions
   We think the topics (whose topic IDs are #32, #26, #23, #38, #41, #42, #44, #46, #48, #51, #60, #93, #94, #97, #98, #99, #101, #104, #108, #112, #113, #116, #123, #125 and #126) are suitable for evaluating XML portion retrieval systems which tend to regard large-size XML portions as retrieval results.

3. The topics for searching aggregated XML portions
   We think that the topics (whose topic IDs are #34, #39, #40, #43, #45, #47, #52, #53, #58, #95, #96 and #111) will be suitable for evaluating XML portion retrieval systems which tend to regard small-size XML portions as retrieval results.

These issues are our own opinions. In the INEX workshop, we hope to discuss these issues in order to construct a valid test collection for XML portion retrieval.

Table 5: SCO topics of the INEX 2003 relevance assessments.

topic ID	title	# of tokens (average) (E, S) (3, 3)	(3,2), (3, 1)	(2, 3)
93	"Charles Babbage" -institute -inst	186	3,377	62
94	"hyperlink analysis" +"topic distillation"	232	83	333
97	Converting Fortran source code	186	753	27
98	"Information Exchange" +XML "Information Integration"	383	0	347
99	perl features	69	314	18
101	+"t test" +information	228	364	222
104	Toy Story	114	735	0
108	ontology ontologies overview "how to" practical example	466	872	367
112	+"Cascading Style Sheets" -"Content Scrambling System"	228	332	61
113	"Markov models" "user behaviour"	438	1,010	90
116	"computer assisted art" "computer generated art"	330	702	207
123	multidimensional index "nearest neighbour search"	245	546	48
125	+wearable ubiquitous mobile computing devices	154	249	47
126	Open standards for digital video in distance learning	288	710	455

Table 6: ACO topics of the INEX 2003 relevance assessments.

topic ID	title	# of tokens (average) (E, S) (3, 3)	(3,2), (3, 1)	(2, 3)
95	+face recognition approach	940	593	486
96	+"software cost estimation"	885	1,174	537
107	"artificial intelligence" AI practical application industry "real world"	1,487	0	633
110	"stream delivery" "stream synchronization" audio video streaming applications	811	669	162
111	"natural language processing" -"programming language" -"modeling language" +"human language"	806	474	253

5. CONCLUSION

In this paper, we analyzed the INEX 2003 relevance assessments based on statistics of their answer XML portions of CO topics, and reported some controversial points of the relevance assessments for our keyword-based XML portion retrieval system. From the viewpoint of statistics, we found that the CO topics unsuitable for our purpose of XML portion retrieval caused low retrieval accuracy of our system; thus we should fix the relevance assessments for evaluating our system.

However, we think that fundamental problem has been remained in the relevance assessments. In other words, we have to clarify what XML portion retrieval is. As we described in Section 4, we believe that retrieval purpose of keyword-based XML portion retrieval is classified into two types such as searching specific XML portions and aggregated ones. The retrieval purposes of these approaches are different; thus we think that the relevance assessments for each approach are required for evaluation. Therefore it is necessary for the INEX 2004 relevance assessments to define the SCO and ACO topics. Moreover, the INEX 2002 relevance assessments were not utilized for evaluation in INEX 2003. We think that excellent topics of the relevance assessments should be used for evaluation in every year, and it helps us to reduce labor of INEX participants. Consequently, we should define the baseline of excellent topics, and should adopt the topics of INEX 2002/2003 relevance assessments which meet the baseline as topics of the INEX 2004 ones.

6. ACKNOWLEDGMENTS

This work is partly supported by the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan, under grants #14780130, #14780325 and #15017243, and also supported by Core Research for Evolutional Science and Technology (CREST) Program "Advanced Media Technology for Everyday Living" of Japan Science and Technology Agency (JST), Japan.

7. REFERENCES

[1] S. Agrawal, S. Chaudhuriand, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proc. of the 18th International Conference on Data Engineering*, pages 5–16. IEEE CS Press, Feb./Mar. 2002.

[2] S. Amer-Yahia and P. Case. XQuery and XPath Full-Text Use Cases. `http://www.w3.org/TR/xmlquery-full-text-use-cases/`, Feb. 2003. W3C Working Draft 14 February 2003.

[3] H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors. *Intelligent Search on XML Data, Applications, Languages, Models, Implementations, and Benchmarks*, volume 2818 of *LNCS*. Springer-Verlag, Sep. 2003.

[4] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J. Siméon. XQuery: A Query Language for XML. `http://www.w3.org/TR/xquery`, Nov. 2002. W3C Working Draft 15 November 2002.

[5] T. Bray, J. Paoli, C. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0

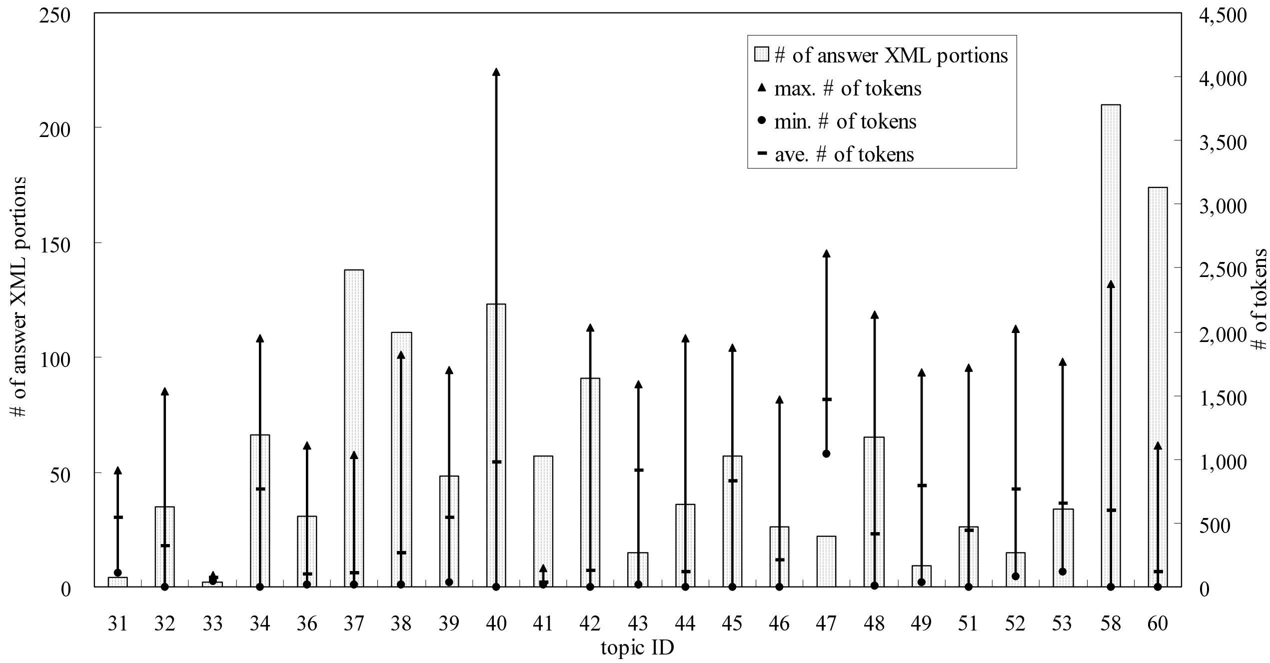


Figure 6: Analyses of the INEX 2002 relevance assessments.

(Second Edition). `http://www.w3.org/TR/REC-xml`, Oct. 2000. W3C Recommendation 6 October 2000.

[6] S. Buxton and M. Rys. XQuery and XPath Full-Text Requirements. `http://www.w3.org/TR/xquery-full-text-requirements/`, May 2003. W3C Working Draft 02 May 2003.

[7] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/xpath`, Nov. 1999. W3C Recommendation 16 November 1999.

[8] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In *Proc. of 29th International Conference on Very Large Data Bases*, pages 45–56. Morgan Kaufmann, Sep. 2003.

[9] A. de Vries, D. Hiemstra, G. Kazai, J. Kamps, J. Thom, K. Hatano, N. Fuhr, N. Gövert, P. Ogilvie, T. Röllenke, and Y. Mass. Evaluation Metrics Workgroup's Report. `http://qmir.dcs.qmul.ac.uk/inex/Slides/wg_metrics.pdf`, Dec. 2002.

[10] N. Gövert, N. Fuhr, M. Abolhassani, and K. Großjohann. Content-Oriented XML Retrieval with HyREX. In *Proc. of the First Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 26–32. ERCIM, Mar. 2003.

[11] N. Gövert, G. Kazai, N. Fuhr, and M. Lalmas. Evaluating the Effectiveness of Content-oriented XML Retrieval. Technical report, University of Dortmund, Computer Science 6, Sep. 2003.

[12] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *Proc. of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 16–27. ACM Press, June 2003.

[13] K. Hatano, H. Kinutani, M. Watanabe, M. Yoshikawa, and S. Uemura. Determining the Unit of Retrieval Results for XML Documents. In *Proc. of the First Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 57–64. ERCIM, Mar. 2003.

[14] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword Proximity Search on XML Graphs. In *Proc. of the 19th International Conference on Data Engineering*, pages 367–378. IEEE CS Press, Mar. 2003.

[15] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. XML Retrieval: What to Retrieve? In *Proc. of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, pages 409–410. ACM Press, Jul./Aug. 2003.

[16] M. Kaszkiel and J. Zobel. Passage Retrieval Revisited. In *Proc. of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 178–185. ACM Press, July 1997.

[17] J. List and A. de Vries. CWI at INEX2002. In *Proc. of the First Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 133–140. ERCIM, Mar. 2003.

[18] G. Navarro and R. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. *ACM Transactions on Information Systems*, 15(4):400–435, Oct. 1997.

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 '$\sim$' 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 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 Fle<u>x</u>ible <u>X</u>ML Search <u>L</u>anguage 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

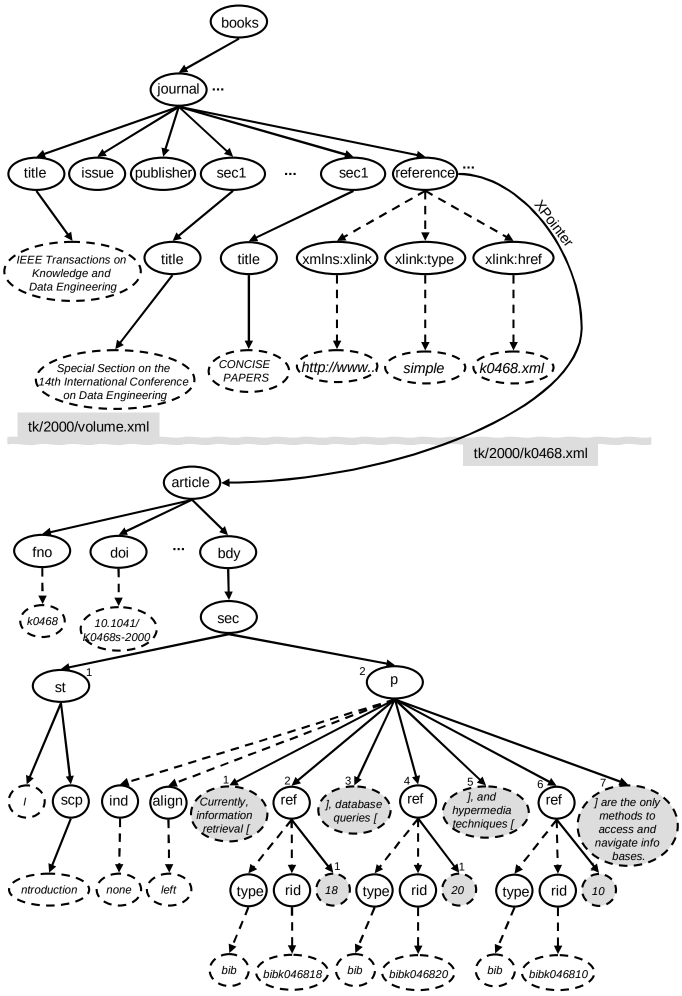


Figure 1: XML data graph

`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 '$\sim$' 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.

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 freqency–based correlations of concepts using large web crawls. In our approach, we compute the similariey of two concepts using correlation coefficients from statistics, e.g. the Dice or Overlap coefficient [14]. Figure 2 shows an excerpt of an example ontology graph around the first sense for the word "star".

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:

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

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

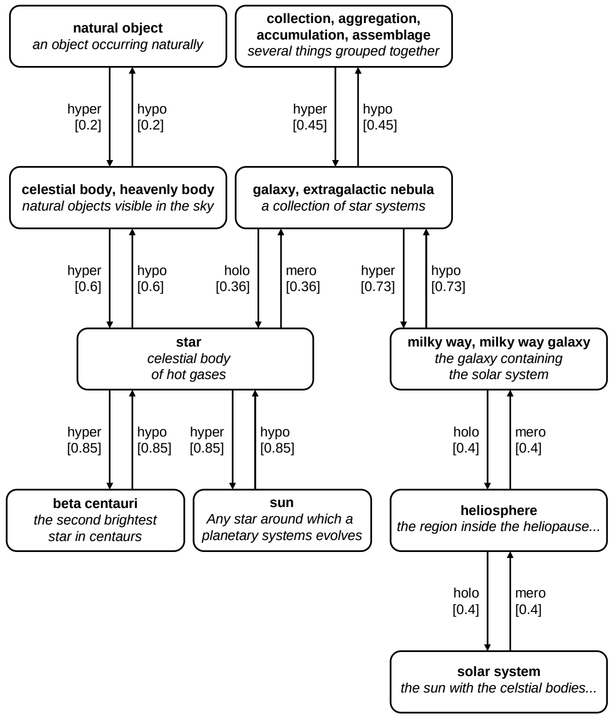


Figure 2: Excerpt of an ontology graph O with labeled edges

the maximal similarity along any path between u and v:

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

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. 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.

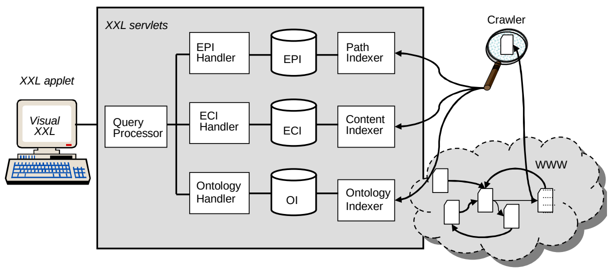


Figure 3: Architecture of the XXL search engine

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.

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.

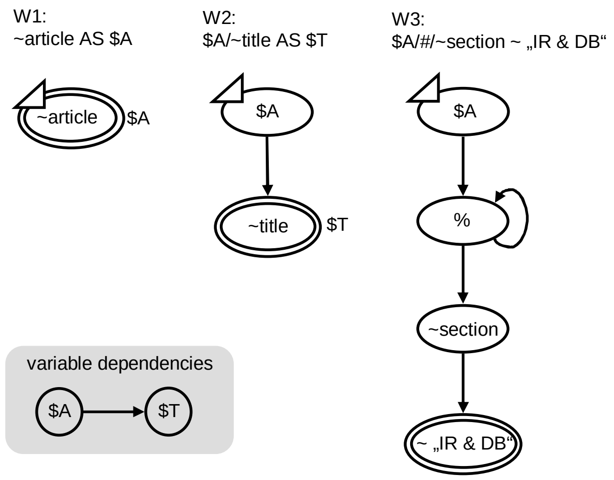


Figure 4: XXL search graphs for each subquery of the given XXL query

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:

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.

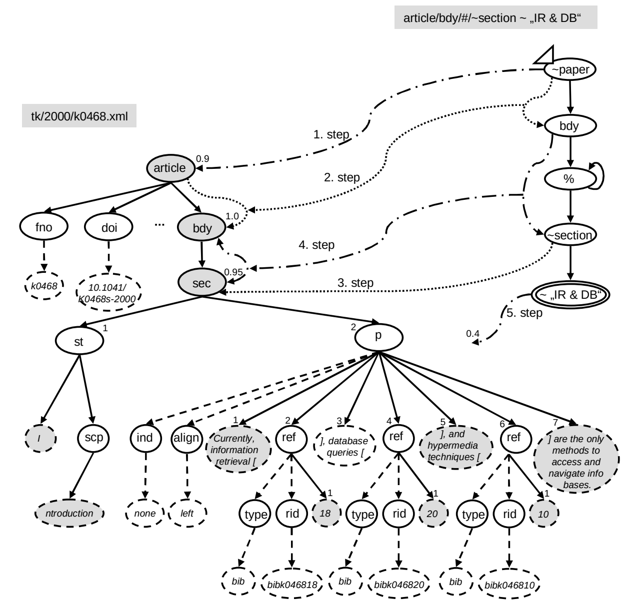


Figure 5: Evaluation of a XXL search graph in top–down manner

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 INEX

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	12.233
names	219
nodes	12.061.347
edges	12.049.114
links	408.085
contents	7.970.615
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 benchmark

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 query 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–query 98:

```
<?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 ex-
    change (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 struc-
    tures of participating data sources are diffe-
    rent although they might use the same ontolo-
    gies about the same content.
  </narrative>
  <keywords>
    information exchange, XML, information inte-
    gration, heterogeneous data sources
  </keywords>
</inex_topic>
```

To automatically transform a CO query into an XXL query we consider only the keywords 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 . For the example CO query this process yields the following XXL query:

```
SELECT *
FROM INDEX
WHERE article/# ~  " information exchange
                   & XML
                   & information extraction "
```

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.

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 synonyms:

```
SELECT *
FROM INDEX
WHERE article/# ~
   " (information exchange | data exchange |
      object exchange | OEM)
   & (XML | semistructured data)
   & (information extraction | data integration |
      database integration | Mediator) "
```

Each potentially relevant component of a journal article is assessed by a human who assigns an *exhaustiveness* 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, the INEX 2003 benchmark applies a metric based on the traditional recall/precision metrics. 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.

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

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 before.

We now consider the quality of the top 10 hits of some of the runs we made for the example query.

For the run with OI disabled, we obtain the following result (in the INEX result format):

```
<inex-submission participant-id="12"
  run-id="OntoA" task="CO"
  query="automatic" topic-part="T">
<description>
  scoring is based on Oracle's tf/idf
</description>
<!-- #Results: 8
     time:     0min 6sec 736msec -->
<topic topic-id="98">
  <result>
    <file>ic/2001/w3032</file>
    <path>article[1]/</path>
    <rank>1</rank>
    <rsv>0.08</rsv>
  </result>
  <result>
    <file>ic/2001/w3021</file>
    <path>article[1]/</path>
    <rank>1</rank>
    <rsv>0.08</rsv>
  </result>
...
</topic>
</inex-submission>
\vspace{-2mm}
```

The following table shows the quality of the top 10 results for the example query:

Run	#Results	1.0	0.75	0.5	0.25	0
1. without onto	8	0	4	1	2	1
2. with onto	40	1	0	9	0	0

Each run needs only some seconds for the complete evaluation and result construction. It is evident that using the ontology returns more results that have a higher quality.

However, if we carefully look at the query, it turns out that a reformulation like the following could return better results:

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

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 result includes some non-relevant results:

Run	#Results	1.0	0.75	0.5	0.25	0
3. without onto	1,225	0	4	4	2	0
4. with onto	3,541	2	3	2	0	3

For evaluating the 3rd and the 4th run the XXL Search Engine needs about 10 minutes for evaluation and result construction.

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.

Applying the IRStream Retrieval Engine to INEX 2003

Andreas Henrich, Günter Robbert
University of Bayreuth
D-95440 Bayreuth, Germany
{andreas.henrich|guenter.robbert}@uni-bayreuth.de

Volker Lüdecke
University of Bamberg
D-96045 Bamberg, Germany
volker.luedecke@wiai.uni-bamberg.de

ABSTRACT

Last year in the context of the INEX evaluation initiative we could show that our retrieval system IRStream is successfully applicable as retrieval engine for XML-documents. Nevertheless, we have to point out that IRStream can be further optimized in many directions.

In the present paper we show, how IRStream was extended and improved for its application for INEX 2003 in order to achieve better retrieval results. Furthermore we point out some first retrieval results, which demonstrate the impact of the improvements of IRStream concerning the quality of the retrieval results.

1. MOTIVATION

Last year, as a participating organization at the INEX evaluation initiative [11], we applied IRStream to the collection of XML documents provided by INEX. Hereby, we investigated the usability of IRStream for structured text documents. By the application of IRStream als retrieval system for XML-documents, we have recognized, that IRStream can be further improved and optimized in many directions.

As the main two drawbacks of IRStream we have identified the absence of a component for the automatic generation of queries based on topics and the problem, that IRStream sometimes delivered wrong granules as the result of a query. Therefore we decided to improve and extend IRStream in order to avoid the drawbacks mentioned above.

IRStream in this respect is intended as a powerful framework to search for components of arbitrary granularity – ranging from single media objects to complete documents. IRStream combines traditional text retrieval techniques with content-based retrieval for other media types and fact retrieval on meta data. In contrast to other retrieval services which permit set-oriented or navigation-oriented access to the documents, we argue for a *stream-oriented* approach. In the following, we shortly describe the significant features of this approach and point out the system architecture of IRStream. Furthermore, we present the application of an extended and improved version of our IRStream retrieval engine as a retrieval system for XML documents in the context of INEX 2003 [4].

The rest of the paper is organized as follows: In section 2 we will give a short overview about the ideas and main components of IRStream. The concrete architecture of our IRStream implementation is presented in section 3. Section 4 shows how we improved our retrieval system IRStream in order to use it as a retrieval engine for XML documents in the context of INEX 2003. Afterwards in section 5 we present some first experimental results concerning the improved version of IRStream. Finally, section 6 concludes the paper.

2. STREAM-ORIENTED QUERY PROCESSING

"Stream-oriented" means that the entire query evaluation process is based on components producing streams one object after the other. First, there are components creating streams given a base set of objects and a ranking criterion. We call these components *rankers*. Other components consume one or more input streams and produce one (or more) output stream(s). *Combiners*, *transferers* and *filters* are different types of such components.

2.1 Rankers

The starting point for the stream-oriented query evaluation process are streams generated for a set of objects based on a given ranking criterion. For example, text objects can be ranked according to their content similarity compared to a given query text and images can be ranked with respect to their color or texture similarity compared to a given sample image.

Such "initial" streams can be efficiently implemented by access structures such as the M-tree, the X-tree, the LSD^h-tree, or by approaches based on inverted files. All these access structures can perform the similarity search in the following way: (1) the similarity search is initialized and (2) the objects are taken from the access structure by means of some type of "getNext" method. Hence, the produced streams can be efficiently consumed one element after the other.

2.2 Combiners

Components of this type combine multiple streams providing the same objects ranked with respect to different ranking criteria. Images are an example for media types, for which no single comprehensive similarity criterion exists. Instead, different criteria addressing color, texture and also shape similarity are applicable. Hence, components are needed which merge multiple streams representing different rankings over the same base set of objects into a combined ranking.

Since each element of each input stream is associated with some type of retrieval status value (RSV), a weighted average over the retrieval status values in the input streams can be used to derive the overall ranking [3]. Other approaches are based on the ranks of the objects with respect to the single criteria [12, 7]. To calculate such a combined ranking efficient algorithms, such as Fagin's algorithm [1, 2], Nosferatu [14], Quick Combine [5] and J^* [13] can be deployed.

2.3 Transferers

With structured documents, ranking criteria are sometimes not defined for the required objects themselves but for their components or other related objects. An example arises when searching for images where the text in the "vicinity" (for example in the same section) should be similar to a given sample text. In such situations the ranking defined for the related objects has to be transferred to the desired result objects.

More precisely said, we are concerned with a query which requires a ranking for objects of some desired object type ot_d (image for example). However, the ranking is not defined for the objects of type ot_d, but for related objects of type ot_r (text for example).

We assume that the relationship between these objects is well-defined and can be traversed in both directions. This means that we can determine the concerned object or objects of type ot_d for an object of type ot_r and that we can determine the related objects of type ot_r for an object of type ot_d. The concrete characteristics of these traversal operations depend on the database or object store used to maintain the documents. In objectrelational databases join indexes and index structures for nested tables are used to speed up the traversal of such relationships. For a further improvement additional path index structures can be maintained on top of the ORDBMS (cf. section 3).

Furthermore, we assume there is an input stream yielding a ranking for the objects of type ot_r. For example, this stream can be the output of a ranker or combiner.

To perform the actual transfer of the ranking we make use of the fact that each object of type ot_r is associated with some type of retrieval status value (RSV_r) determining the ranking of these objects. As a consequence, we can transfer the ranking to the objects of type ot_d based on these retrieval status values. For example, we can associate the maximum retrieval status value of a related object of type ot_r with each object of type ot_d. Another possibility would be to use the average retrieval status value over all associated objects of type ot_r. In [10] you will find a detailed description of an algorithm called "RSV-Transfer", which is used by IRStream to perform the transfer of rankings between different object types.

2.4 Filters

Of course, it must be possible to define filter conditions for all types of objects. With our stream-oriented approach this means that filter components are needed. These filter components are initialized with an input stream and a filter condition. Then only those objects from the input stream which fulfill the given filter condition are passed to the output stream.

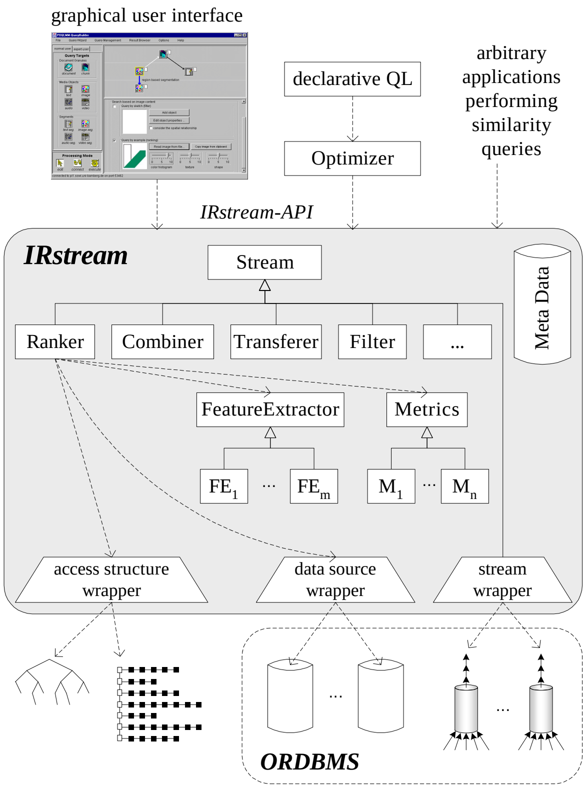


Figure 1: Architecture of the IRStream system

3. THE IRSTREAM ARCHITECTURE

The architecture of our IRStream system is based on the idea that the data is maintained in external data sources. In our implementation, an ORDBMS is used for this purpose. The stream-oriented retrieval engine is implemented in Java on top of this data source and provides an API to facilitate the realization of similarity based retrieval services. Figure 1 depicts this architecture.

The core IRStream system — shaded grey in figure 1 — comprises four main parts: (1) Implementations for rankers, combiners, transferers, and filters. (2) Implementations of various methods for the extraction of feature values as well as corresponding similarity measures. (3) A component maintaining meta data for the IRStream system itself and applications using IRStream. (4) Wrappers needed to integrate external data sources, access structures and stream implementations.

Feature Extractors and Similarity Measures

A feature extractor receives an object of a given type and extracts a feature value for this object. The similarity measures are methods which receive two feature representations — usually one representing the query object and an object from the database. The result of such a similarity measure is a retrieval status value.

Ranker, Combiner, Transferer, Filter,...

All these components are subclasses of the class "Stream". The interface of these classes mainly consists of a specific constructor and a `getNext` method.

For example, the constructor of a *ranker* receives a specification of the data source, a feature extractor, a similarity measure and a query object. Then the constructor inspects the meta data to see if there is an access structure for this data source, this feature extractor, and this similarity measure. In this case, the access structure is employed to speed up the ranking. Otherwise, a table scan with a subsequent sorting is performed.

For the construction of a *combiner* two or more incoming streams with corresponding weights have to be defined. Here it is important to note that combiners such as Fagin's algorithm or Quick Combine rely on the assumption that random access is supported for the objects in the input streams. The reason for this requirement is simple. When these algorithms receive an object on one input stream, they want to calculate the mixed retrieval status value of this object immediately. To this end, they perform random accesses on the other input streams. Unfortunately, some input streams are not capable of such random access options, or a random access would require an unreasonable high effort. In these cases, other combine algorithms — such as Nosferatu or J^* — have to be applied.

For the construction of a *transferer*, an incoming stream, a path expression and a transfer semantics have to be defined. In our implementation, references and scoped references provided by the underlying ORDBMS are used to define the path expressions.

To construct a *filter*, an incoming stream and a filter predicate have to be defined.

Meta Data

This component maintains data about the available feature extractors, similarity measures, access structures, and so forth. On the one hand, this meta data is needed for the IRstream system itself in order to decide if there is a suitable access structure for example. On the other hand, the meta data is also available via the IRstream-API for applications.

Wrapper

IRstream allows for the extension of the retrieval service in various directions by the use of wrappers and interfaces: *Data source wrappers* are needed to attach systems maintaining the objects themselves to our retrieval system. At present, objectrelational databases can be attached via JDBC. Whereas *access structure wrappers* can be used to deploy access structures originally not written for our system. For example, we incorporated an LSD$^{\mathrm{h}}$-tree written in C++ via a corresponding wrapper. In contrast, the *stream wrapper interface* is used to incorporate external sources for streams into our system. It can be used to incorporate external stream producers. At present, the text module of the underlying ORDBMS is integrated via a stream wrapper.

On top of the IRStream API various types of applications can be realized. An example is a graphical user interface where the user can define the query as a graph of related query objects [8]. Another possibility is to implement a declarative query language on top of the API. At present, we are working on a respective adaptation of our POQL$^{\mathrm{MM}}$ query language [6, 9].

4. EXTENSIONS AND IMPROVEMENTS OF IRSTREAM FOR INEX2003

This section describes the main improvements applied to the IRStream retrieval system in the context of INEX 2003. For that, every retrieval system had to be able to perform an automatic query generation from topic data. While a topic is interpreted as a representation of an information desire, a query in this context is an internal representation for the system's retrieval process. Thus, the first extension of IRStream was to integrate a query generation step into this retrieval process. An evaluation of last year's results shows that one of main problems of IRStream02 was the determination of a fitting granule of retrieval results for CO-topics, and furthermore an automatic processing of structural constraints of CAS-topics, as well as automatically generating multiple results from one document (e.g. a list of authors). To solve these problems, the retrieval process of the system was completely redesigned, which is described in this section.

To determine fitting granules for retrieval results (and their corresponding identifying paths), a retrieval system has to be able to perform two tasks: First, extract (possibly several) fragments of one document and determine their unique paths (including node indices). In this case a path expression is given as part of the query, which describes a structural constraint for result granules, as is the case with CAS-topics. Second, the system must be able to process queries which do not contain a constraint regarding the result granule (CO-topics). In this case, the decision about the fitting granule is to be made automatically within the retrieval process.

4.1 Automatic query generation

The queries used internally by a retrieval system, generated from the topic data, may influence the quality of retrieval results significantly. In order to compare the results of different retrieval systems or even the result of a retrieval system in various development states, the influence of manual (pre-) processing must be eliminated. Therefore an automatic query generation was added to the IRStream system, which was also a requirement for retrieval systems participating in INEX 2003. For reasons of performance, two different approaches for CO- and CAS-topics were used, although every CO-topic may be converted into CAS-format by interpreting a CO-topic title as `//[about(.,'CO-title')]`. The different retrieval processes for these two topic types will be described later in this section.

The general architecture is the same for both variants. A wrapper-class *Topic* parses a topic file and provides means

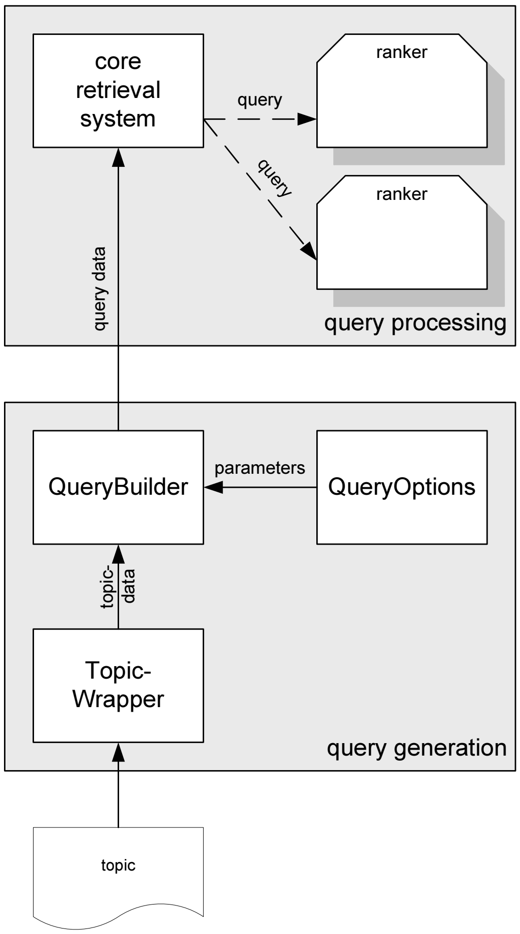


Figure 2: Architecture of query generation

of access in form of a java-API. The system is thereby also prepared for changing topic-formats, which will result in additional sub-classes of this wrapper. The methods provided by *Topic* are used by a *QueryBuilder* component specialized in CO-topics or CAS-topics respectively. This component creates the queries internally used by the *Rankers* of the core-retrieval system. To configure the query generation, a *QueryOptions* class is used, which contains all kinds of parameters used in the generation process. Figure 2 gives an overview of the general architecture and the differentiation between query generation and query processing.

Every query may make use of any of the following three topic parts: the *title*, the *description* and the *keywords*. Within the topic title, terms may further be categorized in must-terms (marked by a `+`), must-not terms (marked by a `-`) and terms not marked at all. For each part or each category of terms, the *QueryOptions* class contains parameters about:

Consideration: Shall these terms be considered for query generation at all?

Weighting: What weight shall be associated to these terms (1-10)?

Stemming: Shall the stemming operator of the underlying ORDBMS be used for these terms?

Connectors: Which connecting operator (OR, AND, ACCUMulate) shall be used to connect terms of this class or between classes of terms?

Compound terms: Which way shall compound terms be treated?

4.2 CAS-topics

A CAS-topic contains structural constraints as well as content information, so that three logic parts of a CAS-topic may be identified: First, a constraint regarding the granule of result elements. Second, content and structure information about the result element itself — i.e. its inner context —, which shall be called *content constraint*. Third, there may be content and structure information about the result element's parent or sibling elements — i.e. the element's environment —, which shall be called *structure constraint*.

The differentiation between content and structure constraint may easily be done by looking at the syntax of a CAS-topic title:

```
[node [filter]]* target-node filter
```

Every filter (which corresponds to constraints) before the target-node belongs to the structure constraint, while the filter given for the target-node contains the content constraint.

The title of a CAS-topic contains a path expression that must be matched by the path of a result element. For the automatic query generation, this path expression is simply the concatenation of all nodes. Normally, there are several elements within a document with matching paths, since the path expression may contain wildcards and does not have to use node indices. Thus, a retrieval system not only has to find relevant documents and determine fitting sub-elements of that element, but it also has to determine relevance scores for each sub-element. Therefore we inserted a new table into the underlying ORDBMS which contains every addressable element of the document collection, i.e. every element that matches the XPath-expression `//*`, which are about 8 mio elements. Each table entry consists of an element with all its sub-elements and their textual content, its unique path expression, and its path expression without indices. To determine the unique path of an element, which is needed for the creation of the submission-file, this data can simply be read from this table. To fulfill the structural constraint of a CAS-topic regarding the result granule, only a selection of those elements is evaluated whose path matches the path expression given in the topic title. Apparently, this approach implies a high degree of redundancy, since the table contains every textual content multiple times. Further developments of IRStream will address this problem, probably by making extended use of the transferer functionality.

The content constraint includes every information that is given about the result element itself. That may be content only, but also constraints concerning the internal structure of an element, like a section having a title about `information retrieval`:

```
/article/bdy/
    sec[about(.,//st,'"information retrieval"')]
```

The crucial factor of this logic part of the topic is that every information needed is within the result element itself and thus may be addressed via the table mentioned above.

The structure constraint includes every information given about the environment of the result element, i.e. its sibling and parent elements. This may include both structure and content information which is not contained in the result element itself and therefore cannot be addressed via the table mentioned above, since the table entries are decoupled from their environment. To fulfill this constraint, a document as a whole has to be evaluated, i.e. it refers to a whole article instead of a result element only.

By looking at an example (topic 77), the retrieval process of IRStream for CAS-topics and the integration of a query generation step into this process will be depicted. The title of topic 77 states:

```
//article[about(.//sec,'"reverse engineering"')]//
    sec[about(.,'legal') OR about(.,'legislation')]
```

The concatenation of all nodes is `//article//sec`, which is the given path expression that all result elements have to fulfill. Therefore only elements with the fitting granule will be ranked in the query process, which is implemented via a corresponding `WHERE`-clause.

The content constraint, referring to the result element itself, is contained in the last filter. It says that the result element has to be about concepts of `legal` or `legislation`. The query generation component successively reads all about-clauses and their connectors. Each about-clause is translated into a corresponding `INPATH`-clause of the ORDBMS, which reads `(terms INPATH path)` and includes any given structural constraints. In this example, `(legal INPATH /sec)` would be the resulting query part. The `INPATH`-clauses, their connectors and the result element's path expression form the main part of the content query, which is applied to the table containing every addressable element.

The structure query on the other hand has to be applied to a table of whole articles, which contain the complete structure information of a document. The query generation is done accordingly, reading each filter successively and connecting the resulting `INPATH`-expressions. The last filter in the topic-title may or may not be part of the structure query. Not including it means that some articles are probably marked relevant that do not contain any elements that satisfy the content constraint. IRStream therefore considers the content query being a part of the structure query.

In order to get a result ranking, these two queries have each to be processed by a ranker-component and then be joined into a final ranking. These two rankers create streams of two different object types — article (structure query) and element (content query) —, which cannot directly be combined by a combiner-component. Therefore a transferer-

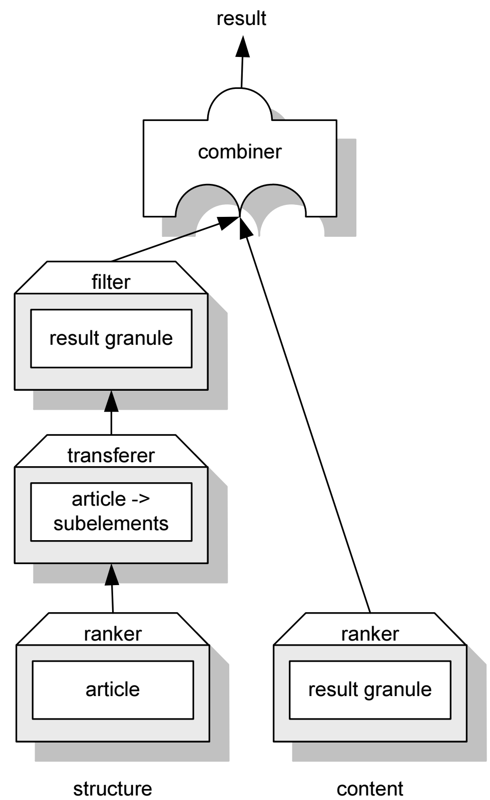


Figure 3: CAS-topic processing

component is needed, which transfers the ranking of an article to all its sub-elements. A special filter-component filters all elements whose path does not fulfill the given path expression. The output of this filter is a stream of elements, and thus a combiner can finally merge the two streams into a result ranking. This procedure is shown in figure 3.

Obviously, this (general) procedure can be optimized, because the transferer creates hundreds of elements that are immediately eliminated by a filter. Therefore the task of ranking, transferring and filtering was integrated into a specialized component `InexRanker`, which relocates the transferring-process into the DBMS. The described three logical steps can thereby be performed by a single SQL-query:

1. ranking an article in reference to the structure query

2. transferring the RSV to all sub-elements, identified via foreign key relationship

3. selection of those elements that fulfill the given path expression

4.3 CO-topics

The special challenge while processing CO-topics is that the retrieval system has to decide autonomously, which granule of the result elements is the most fitting. For INEX 2003, the procedure for handling CO-topics is based on the table mentioned above, which contains every addressable element including all its textual content and that of its sub-elements. A single ranker-component simply creates a ranking of all those elements, and an element's filename and unique path may be read from this table. The aim of this approach was to evaluate whether it is worthwhile to base further optimizations on it, which are obviously possible, since this table contains about eight million elements, every layout-tag (italic, bold etc.) being contained.

For CO-topics, four characteristics can be identified. Based on these, the general applicability of this approach is to be shown:

CO-topics do not contain structural information

The elements in the table used are decoupled from their structural environment and are treated as single documents. No structure information is needed for this query processing.

CO-topics do not contain constraints regarding the granule of result elements

By this procedure, elements of all granules are ranked likewise, so that every granule may be contained in the result ranking. Possible optimizations will be addressed in section 6.

An ideal result element satisfies the information need completely

A retrieval system cannot validate a complete answering of an information need, but this requirement has to be considered in the process of determining relevance scores. Regarding a XML-document as being a tree of elements, that one element obviously fulfills that requirement best, which is superior to all elements which contain relevant information. If several paragraphs are marked relevant, for example, their corresponding section seems to be most fitting element. The calculation of a *score*-value that is done by the underlying OR-DBMS provides an according evaluation, because it is in principle based on absolute term frequencies. Thus, superior elements normally get a relevance score which is equal to or greater than that of their child elements.

An ideal result element is specific about the topic's theme

For INEX 2003, IRStream did not eliminate multiple result elements within a branch of the document tree, the consequences of it with respect to retrieval effectiveness has not yet been evaluated, but it will be addressed in the near future. If several elements of a branch have the same RSV-score, it is obviously the smallest element that conforms best to this requirement. It remains to be seen whether elimination of such duplicates or considering document lengths will improve retrieval effectiveness.

The query generation for CO-topics is similar to that of CAS-topics, but here only one query has to be created, and no structural information has to be included. Terms in the title of CO-Topic may be marked by a `+` (declared as must-terms). The IRStream query generation allows to interpret these markings as strict or vague. A strict interpretation means that only those elements may be relevant that contain all must-terms. Therefore these terms are connected to each other by `AND`-operators, and must-terms and all other terms are each encapsuled by brackets which are also connected by an `AND`-operator. Interpreting these terms vague, other connecting operators may be used, like `ACCUMulate` or `OR`.

5. EVALUATION OF THE NEW IRSTREAM ENGINE AT INEX2003

With the runs submitted to INEX 2003, two things were to be looked at: First, we wanted to see, whether our interpretation of CAS-topics and thus the differentiation between content and structure constraints would lead to good results compared to those of the other participating retrieval systems. Second, we wanted to get an estimation of how applicable our approach for processing CO-topics is.

Figures 4 and 5 show the recall/precision graphs for IRStream's CAS-run — with strict and generalized quantization — in comparison to all officially submitted retrieval runs. Rank 12 of 38 for strict quantization and rank 10 of 38 for generalized quantization seem promising that the chosen query architecture forms a solid basis for further efforts.

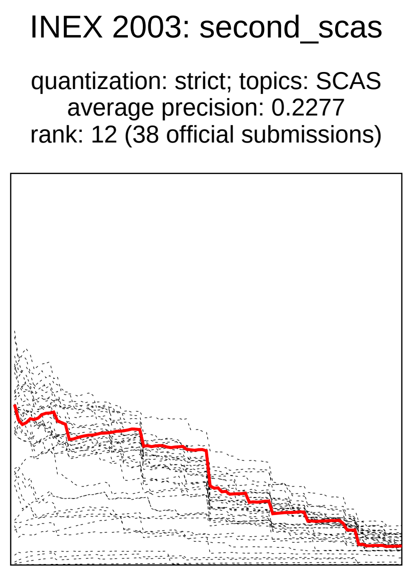


Figure 4: summary CAS strict

The recall/precision graphs for IRStream's CO-run are shown in figures 6 and 7. Rank 10 of 56 for strict and rank 7 of 56 submissions for generalized quantization indicate that further efforts to optimize our approach seem to be worthwhile.

In order to compare the results of IRStream02 and IRStream03 — and thus to evaluate the effect of the system changes — we used the new system to create a retrieval run on the topics of INEX 2002. Since the topic syntax for CAS-topics has changed, only those topics were processed in this run which could be converted to the new syntax. Topics without explicitly stating a target element or those with multiple target elements do not conform to INEX 2003 syntax and thus were omitted.

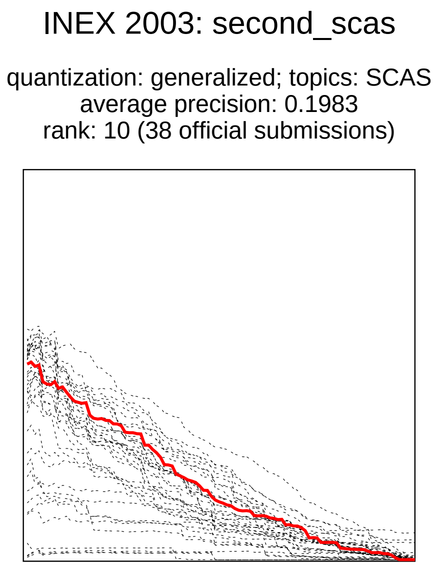


Figure 5: summary CAS generalized

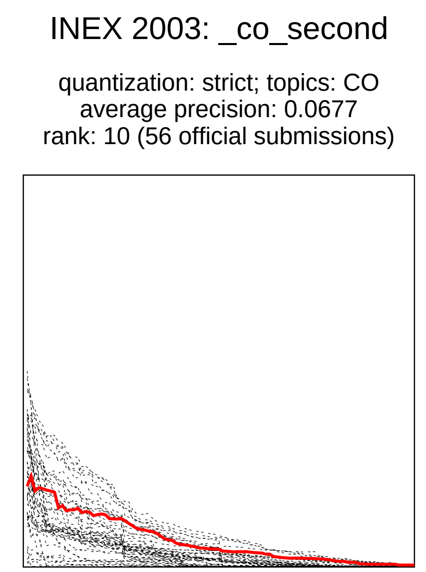


Figure 6: summary CO strict

6. CONCLUSION

In this paper, we have presented an improved version of our retrieval system called IRStream, which was successfully used in the context of INEX 2002. The main idea of IRStream is intended to complement traditional query processing techniques for queries dominated by similarity conditions. The IRStream retrieval engine has been implemented as a prototype in Java on top of an ORDBMS and first experimental results achieved with this prototype are promising. With regard to INEX2003 IRStream was extended and improved in several directions. IRStream now supports automatic query generation as well as the automatic detection of the best fitting result granule for a given query.

In the near future, we will develop a query language for this approach and consider optimization issues regarding

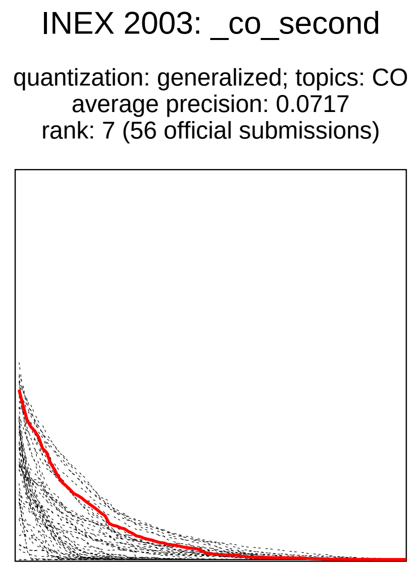


Figure 7: summary CO generalized

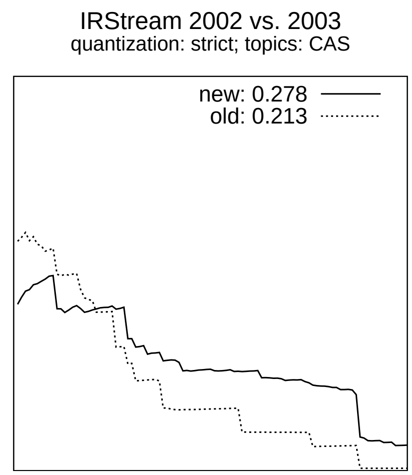


Figure 8: improvement CAS strict

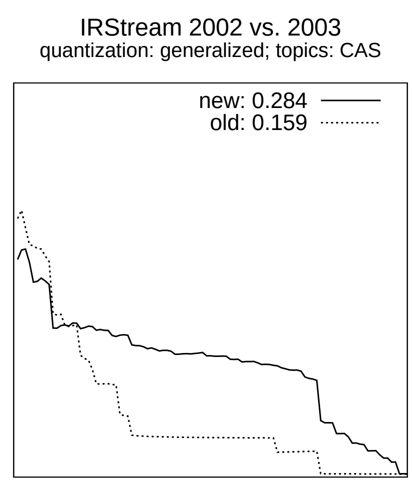


Figure 9: improvement CAS generalized

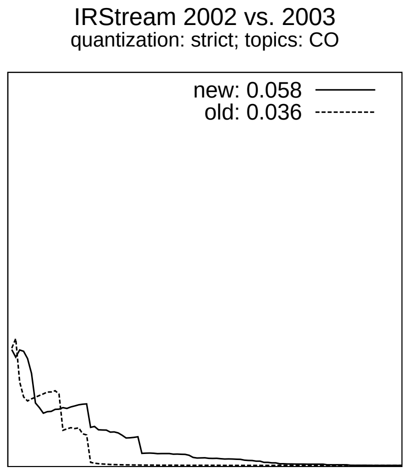


Figure 10: improvement CO strict

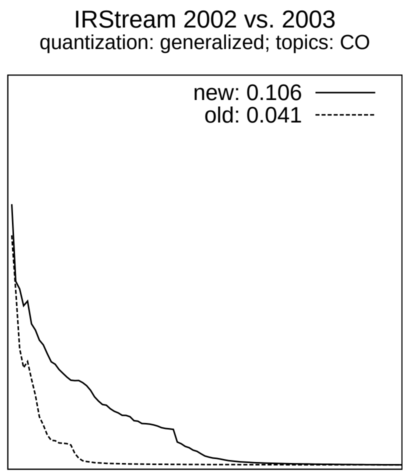


Figure 11: improvement CO generalized

the interaction between the underlying ORDBMS and the IRStream system. Last but not least, IRStream should build a good basis for the integration of further query criteria — like context information or domain specific thesauri — into the query execution in order to improve the precision of the system.

7. REFERENCES

[1] R. Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999.

[2] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. 10th ACM Symposium on Principles of Database Systems: PODS*, pages 102–113, New York, USA, 2001.

[3] R. Fagin and E. L. Wimmers. A formula for incorporating weights into scoring rules. *Theoretical Computer Science*, 239(2):309–338, 2000.

[4] N. Fuhr and M. Lalmas. *Initiative for the Evaluation of XML retrieval (INEX)*. Online available: url: http://inex.is.informatik.uni-duisburg.de:2003, 2002.

[5] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *VLDB 2000, Proc. 26th Intl. Conf. on Very Large Data Bases*, pages 419–428, Cairo, Egypt, 2000.

[6] A. Henrich. Document retrieval facilities for repository-based system development environments. In *Proc. 19th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 101–109, Zürich, Switzerland, 1996.

[7] A. Henrich and G. Robbert. Combining multimedia retrieval and text retrieval to search structured documents in digital libraries. In *Proc. 1st DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries*, pages 35–40, Zürich, Switzerland, 2000. ERCIM Workshop Proceedings.

[8] A. Henrich and G. Robbert. An end user retrieval interface for structured multimedia documents. In *Proc. 7th Workshop on Multimedia Information Systems, MIS'01*, pages 71–80, Capri, Italy, 2001.

[9] A. Henrich and G. Robbert. POQL^{MM}: A query language for structured multimedia documents. In *Proc. 1st Intl. Workshop on Multimedia Data and Document Engineering, MDDE'01*, pages 17–26, Lyon, France, 2001.

[10] A. Henrich and G. Robbert. RSV-Transfer: An algorithm for similarity queries on structured documents. In *Proceedings of the 9th International Workshop on Multimedia Information Systems (MIS 2003)*, pages 65–74, Ischia, Italy, May 2003.

[11] G. Kazai, N. Gövert, M. Lalmas, and N. Fuhr. *The INEX evaluation initiative*, pages 279–293. Lecture Notes in Computer Science. Springer, Heidelberg et al., 2003.

[12] J. H. Lee. Analyses of multiple evidence combination. In *Proc. 20th Annual Intl. ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–276, Philadelphia, PA, USA, 1997.

[13] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *Proc. 27th Intl. Conf. on Very Large Data Bases*, pages 281–290, Los Altos, USA, 2001.

[14] U. Pfeifer and S. Pennekamp. Incremental Processing of Vague Queries in Interactive Retrieval Systems. In *Hypertext - Information Retrieval - Multimedia '97: Theorien, Modelle und Implementierungen*, pages 223–235, Dortmund, 1997.

HyREX at INEX 2003

Mohammad Abolhassani, Norbert Fuhr, Saadia Malik
University of Duisburg-Essen, Germany

9th December 2003

1 Introduction

The HyREX (Hypermedia Retrieval Engine for XML) system developed by our group [Fuhr & Großjohann 01, Fuhr & Großjohann 04, Fuhr et al. 02] supports document ranking based on index term weighting, specificity-oriented search for retrieving the most relevant parts of documents, datatypes with vague predicates for dealing with specific types of content and structural vagueness for vague interpretation of structural query conditions. In INEX 2002, HyREX performed very well for content-only (CO) queries, but only poorly for content-and-structure(CAS) queries (although this was due to a bug in the implementation).

In this paper, we describe a new retrieval model for CO queries. For the CAS queries, we investigated several methods for transforming INEX topics into our own query language XIRQL.

2 Content-Only queries

In [Fuhr & Großjohann 01], we proposed the *augmentation* method for processing content-only queries. This method gave very good results in INEX 2002. This approach can be combined with any kind of indexing method (we were using the BM25 formula for this purpose). This year, we were/are trying to adopt a kind of language model, namely the divergence from randomness (DFR) approach developed by Gianni Amati.

2.1 The DFR approach

[Amati & Rijsbergen 02] introduce a framework for deriving probabilistic models of IR. These models are non-parametric models of IR as obtained in the *language model* approach. The term weighting models are derived by measuring the divergence of the actual term distribution from that obtained under a random process.

In this framework, the weighting formula for a term in a document is the product of the following two factors:

1. $Prob_1$ is used for measuring the *information content* of the term in a document, and $(-\log_2 Prob_1)$ gives the corresponding amount of information.
2. $Prob_2$ is used for measuring the *information gain* of the term with respect to its 'elite' set (the set of all documents in which the term occurs). The less the term is expected in a document with respect to its frequency in the elite set (measured by the counter-probability $(1-Prob_2)$, the more the amount of information is gained with this term.

Then the weight of a term in a document is defined as:

$$w = (1-Prob_2)\cdot(-\log_2 Prob_1) = Inf_2 \cdot Inf_1 \quad (1)$$

For computing the two probabilities, the following parameters are used:

N number of documents in the collection,

tf term frequency within the document (since different normalisations are applied to the term frequency, we use tf_1 and tf_2 in the following formulas),

n size of the elite set of the term,
F term frequency in elite set.

Furthermore, let $\lambda = F/N$ in the following.

As probability distribution for estimating $Prob_1$, different probabilistic models are regarded in [Amati & Rijsbergen 02]. In this paper, we use only two of them:

D The approximation of the binomial model with the divergence:

$$Inf_1 = tf_1 \cdot \log_2 \frac{tf_1}{\lambda} + \left(\lambda + \frac{1}{12 tf_1} - tf_1\right) \cdot \log_2 e + 0.5 \log_2(2\pi \cdot tf_1) \quad (2)$$

G The Geometric as limiting form of the Bose-Einstein model:

$$Inf_1 = -log_2 \frac{1}{1+\lambda} - tf_1 \cdot \log_2 \frac{\lambda}{1+\lambda} \quad (3)$$

For the parameter $Inf_2 = (1 - Prob_2)$ (which is also called *first normalisation*), $Prob_2$ is defined as the probability of observing another occurrence of the term in the document, given that we have seen already tf occurrences. For this purpose, Amati regards two approaches:

L Based on Laplace's law of succession, he gets

$$Inf_2 = \frac{1}{tf_2 + 1} \quad (4)$$

B Regarding the ratio of two Bernoulli processes yields

$$Inf_2 = \frac{F+1}{n \cdot (tf_2 + 1)} \quad (5)$$

These parameters do not yet consider the length of the document to be indexed. For the relationship between document length and term frequency, we apply the following formula:

$$\rho(l) = c \cdot l^{\beta} \quad (6)$$

where l is the document length, $\rho(l)$ is the density function of the term frequency in the document, c is a constant and β is a parameter to be chosen.

In order to consider length normalisation, Amati maps the tf frequency onto a normalised frequency tfn computed in the following way: Let $l(d)$ denote the length of document d and avl is the average length of a document in the collection. Then tfn is defined as:

$$tfn = \int_{l(d)}^{l(d)+avl} \rho(l) dl \quad (7)$$

For considering these normalisations, Amati sets $tf_1 = tf_2 = tfn$ in formulas 2–5 and then computes the term weight according to eqn 1.

For retrieval, the query term weight qtf is set to the number of occurrences of the term in the query. Then a linear retrieval function is applied:

$$R(q,d) = \sum_{t \in q} qtf \cdot Inf_2(tf_2) \cdot Inf_1(tf_1) \quad (8)$$

2.2 Applying divergence from randomness to XML documents

2.2.1 Direct application of Amati's model

In Section 2.1, we have described the basic model along with a subset of the weighting functions proposed by Amati. Given that we have two different formulas for computing Inf_1 as well as two different ways for computing Inf_2, we have four basic weighting formulas which we are considering in the following.

In a first round of experiments, we tried to apply Amati's model without major changes. However, whereas Amati's model was defined for a set of atomic documents, CO retrieval is searching for so-called *index nodes*, i.e. XML elements that are meaningful units for being returned as retrieval answer.

As starting point, we assumed that the complete collection consists of the concatenation of all XML documents. When we regard a single index node, we assume that the complete collection consists of documents having the same size as our current node. Let L denote the total length of the collection and $l(d)$ the length of the current node (as above), then we compute the number of hypothetical documents as $N = L/l(d)$.

Table 1 shows the experimental results. The first two result columns show the average precision values for this setting when applying the four different weighting functions. We assume that the poor

Table 1: Results from direct application vs. augmentation approach

document length	Dynamic		Fixed	
	B Norm.	L Norm.	B Norm.	L Norm.
Bernoulli	0.0109	0.0356	0.0640	0.0717
Bose-Einstein	0.0214	0.0338	0.0468	0.0606
Augmentation	0.1120			

Table 2: Results from 2nd normalisation with two basic values for β

	$\beta = 0$		$\beta = -1$	
	B Norm.	L Norm.	B Norm.	L Norm.
Bernoulli	0.0391	0.0586	0.0640	0.0900
Bose-Einstein	0.0376	0.0609	0.0376	0.0651

performance is due to the fact that the weights derived from different document lengths are not comparable.

As an alternative method, we computed the average size of an index node. The two last columns in table 1 show a much better retrieval quality for this case.

In the subsequent experiments, we focused on the second approach. By referring to the average size of an index node we were also able to apply document length normalisation according to Equation 6. Table 2 shows the corresponding results for $\beta = 0$ and $\beta = -1$. The results show that length normalisation with $\beta = -1$ improves retrieval quality in most cases. These results were also in conformance with Amati's findings that $\beta = -1$ gives better results than $\beta = 0$.

Subsequently we tried some other values for β. Table 3 shows the corresponding results for $\beta = -0.75$ and $\beta = -0.80$, with which we got better results.

Overall, using a fixed average document length, and length normalisation, gave better results than those achieved in the first round. However, the resulting retrieval quality was still lower than that of the augmentation approach (see table 1). Thus, in order to arrive at a better retrieval quality, we investigated other ways than straightforward application of Amati's model.

2.2.2 Considering the hierarchical structure of XML documents

In order to consider the hierarchical structure of our documents, we investigated different ways for incorporating structural parameters within the weighting formula. Considering the basic ideas, as described in Section 2.1, the most appropriate way seemed the modification of the Inf_2 parameter, which refers to the 'elite' set. Therefore, we computed Inf_1 as above, by performing document length normalisation with respect to the average size of an index node.

For computing Inf_2, we also applied document length normalisation first, thus yielding a normalised term frequency tfn. Then we investigated several methods for 'normalising' this factor with respect to the hierarchical document structure; we call this process *third normalisation.* For this purpose, we introduced an additional parameter $h(d)$ specifying the height (or level) of an index node relative to the root node (which has $h = 1$).

Using the level information, we first tried several heuristic formulas like $tf_2 = tfn \cdot h(d)^{\alpha}$ and $tf_2 = tfn \cdot h(d)^{-\alpha}$, which, however, did not result in any improvements. Finally, we came up with the following formula:

$$tf_2 = tfn \cdot (h(d)/\alpha) \tag{9}$$

Here α is a constant to be chosen, for which we tried several values. However, the experiments showed that the choice of α is not critical.

Table 4 shows the results for the combination

Table 3: Results from 2nd normalisation with two other values for β

	$\beta = -0.75$		$\beta = -0.80$	
	B Norm.	L Norm.	B Norm.	L Norm.
Bernoulli	0.0799	0.1026	0.0768	0.1005
Bose-Einstein	0.0453	0.0653	0.0448	0.0654

Table 4: Average precision for the Bose-Einstein L Norm combination with various values of α

α	2	4	9	16	20	32	64	96	104	116	128
prec.	0.0726	0.0865	0.0989	0.1059	0.1077	0.1083	0.1089	0.1094	0.1087	0.1081	0.1077

of Bose-Einstein and Laplace normalisation, for which we got significant improvements. This variant also gave better results in Amati's experiments.

2.2.3 INEX 2003 Submissions

Our CO submissions in INEX 2003 include:

- factor 0.5
- factor 0.2
- difra_sequential

The first two submissions use the "augmentation" method (the same as in our 2002 INEX submission) with 0.5 and 0.2 as "augmentation factor", respectively. The third submission is based on the "DFR" method. Here, we chose the best configuration according to our experiments results, i.e. Bose-Einstein and L Normalisation with the parameters $\alpha = 96$ and $\beta = -0.80$.

Table 5 shows the evaluation results of our submissions, based on different metrics, in INEX 2003. The results show that the latter two submissions both performed very well, with the augmentation method still slightly better than the DFR approach.

3 Content-and Structure(CAS) Topics

The query language XIRQL of our retrieval system HyREX is very similar to the INEX CAS topic specification. However, our experience from INEX 2002 has shown that a 'literal' interpretation of the CAS queries does not lead to good retrieval results. Thus, we were looking for 'vague' interpretations of the INEX topics. Since XIRQL has a high expressiveness, we did not want to change the semantics of XIRQL (by introducing vague interpretations of the different language elements). Instead, we focused on the transformation from the INEX topic specification into XIRQL.

XIRQL is an extension of XPath [Clark & DeRose 99] by IR concepts. We assume that XML document elements have specific data types, like e.g. person names, dates, technical measurement values and names of geographic regions). For each data type, there are specific search predicates, most of which are vague (e.g. phonetic similarity of names, approximate matching of dates and closeness of geographic regions). In addition to Boolean connectors, there also is a weighted sum operator for computing the scalar product between query and document term weights.

The general format of a of XIRQL query is
//TE[filter] or
//CE[filter]//TE[filter]
Where TE stands for Target Element and CE stands for Context Element.

In XIRQL, single query conditions can be combined in the following way:

Conjunctions(C) Filter conditions(conditions within [..]) can be combined by the and operator

Disjunctions(D) Filter conditions can be combined by the or operator.

Weighted Sum (WS) and Precedence (Pr) Weighted sum notation can be used to indicate the importance of a query term, e.g.

```
//article[wsum(
```

Table 5: Average precision for our CO submissions in INEX 2003, based on different metrics

Submission	Average Precision					
	inex_eval		inex_eval_ng			
			consider overlap		ignore overlap	
	strict	generalised	strict	generalised	strict	generalised
factor 0.5	0.0703	0.0475	0.1025	0.0623	0.0806	0.0590
factor 0.2	0.1010	0.0702	0.1409	0.0903	0.1219	0.0964
difra_sequential	0.0906	0.0688	0.1354	0.0774	0.1217	0.0920

```
0.7,.//atl//#PCDATA $stem$ "image",
0.3,.//atl//#PCDATA $stem$ "retrieval"
)]
```

Phrases (P) Since HyREX has no specific phrase operator (yet), we represented phrases as conjunctions of the single words, e.g.

```
//article[wsum(
1.0,.//atl//#PCDATA [. $stem$ "image"
$and$ . $stem$ "retrieval"],
1.0,. $stem$ "colour")]
```

3.1 Experimentation

In order to search for better transformations from INEX CAS topics into XIRQL, we performed a number of experiments using the INEX 2002 topics (which we transformed into the 2003 format). For generating our XIRQL queries, we used only titles and keywords of the topics. In the following we briefly characterise the different kinds of transformations investigated. We illustrate each method by showing the resulting XIRQL expression for the following INEX topic:

```
//article[about(.//atl,'"image retrieval"'
) and about(.,'image retrieval colour
shape texture')]
```

3.1.1 SCAS-I

1. Only query title is used.
2. Phrases are represented using conjunctions.
3. Query terms are represented using disjunctions
4. '+' prefixed terms are handled as phrases.

```
//article[(.//atl//#PCDATA[
. $stem$ "image" $and$
. $stem$ "retrieval"]) $and$
(.//#PCDATA[ . $stem$ "image"] $or$
.//#PCDATA[ . $stem$ "retrieval"]
$or$ .//#PCDATA[ . $stem$ "colour"]
$or$ .//#PCDATA[ . $stem$ "shape"]
$or$ .//#PCDATA[ . $stem$ "texture"]
)]
```

3.1.2 SCAS-II

1. Only query title is used.
2. Phrases are represented using conjunctions.
3. Terms are represented using weighted sum notation and assigned weight 1.
4. '+' prefixed terms are assigned higher weights.

```
/article[ wsum(1.0,.//atl//#PCDATA[
. $stem$ "image" $and$
. $stem$ "retrieval"],
 1.0, ... $stem$ "image",
 1.0, ... $stem$ "retrieval",
 1.0, ... $stem$ "colour",
 1.0, ... $stem$ "shape",
 1.0, ... $stem$ "texture")]
```

3.1.3 SCAS-III

1. Only query title is used.
2. Phrases are represented using conjunctions.
3. Terms are represented using weighted sum notation and XPath notations. These two notations are joined with or operator.
4. '+' prefixed terms are assigned higher weight 5 and also represented as phrases.

So this variation is a combination of SACS-I and SCAS-II.

```
//article[(.//atl//#PCDATA[
. $stem$ "image" $and$
. $stem$ "retrieval"])
$and$
```

Table 6: Survey over query variations

	Query part	Notation	Terms	Phrases	+ Prefixed Terms
SCAS-I	T[a]	XPath	D [b]	C [c]	C
SCAS-II	T	WSum	W [d]1.0	C	W 5.0
SCAS-III	T	XPath & WSum	D & W 1.0	C	C & W 5.0
VCAS-I	T & K [e]	XPath	D	C	C
VCAS-II	T	WSum	W 1.0	W 1.0	W 5.0

[a]title
[b]disjunction
[c]conjunction
[d]weight
[e]keywords

Table 7: Experimentation results

	SCAS-I	SCAS-II	SCAS-III	VCAS-I	VCAS-II
IE[f]$-Gen$[g]	.2338	.1215	.1475	.1179	.1077
CE[h]$-Gen$	.1508	.0798	.0916	.0877	.0872
IE-Strict	.2640	.1325	.1724	.1297	.1327
CE-Strict	.1692	.0859	.1045	.0959	.0806

[f]ignore empty
[g]generalised quantisation
[h]consider empty

```
(.//#PCDATA[ . $stem$ "image"] $or$
 .//#PCDATA[ . $stem$ "retrieval"] $or$
 .//#PCDATA[ . $stem$ "colour"] $or$
 .//#PCDATA[ . $stem$ "shape"] $or$
 .//#PCDATA[ . $stem$ "texture"]) $or$
 wsum(1.0,.//atl//#PCDATA $stem$ "image",
  1.0,.//atl//#PCDATA $stem$ "retrieval",
  1.0, ... $stem$ "image",
  1.0, ... $stem$ "retrieval",
  1.0, ... $stem$ "colour",
  1.0, ... $stem$ "shape",
  1.0, ... $stem$ "texture")]
```

3.1.4 VCAS-I

1. Query titles and keywords are used. Keywords are considered in case there are less than 3 query terms in the title.
2. Phrases are represented using conjunctions.
3. Terms are represented using disjunctions
4. '+' prefixed terms are handled as phrases.

```
//article[  (  .//atl//#PCDATA[
. $stem$ "image" $and$
. $stem$ "retrieval"]) $and$
 (.//#PCDATA[ . $stem$ "image"] $or$
  .//#PCDATA[ . $stem$ "retrieval"]
  $or$ .//#PCDATA[ . $stem$ "colour"]
  $or$ .//#PCDATA[ . $stem$ "shape"]
  $or$ .//#PCDATA[ . $stem$ "texture"
  ])]
```

3.1.5 VCAS-II

1. Only query title is used.
2. Phrases are also handled as terms and assigned weight 1.0.
3. Terms are combined by wsum operator.
4. Higher weight (5) is assigned to terms prefixed with '+'.

```
//article[
wsum( 1.0,.//atl//#PCDATA $stem$ "image",
 1.0,.//atl//#PCDATA $stem$ "retrieval",
 1.0, ... $stem$ "image",
 1.0, ... $stem$ "retrieval",
 1.0, ... $stem$ "colour",
 1.0, ... $stem$ "shape",
 1.0, ... $stem$ "texture")
]
```

Table 8: Average precision for our CAS submissions in INEX 2003

Submission	Average Precision	
	inex_eval	
	strict	generalised
SCAS03-I-alias	0.2594	0.2037
SCAS03-II-alias	0.2213	0.1744
SCAS03-III-noalias	0.2034	0.1707

3.1.6 Evaluation

One of the results of the first INEX workshop (INEX 2002) has been the definition of a metric for the evaluation of XML retrieval approaches [Gövert & Kazai 03]. This metric, based on the notion of precision and recall – together with the relevance assessment package – has been used here for the evaluation.

3.1.7 INEX 2003 Submissions & Results

Except topic formats, there are two differences between CAS topic tasks of this year and the last year; this year there is a VCAS subtask and there are aliases defined.

Our CAS submissions in INEX 2003 include:

- SCAS03-I-alias
- SCAS03-II-alias
- SCAS03-III-noalias
- VCAS03-I-alias
- VCAS03-II-alias
- VCAS03-I-noalias

These submissions have been formed using the methods described above, with and without alias option. Some submissions were made with no alias option, due to the reason that our system could not process some of the topics with in the aliased version.

Table 8 shows the evaluation results of our submissions in INEX 2003. The results confirm the outcome of our own experimentation. SCAS03-I-alias is the best of our submitted runs and performed quite well in comparison to other approaches.

4 Conclusions

The results from INEX 2003 show that HyREX yields good retrieval performance both for CO and CAS queries. For the CO queries, we will continue our work on the DFR approach towards a full-fledged language model for XML retrieval. On the CAS side, besides dealing with some weaknesses of the current implementation, we will investigate further methods for 'vague' interpretations of this type of queries.

References

Amati, G.; van Rijsbergen, C. (2002). Probabilistic models of Information Retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems 20(4)*, pages 357–389.

Clark, J.; DeRose, S. (1999). *XML Path Language (XPath) Version 1.0*. Technical report, World Wide Web Consortium. `http://www.w3.org/TR/xpath20/`.

Fuhr, N.; Großjohann, K. (2001). XIRQL: A Query Language for Information Retrieval in XML Documents. In: Croft, W.; Harper, D.; Kraft, D.; Zobel, J. (eds.): *Proceedings of the 24th Annual International Conference on Research and development in Information Retrieval*, pages 172–180. ACM, New York.

Fuhr, N.; Großjohann, K. (2004). XIRQL: An XML Query Language Based on Information Retrieval Concepts. (accepted for publication).

Fuhr, N.; Gövert, N.; Großjohann, K. (2002). HyREX: Hyper-media Retrieval Engine for XML. In: Järvelin, K.; Beaulieu, M.; Baeza-Yates, R.; Myaeng, S. H. (eds.): *Proceedings of the 25th Annual International Conference on Research and Development in Information Retrieval*, page 449. ACM, New York. Demonstration.

Gövert, N.; Kazai, G. (2003). Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002. In: Fuhr, N.; Gövert, N.; Kazai, G.; Lalmas, M. (eds.): *INitiative for the Evaluation of XML Retrieval (INEX). Proceedings of the First INEX Workshop. Dagstuhl, Germany, December 8–11, 2002*, ERCIM Workshop Proceedings, pages 1–17. ERCIM, Sophia Antipolis, France. `http://www.ercim.org/publication/ws-proceedings/INEX2002.pdf`.

The SearX-Engine at INEX'03: XML enabled probabilistic retrieval

Holger Flörke
doctronic GmbH & Co. KG
Adenauerallee 45-49
D-53332 Bornheim, Germany
floerke@doctronic.de

ABSTRACT

In this paper we describe how we used our „out of the box“ search engine for INEX'03. The *SearX-Engine* integrates structural information into the query language and the retrieval function and is based on the widely used probabilistic retrieval (TF*IDF).

1. Query Language

A query in the probabilistic retrieval model can be represented by an unordered set of terms. The user can easily express his information need by specifying some related terms.

To integrate structural assignments and weightings into the query language, we introduce the concept of *roles*. One role (such as 'author' or 'heading') combines all parts of the collection with a common semantics, so the user does not have to know about the specific structure of the underlying collection. The mapping from the collection data to structural roles is done at indexing time by the content provider, who should know about his data. Furthermore roles can help to query distributed and heterogeneous document collections.

The user can integrate structural information into his query by assigning query terms to structural roles and choosing one retrieval role. The retrieval role determines the parts of the collection that should be returned and ranked.

We also support a mechanism to weight roles. If a query term is found, the score of this occurrence will be influenced by the structural context. This weighting is often made by the publisher, who can provide his knowledge about the data and the information needs of the users.

The *SearX-Engine* knows the concept of headings, so structural implications (eg scoring an article title should score all sections within this article) can be expressed. Furthermore the operators "+" (must have) and "–" (must not have) and phrases are supported.

2. INEX'03

To evaluate INEX'03 topics, we made a mapping of the used structural assignments to roles and transformed the topics to our query format described above. Weighting was done to push up hits within article titles, abstracts and keywords.

2.1 CO-Topics

For Content-Only queries we decided to rank always whole articles and search for the title and the keywords of the topic description within the whole article. So we did not make any structural assignments besides the weighting.

```
<inex_topic ct_no="35" query_type="CO" topic_id="100">
  <title>+association +mining +rule +medical</title>
  <description>
    Retrieve information about association rule mining in medical databases </description>
  <narrative> We have a medical data mining ... </narrative>
  <keywords>association, mining, rule, medical</keywords>
</inex_topic>
```

```
<query name="INEX">
  <retrieval-role><constant value="article"/></retrieval-role>
  <filter/>
  <query-item>
    <role><constant value="article"/></role>
    <terms>
      <constant value="+association +mining +rule +medical association mining rule medical"/>
    </terms>
  </query-item>
</query>
```

2.2 CAS-Topics

The CAS-Mapping is somewhat more difficult because of the CAS topic format introduced in INEX'03.

The last element in the path of the title is taken as the retrieval element. Every about-predicate creates one query item (pair of role and terms).

```
<inex_topic ct_no="14" query_type="CAS" topic_id="63">
  <title>
  //article[about(.,'"digital library"') AND about(.//p,'+authorization +"access control" +security')]
  </title>
  [...]
</inex_topic>
```

```
<query name="INEX">
  <retrieval-role><constant value="article"/></retrieval-role>
  <filter/>
  <query-item>
    <role><constant value="article"/></role>
    <terms><constant value=" 'digital library‘ "/></terms>
  </query-item>
  <query-item>
    <role><constant value="article//p"/></role>
    <terms>
      <constant value="+authorization +‘access control‘ +security"/>
    </terms>
  </query-item>
</query>
```

A Filter on an explicit attribute value in the CAS title (eg /article[.//yr <= '2000']) was translated into a *SearX-Engine* filter to exclude document parts based on attribute values.

2.3 Submission

The translated topics were evaluated against the INEX'03 document collection. The XML results produced by the *SearX-Engine* were easily transformed into the submission format.

Retrieving the most relevant XML components

Yosi Mass, Matan Mandelbrod
IBM Research Lab
Haifa 31905, Israel
+972-3-6401627
{yosimass, matan}@il.ibm.com

Abstract
XML enables to encode semantics in full text documents through XML tags. While query results on corpora of full text documents is typically a sorted list of ranked documents, this granularity can be refined to return sub components when searching over XML documents. In this paper we describe an approach for finding the most relevant XML components for a given query.

Keywords
XML Search, Information Retrieval, Vector Space Model

1. Introduction

XML documents represent a family of semi-structured documents in which data has some structure but is not fully structured as in databases. It is thus not surprising that approaches for searching in collections of XML documents are either extension of Information Retrieval (IR) techniques or of database query languages. The main difference between the two approaches is that while the results of Information Retrieval techniques is a list of documents sorted by their relevance to the query, the results of a database query are strict matches with no relevance values. In this paper we focus on Information Retrieval approaches and explore a technique whereby we rank individual XML components rather than full documents.

The Initiative for the evaluation of XML Retrieval (INEX) [7] coined two types of queries over XML documents: In **Content Only (CO)** queries the user has no knowledge of the document structure and the search engine is supposed to return the best components that match the query concepts. In **Content and Structure (CAS)** queries the user has some knowledge of the document structure and can use it to constrain content to a specific structure and also to specify the XML components to be returned.

It should be noted that techniques that are suited for returning a specific XML component that matches a CAS query may be orthogonal to the task at hand, which requires that the best matching component be retrieved. Indeed the version of JuruXML[11] that we used in INEX'02[7] could retrieve XML components as specified by CAS topics yet it could score only full documents. Consequently, all relevant components in a retrieved document where assigned identical scores – the score of their enclosing document, and individual component ranking was not supported.

In modern Information Retrieval engines document ranking is done based on the vector space model[13]. The idea is to treat both the documents and the query as a vector of terms (typically words). Each term is given a weight proportional to its Term Frequency (TF) in a document/query and inversely proportional to its Document Frequency (DF), which is the number of documents in which the term appears. The similarity between a document and a query is defined as the distance between the two vectors usually measured as the cosine between the two.

In order to rank components rather than entire documents, this classic model must be expanded to take into account component level statistics. The problem is that components in XML documents are nested and this hierarchy needs to be taken into account when counting term occurrences. More specifically, a specific term should not be counted more than once. For example consider a term inside a paragraph, which is itself nested in a section. What is the component frequency of this term? If it is counted as belonging to two components, it may distort ranking since the term actually appears only once in the document. On the other hand, if it is counted only once, with which component should this count be associated?

In this paper we describe an extension to the classic vector space model that can correctly handle retrieval at the component level. We demonstrate the use of this method on the INEX topics. This method can be implemented as an extension of any vector space based text search engine with no need to modify its basic structures and algorithms, making it highly applicable for any search engine wishing to rank components.

The remainder of the paper is organized as follows: In section 2 we outline some related work. In section 3 we describe our novel approach for selecting the most relevant XML component and how it was used for the INEX CO topics. In section 4 we show how this method was extended to handle the CAS topics. Our method for result clustering and for filtering redundant components is described in Section 5. We conclude with summary and future work.

2. Related Work

The idea of ranking document subcomponents has been explored in the context of *passage retrieval* [10],[14]. The goal there is to identify the sentences that best match the user's query and assemble them into passages that are then returned to the user. The returned unit can be any combination of sentences even if they are inconsecutive. This technique is not suitable for XML components retrieval where the returned unit must be a fixed XML component.

The work described in [12] tries to identify subject boundaries in a text document based on the assumption that words that are related to a certain subject will be repeated whenever that subject is mentioned. Again this work assumes a flat text document with freedom to pick a portion of the text as an answer. This is not suitable for XML retrieval where the retrieval unit must be a predefined XML component.

The idea of scoring XML components separately has been suggested in the context of XML retrieval [5] [6]. In both cases, the term and document frequency is accumulated at the basic component level. An augmentation factor is used to propagate statistics from child to parent components. The problem with this technique is that the augmentation factors are either set manually by the user or set empirically and thus cannot be proven to give the best results.

3. Approach for Content Only topics

We start by describing our approach for Content Only (CO) tasks and then we show how this approach was extended to handle Context and Structure (CAS) topics as well.

As a reminder, in a CO task the query is specified in full text (with additions of +/- and phrases) and the search engine is expected to return the most relevant XML components that match the query concepts.

Based on a training set composed of the INEX'02 topics and assessments, we found that the majority of the highly ranked components for CO topics (1296 out of 1394) were taken from the set: {article, bdy, abs, sec, ss1, ss2, p and ip1}. This is quite intuitive since {sec, ss1, and ss2} stand for sections and sub sections and {p, ip1} represent meaningful paragraphs, all good reasonable results for a query. The entire article or its abstract {abs} are also good candidates for component retrieval. The only exception is the {bdy} component that constitutes the main part of the article so whenever a bdy is relevant so is its containing article and vice versa. In that case we rather return the article and not the body.

Realizing that we have a clear list of candidate components for retrieval, our goal was to modify JuruXML[11] so that it could rank each of these candidate components separately. The ranking method used in JuruXML is based on the Vector Space Model where both documents and queries are represented as vectors in a space where each dimension represents a distinct term. It is typically computed using a score of the *tf x idf* family that takes into account the following document and collection statistics -

1. **N** - Total Number of documents in the collection
2. **Term Frequency** $TF_D(t)$ – number of occurrences of a term *t* in a document *D*
3. **Document Frequency** DF(t) – total number of documents containing a term *t*

The relevance of the document D to the query Q, denoted below as $\rho(Q,D)$, is then evaluated by using a measure of similarity between vectors such as the cosine measure (see Formula 1).

$$\boxed{\rho(Q,D) = \frac{\sum_{ti \in Q \cap D} w_Q(t_i) * w_D(t_i)}{\|Q\| * \|D\|}}$$

Formula 1

Where

$$\boxed{w_{X \in \{Q,D\}}(t) = \log(TF_X(t)) * \log(\frac{N}{DF(t)})}$$

Formula 2

It follows that the weight $W_D(t)$ is proportional to the number of occurrences of *t* in D ($TF_D(t)$) and inversely proportional to the number of documents in which t appears (DF(t)). The motivation is that a term *t* that appears in a few documents in the corpus, should contribute a relatively high weight to the score of a document in which it appears compared to terms that are frequent in many documents. The contribution to the document score is additionally proportional to the number of its occurrences in the document.

In order to rank components instead of entire documents, these statistics should be tallied at the component level.

That is, it is necessary to keep track of the following component and collection statistics:

1. **N** - Total Number of components in the collection
2. **Term Frequency** $TF_C(t)$ – number of occurrences of a term *t* in a component *C*
3. **Component Frequency** CF(t) - total number of components containing a term *t*

The problem is that XML components are nested. For example consider a collection consisting of a single document (see Figure 1).

```
<article>
    t1
    <sec>
        <p>t2</p>
    </sec>
</article>
```

Figure 1

The document contains three components $\{C_1$=article, C_2=sec, C_3=p$\}$ and two terms $\{t_1, t_2\}$. Term t_1 appears only in the article while t_2 appears in all 3 components. Therefore we get

- N = 3
- $CF(t_1) = 1$, $CF(t_2) = 3$
- $TF_{C1}(t_1) = 1$, $TF_{C1}(t_2) = 1$
- $TF_{C2}(t_1) = 0$, $TF_{C2}(t_2) = 1$
- $TF_{C3}(t_1) = 0$, $TF_{C3}(t_2) = 1$

By Formula 2 applied to component level statistics, we would get that $W_{c1}(t_1) > W_{c1}(t_2)$ which is not necessarily true since both t_1 and t_2 appear an equal number of times in the document.

One can try to fix this by only counting the Term Frequency $TF_C(t)$ at the component level and still computing N & DF(t) at the document level. However, this imposes another problem that is illustrated in the following example (see Figure 2).

```
<article>
    <sec>t1</sec>
    <sec>t1</sec>
    <sec>t2</sec>
</article>
```

Figure 2

As before, the collection consists of a single document and we have

- N = 1
- $DF(t_1) = 1$, $DF(t_2) = 1$
- If we mark the 3 sections by C_1, C_2, C_3 we get
    - $TF_{C1}(t_1) = 1$
    - $TF_{C2}(t_1) = 1$
    - $TF_{C3}(t_2) = 1$

By Formula 2 it follows that $W_{c1}(t_1) = W_{c2}(t_1) = W_{c3}(t_2)$. However if we regard each section as a standalone component then since t_2 appears only in one section while t_1 appears in 2 sections we expect to get $W_{c1}(t_1) < W_{c3}(t_2)$ (which is what would have happened if the sections were in different documents, since we would then have then $DF(t_1) = 2$). With this approach to counting statistics it is thus impossible to differentiate between the rankings of the three sections.

In view of the above problems, we selected a strategy whereby we create a different index for each component type. Statistics can thus be tallied at the precise level of granularity for each component. In particular, we created six indices corresponding to the following tags: {article, abs, sec, ss1, ss2, p, and ip1[1]}. The *article* index contains the full data of all documents. The *sec* index contains each sec from each article as a separate document and so on for each of the six tags above. For example the document in Figure 2 above will result in 3 separate documents in the *sec* level index.
For each index, the entities are determined according to the topmost XML tag of the corresponding type. That is, nested components of the same type do not yield a new partition of the document. For example consider a document as in Figure 3

```
<Article>
    <sec>
        <p>some text
            <p>some internal text</p>
        </p>
    </sec>
    <p>some higher level text</p>
</article>
```

Figure 3

This document will add two "documents" to the paragraph level index (See Figure 4 & Figure 5)

```
<p>some text
    <p>some internal text</p>
</p>
```

Figure 4

[1] P and IP_1 were indexed into one Index

And

<p>some higher level text</p>

Figure 5

The search engine's regular ranking formula can now be used to accurately score and rank individual components among themselves. In other words, given a query, the system can return the best matching articles, sections, sub-sections, etc. Our goal however is to return one ranked list of the best matching components regardless of granularity and thus need to compare scores from the individual indices. To achieve this, the query is submitted in parallel to each index, resulting in six sorted lists of components – one from each index.

The scores in each index are normalized into the range (0,1) using a formula that ensures that this normalization yields absolute numbers and is index independent. This is achieved by each index computing *P(Q,Q)* (see Formula 1) which is the score of the query itself as if it was a document in the collection. Since the score measures the cosine between vectors, then the max value is achieved between two identical vectors. Each index therefore normalizes all its scores to its computed *P(Q,Q)*. The normalized results are then merged into a one ranked list consisting of components of all granularities.

It should be noted that this approach can be implemented on top of almost any full text ranking engine resulting in a system than is able to rank XML omponents without modifying the core search engine code. It simply requires an XML parser that can parse documents and feed the components into separate indexes. At run time, queries are submitted to each index and the results are merged as described above.

3.1 The CO runs

We now describe the implementation of this method on the INEX collection. The size of the collection is ~500Mb. Six indices were created as described above, resulting in the following index sizes:

- Article – 290Mb
- Sec – 270Mb
- Ss1 – 158Mb
- Ss2 – 38Mb
- P, ip1 – 280Mb
- Abs – 14Mb

Overall we get an index size that is about twice as large as the original collection. While this can be an inhibiting factor, our goal was to prove the viability of this method from a quality standpoint. We believe there is room for optimisation in terms of index sizes.

We submitted three CO runs. Recall that a CO topic consists of full text with additions of +/- and Phrases. According to the topic development guide [8] the +/- "should be interpreted with a fuzzy flavour and not simply as must contain and must not contain conditions". We applied this vagueness to "+" terms but still we believe that if the user specify a "-" term then this term should not be returned. Therefore we treated the "-" strictly (namely results that contain such terms were never returned). The runs we submitted were -

- In the first run we considered all query parts: Title, Description and Keywords.
- In the second run we applied post clustering on the first run (see Section 5 below).
- In the third run we considered only the Title. In this run we ignored phrases and treated the phrase terms as regular words. We applied the clustering algorithm on this run as well (see Section 5 below).

At the time this report is written, Recall Precision graphs for the CO topics were published and our run that used both Title Description and Keywords was ranked first

4. Approach for CAS Topics

The Content and Structure (CAS) topics differ in two aspects from the CO topics. First the query content can be limited to a given XML tag and second there is less freedom in selecting the component to be returned. The topic format is an extension of XPath[15] and as such the last component in the path specifies the component that should be returned.

For example topic 66 (Figure 6) defines a constraint on the year <yr> and on section <sec>. <sec> is also the element to be returned.

```
/article[./fm//yr < '2000']
        //sec[about(.,'"search engines"')]
```

Figure 6

To enable fuzziness in the query constraints we introduce a *Synonyms* mechanism. We divide the XML tags into synonym groups such that all tags in the same group are regarded equivalent. Whenever there is a tag in the query that belongs to some synonym group, we substitute it by all tags in its group.

For example if we set {sec, ss1} to be in the same synonym group then in the query in Figure 6 above we substitute *sec* by {sec, ss1} and we get[2] the query in Figure 7. The synonyms mechanism is used at different granularity levels for the SCAS and VCAS runs (see below).

```
/article[./fm//yr < '2000']
        //{sec,ss1}[about(.,'"search engines"')]
```

Figure 7

In order to find documents in which all of the query constraints are met, we need to execute the modified query on the full documents. This will indeed return relevant components that match the query constraints, but as described above the components cannot be scored individually using only this one index.

Therefore we execute each query in two steps – in the first step we use the *article* index to locate candidates that fulfill the query constraints. In the second step, relevant parts of the query are extracted for each index (see example below) and the relevant query is submitted in parallel to the other five indexes of {abs, sec, ss1, ss2, p+ip1}. A relevance value is computed only for elements that were marked valid in the first step and a ranked list of results is returned. The separate lists are then merged similarly to what described for the CO case, resulting in one ranked list of results.

Note that although our indices do not cover all of the possible tags in the corpus, we can still resolve queries that request a tag that does not have a dedicated index. For example, topic 67 defines <fm> as the last component in the XPath expression, thus requesting a component for which we do not have a special index. In this case, we simply stop after the first step and use the article's score as the score of the component.

Example

In the following example we define one synonym group that consists of {sec, ss1, ss2} tags and we use it to run the query in Figure 6 above. We run the query first against the *article* level index and then we run the relevant query part on each of the synonyms ∈ {sec, ss1, ss2} so we run

//sec[about(.,'"search engines"')]

against the *sec* level index,

//ss1[about(.,'"search engines"')]

against the *ss1* level index and

//ss2[about(.,'"search engines"')]

against the *ss2* level index. We then merge the results based on their normalized scores as described above.

[2] This is not the syntax we use, the actual substitution is done in the internal implementation.

4.1 SCAS and VCAS

This year there were 2 CAS variants - Strict CAS (SCAS) and Vague CAS (VCAS). The SCAS defines that "structural constraints of a query must be strictly matched" while the VCAS defines that "structural constraints of a query can be treated as vague conditions". The vague means that XML elements that are "structurally similar" to those specified in the query can be returned. We used our synonym groups in different configurations to support both SCAS and VCAS.

For the SCAS runs we used the equivalent tags that were defined in the INEX topic development guide[7]. The synonyms we used were:

- {sec, ss1, ss2} for sections.
- {p, ip1} for paragraphs

The other two tags {article} and {abs} were not synonyms to any other tags so in topics that requested article or abs as results, only those tags were returned.

For the VCAS topics we defined one large synonym group that included all the tags {sec, ss1, ss2, p, ip1, abs}, except for the article tag. Again in topics that requested the article tag as a result we returned only articles.

4.2 The CAS runs

We submitted 3 SCAS and 3 VCAS runs. In all runs we treated the "-" strictly and the "+" with a fuzzy flavour. In all runs we treated query constraints in a strict manner up to the synonym tags. So for example results for the query in Figure 6 will be only sections and all their synonym tags that discuss "search engines" but only from articles that were published before year 2000.

We ran the following 3 runs for both SCAS and VCAS

- In first run, we considered all query parts: Title, Description and Keywords.
- In the second run, we applied a post-clustering algorithm on the first run (see Section 5 below).
- In the third run, we considered only the Title and Keywords and applied a post-clustering algorithm (see Section 5 below).

5. Result clustering

The approach described above may result in redundant components that are returned to the user. For example consider a section with four paragraph children. We can identify two extreme scenarios -

- In the first scenario assume all four paragraphs are highly relevant to the topic. In this case all four paragraphs as well as their parent section will be ranked in high positions.

- In the second scenario assume that only the first paragraph is very relevant to the topic and therefore it is assigned a high score. As a result it may also contribute to its parent's section score even if it is the only relevant paragraph in that section. Again that paragraph and its parent section may be ranked in a high position when merging the results.

One expectation of a good search engine is that it should not return redundant results; therefore in the first scenario it should return the section and not the paragraphs, while in the second scenario it should return the first paragraph only.

To filter such redundancies we developed a clustering algorithm that maps related components to one of the scenarios above. The algorithm gets the result set of the original run and constructs a tree consistent with the parent-child relationship of the components in the XML document. Each node in the tree corresponds to a result component and has the following data -

- Its score as a number between 0 and 1
- Total number of descendant children in the original document. This number is extracted while parsing the document.

The algorithm processes the tree bottom up and at each level compare the score of a node to that of its children. When it manages to identify one of the two scenarios above it remove the redundant components from the result set.

Recall that a score is a number between 0 and 1 so we need some means to compare two scores. Let a node's score be *s1* and a child's score be *s2*. We say that the two scores are *close* if $\frac{|s1 - s2|}{s1} < ScoreThresh$

For some configured *ScoreThresh* value. Otherwise we say that s1 is *higher* than s2 (if s1>s2) or *lower* (if s1<s2).

The algorithm clusters each node into the following cases –

1. **HighParent** - If the node's score is *higher* than all its direct children, then we remove the children from the tree.
2. **HighChild** - If some child's score is *higher* than the node's score, then we remove the node from the tree.
3. **ManyDescendants** – Let N_e be the number of *close* descendants and N_t the number of all descendants of our node. If $\frac{N_e}{N_t} > ManyDescendantsThresh$ for some configured *ManyDescendantThresh* then we remove the direct children from the tree (corresponding to the first scenario above)
4. **SingleChild –** For each direct child C_i let N_i be the number of *close* descendants of C_i and N the total number of *close* descendants of the current node. If there is a child C_i with $\frac{Ni}{N} > SingleChildThresh$ for some configured *SingleChildThresh* value then we say that most good results are concentrated in that child so we remove the parent from the tree (corresponding to the second scenario)
5. For all other cases no filtering takes place, and all components are returned

5.1 Clustering runs

We used the following values for the clustering runs:

- *ScoreThresh*=0.45
- *ManyDescendantThresh* = 0.2
- *SingleChildThresh*=0.42.

According to the INEX evaluations received thus far, it seems that the runs that applied clustering received a lower overall score than runs that applied clustering. It thus seems that there was no penalty for runs returning redundant results. This topic should be discussed in order to devise metrics that evaluate a good overall result set, rather than individual results.

6. Summary

We presented a novel approach and implementation for scoring and ranking individual components of XML documents. At the time this report is written, Recall Precision graphs for the CO topics were published and one of our runs was ranked first indicating that this approach indeed computes more accurate component scores. The approach presented here can be implemented on top of almost any full text search engine without modifying its code to return ranked components for Content Only queries. Similarly the approach can be used by XML search engines to compute more accurate scores for target components specified in CAS topics. One limitation of our approach is that the set of potential components to be returned must be known in advance. We believe however, that this is a reasonable requirement for any given collection. Additionally, some space as well as runtime overhead is incurred by multi-indexing. Improving the efficiency is left for future research.

7. Acknowledgment

We would like to thank Aya Soffer for reviewing the paper and for her useful comments.

8. References

[1] R. Baeza-Yates, N. Fuhr and Y. Maarek, *Second Edition of the XML and IR Workshop,* In SIGIR Forum, Volume 36 Number 2, Fall 2002

[2] D. Carmel, E. Amitay, M. Herscovici, Y. Maarek, Y. Petruschka and A. Soffer, "Juru at TREC 10 - Experiments with Index Pruning", In [1]

[3] D. Carmel, N. Efraty, G. Landau, Y. Maarek, Y. Mass, "An Extension of the Vector Space Model for Querying XML Documents via XML Fragments" In XML and Information Retrieval workshop of SIGIR 2002, Aug 2002, Tampere, Finland.

[4] D. Carmel, Y. Maarek, M. Mandelbrod, Y. Mass, A. Soffer, Searching XML Documents via XML Fragments, SIGIR 2003, Toronto, Canada, Aug. 2003

[5] N. Fuhr and K. GrossJohann, "XIRQL: A Query Language for Information Retrieval in XML Documents". In *Proceedings of SIGIR'2001*, New Orleans, LA, 2001

[6] T. Grabs and H. J. Schek, "Generating Vector Spaces On-the-fly for Flexible XML Retrieval", in [2].

[7] INEX'02 , Initiative for the Evaluation of XML Retrieval, http://qmir.dcs.qmul.ac.uk/inex/

[8] INEX'03 Topic development guide, http://inex.is.informatik.uni-duisburg.de:2003/internal/#topics

[9] Juru, A generic full text search engine, http://w3.haifa.ibm.com/afs/haifa/proj/docs/www/projects/km/ir/juru/index.html

[10] M. Kaszkiel and J. Zobel, "Effective ranking with arbitrary passages", Journal of the American Society of Information Science, volume 52, number 4, pg 344-364, 2001.

[11] Y. Mass, M. Mandelbrod, E. Amitay, D. Carmel, Y. Maarek, A. Soffer, JuruXML - an XML Retrieval System at INEX'02, Proceedings of the First Workshop of the Initiative for The Evaluation of XML Retrieval (INEX), 9-11 December 2002, Schloss Dagstuhl, Germany

[12] K. Richmond, A. Smith and E. Amitay, "Detecting Subject Boundaries within Text: A Language Independent Statistical Approach", Proceedings of the Second Conference on Empirical Methods in Natural Language Processing, pg 47-54, 1997

[13] G. Salton, Automatic Text Processing – The Transformation, Analysis and Retrieval of Information by Computer, Addison Wesley Publishing Company, Reading, MA, 1989.

[14] G. Salton, J. Allan, and C. Buckley. "Approaches to passage retrieval in full text information systems." In Proceedings of SIGIR'93, Pittsburgh, PA, 1993.

[15] XPath – XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20/

[16] XQuery – The XML Query language, http://www.w3.org/TR/2002/WD-xquery-20020430

Searching in an XML Corpus Using Content and Structure

Yiftah Ben-Aharon Sara Cohen Yael Grumbach Yaron Kanza
Jonathan Mamou Yehoshua Sagiv Benjamin Sznajder Efrat Twito

The Selim and Rachel Benin
School of Engineering and Computer Science
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
{yiftahb, sarina, yagrum, yarok, mamou, sagiv, sznajder, efrat111}@cs.huji.ac.il

1. INTRODUCTION

An XML document has a structure in addition to content, and an XML search engine should be capable of taking advantage of the structure in order to improve the quality (i.e., precision and recall) of the results. The structure may also be incorporated into the topic (i.e., query) in two ways. First, the topic may include conditions that relate content to structure (e.g., some keyword should appear in the title of the document). Second, the topic may specify the exact fragment of the document that should be returned as an answer. Even if the topic does not have any hint about the structure, the search engine should still be able to find not just the relevant documents, but also the most relevant fragment (or fragments) within each document.

Several different paradigms have been proposed recently for searching XML documents. In XRANK [7], the main idea is a generalization of the Page-Rank [4] technique of Google [1]. In XSEarch [6], the emphasis is on retrieving only those answers that consist of *semantically related* nodes. Neither one of these approaches is suitable for the INEX corpus, which consists of articles from the IEEE digital library. The XRANK approach is not directly applicable to INEX, since the XML documents of INEX do not have cross references in the form of IDREFs or XLinks. The XSEarch approach is irrelevant to INEX, since all the nodes of any single XML document are deemed semantically related.

Our approach consists of a variety of Information Retrieval techniques augmented with the ability to give different weights to different fragments of a document, based on the tags. Specifically, we use term frequencies, inverse domain frequencies, proximity among occurrences of keywords, and similarity between keywords and words from the given document. Each technique has been implemented as a separate ranker and the final ranking is done by merging the results of the various rankers.

2. TOPIC SEMANTICS AND SYNTAX

The query language of a standard search engine is simply a list of keywords, optionally preceded by the + or − sign. In the context of XML, the query language can also contain information about the structure, in the form of path expressions that describe specific parts of a document where the keywords should appear.

In INEX'03 [9], a query is called a *topic* and comprises four parts: (1) *title:* this part describes the topic in a formal syntax, (2) *description:* a description in a natural language of the information that is needed, (3) *narrative:* a more detailed description, and (4) *keywords:* a set of comma-separated *terms,* where a term is a single keyword or a phrase encapsulated in double quotes. Our system uses only the title.

A topic can be either *content only* (abbr. CO) or *content and structure* (abbr. CAS). In a CO topic, the title contains only content-related conditions; it is a set of space-separated terms, optionally preceded by the + or the − sign. For example,

```
+database +''java programming''
```

is a CO topic about "database" and "java programming."

In a CAS topic, the title relates terms to specific locations in documents. The general form of the title is `CE[filter] CE[filter] ... CE[filter]`, where each `CE` is a *context element* that specifies a path in the document (using XPath syntax). A `filter` is a Boolean combination of XPath predicates (e.g., a comparison between a path expression and a constant) and predicates of the form `about(path,string)`, where `path` is an XPath expression and `string` is a quoted string of terms (each term could be preceded by a + or −). For example,

```
//article[.//@yr > '2000']
  //sec[about(.,'+''java programming''')]
```

specifies that sections about "java programming" from articles written after 2000 should be retrieved.

We use T to denote a CO or a CAS topic. By a slight abuse of notation, T also denotes the list of all stemmed terms appearing in the title of T. (stop words are eliminated). T_+ denotes the list of terms in T that are preceeded by a + sign, T_- denotes the list of terms that are preceded by a − sign, and T_o is the list of all the remaining (i.e., optional) terms in T.

3. AN OVERVIEW OF THE SYSTEM

The design of our system was influenced by two major considerations. First, our goal was to build an extensible system so that various information-retrieval techniques could be combined in different ways and new techniques could be easily added. Second, the system had to be developed in a very short time.

The first consideration led to the decision to implement each information-retrieval technique as a separate ranker and to implement a merger that would merge the results of the individual rankers.

The second consideration influenced the implementation of the topic (i.e., query) processor. In INEX, a topic may include expressions in XPath (augmented with the "about" function) that refer to the structure of the documents to be retrieved. Thus, an XPath processor is needed in order to evaluate a given topic. However, any existing XPath processor cannot be applied to the complete description of a topic that is written in the formal syntax of INEX; instead, it can only be applied separately to each XPath expression that is embedded inside the topic. This is not sufficient for an accurate processing of CAS (content and structure) topics, since when different XPath expressions from the same topic are evaluated separately, it is impossible to tell how to combine their results correctly. So, it seemed that the topic processor would require a complete implementation of an XPath parser (and that would be time consuming). Instead, we implemented (in Java) a parser for INEX topics that creates an XSL stylesheet (i.e., a program written in XSL). Since XPath is included in XSL, we circumvented the need to implement an XPath parser as a part of our topic processor.

Figure 3 depicts the main components of the system. The first step is building the indices, which are described in detail in Section 4. Given a topic, the indices are used to filter the whole corpus in order to retrieve the documents that contain all the required keywords (i.e., keywords preceded by +). Documents that pass through the filtering phase are processed by an XSL stylesheet that is generated from the topic. The XSL stylesheet retrieves from each document all the fragments that are relevant to the processing of the given topic. The retrieved fragments are produced as an XML file (one per document) in a manner that preserves the original hierarchy among these fragments. In the next step, each ranker process all the XML files and creates a new XML file of the ranked results. In the final steps, the results of the various rankers are merged into a single XML file.

4. INDEXING

The system uses several indices when processing topics (i.e., queries). The creation of the indexes is done as a preprocessing step by the *indexer*. The indices are described below.

Document-Location Array

The system assigns a unique *document identifier* (also called *did*) to each document. The *document-location array* is used to associate each *did* with the physical location, in the file system, of the corresponding document.

Inverted Keyword Index

The *inverted keyword index* associates each keyword with the list of documents that contain it. Stop words, i.e., words that are used very frequently in English (e.g., "in," "to," "the," etc.) do not appear in the index. Also, regular stemming, using the Porter's stemmer [12], is done in order to achieve a higher flexibility when searching for a particular keyword. The inverted-keyword index stores stems of words. For each stem w, there is a *posting list* of the *did*'s of all the documents that contain some keyword with stem w.

Keyword-Distance Index

The *keyword-distance index* stores information about proximity of keywords in the corpus. For each pair of keywords, the system computes a score and the keyword-distance index holds this score. The score reflects the number of occurrences of that pair of keywords in any single sentence. It also reflect the distance between the two keywords when they appear in the same sentence. The score for a given pair of keywords is the sum of the inverse of the distance between the two keywords over all the sentences in all the documents of the corpus. Formally, the score of the pair (w_i, w_j) is

$$D(w_i, w_j) = \sum_{d \in \mathcal{D}} \sum_{s \in d} \sum_{(w_i, w_j) \in s} \frac{1}{distance(w_i, w_j)}$$

where $\mathcal{D}$ is the set of all the documents in the corpus, d is a document, s is a sentence, and $distance(w_i, w_j)$ is the number of words separating w_i and w_j. Scores are normalized and $D(w, w)$ is defined to be 1 (the maximum). The keyword-distance index actually stores the scores for pairs of stems rather than complete keywords.

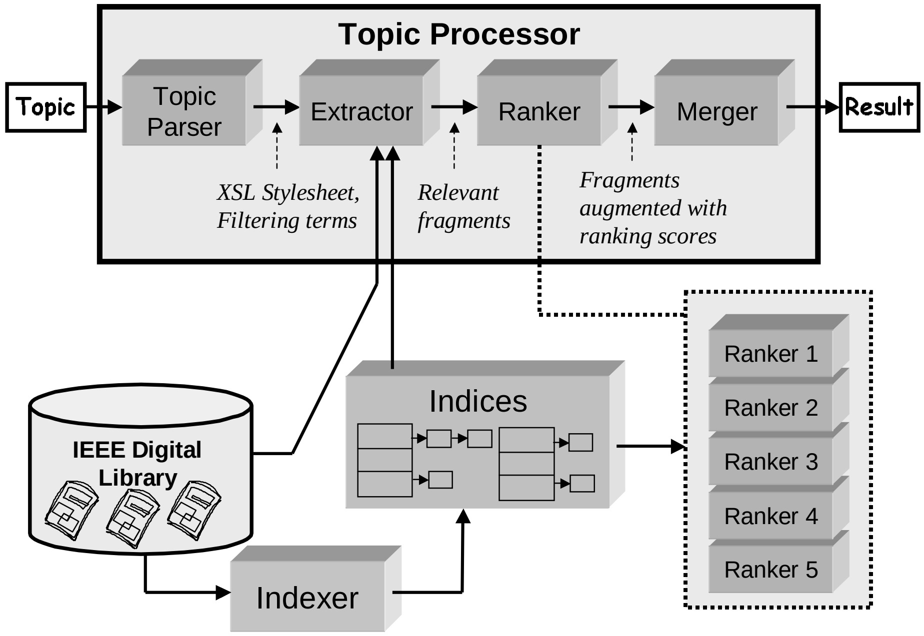


Figure 1: Overall Architecture

Tag Index

Tags are given weights according to their importance. The weight of each tag is a parameter that can be easily modified by anyone who uses the system. A Java property file stores the weight $tw(t)$ of each tag t.

Inverse-Document-Frequency (idf) Index

The *document frequency* of a keyword k is the number of documents that contain k, divided by the total number of documents in the corpus. The *inverse document frequency* is defined as follows:

$$idf(k) := \log\left(1 + \frac{|\mathcal{D}|}{|\{d \mid d \in \mathcal{D} \text{ and } k \in d\}|}\right)$$

where $\mathcal{D}$ is the set of all the documents in the corpus. The *inverse-document-frequency index* is a hash table that holds the inverse document frequency for the stem k of each keyword.

5. TOPIC PROCESSING

The processing of a topic T is done in four phases. In the *filtering phase,* the documents that contain all the keywords in T_+ are retrieved from the corpus. In the *extraction phase,* the relevant fragments are extracted from each document. In the *ranking phase,* the fragments from the previous phase are ranked by each ranker. In the *merging phase,* the results of the various rankers are merged together. Next, we describe each phase in detail.

In the filtering phase, for each keyword $k \in T_+$, the posting list L_k of the stem of k is extracted from the inverted keyword index. The intersection $L_T = \cap_{k \in T_+} L_k$ is computed and the result is an XML document that contains a list of all the *did*'s of the documents in L_T.

In the extracting phase, an XSL stylesheet is generated from the title of the topic. This stylesheet extracts the relevant fragments from each document that passed the filtering phase. For CAS topics, the relevant fragments are determined by the title. For CO topics, the system has to determine which fragments are relevant. In our system, the fragments that could be returned are determined in advance; this policy was proposed by [3] and is also used in XRANK [7]. Specifically, these fragments are either the whole document, the front matter, the abstract, any section or any subsection.

A potentially relevant fragment must also satisfy some conditions. For one, it must include all the terms that are preceded by +. Moreover, it may have to satisfy some predicates, e.g., `.//@yr > '2000'`. Thus, extracting the relevant fragments requires a processor that is capable of parsing titles of CO and CAS topics. An XPath processor is not suitable for the job, since the syntax of titles is more general than that of XPath. In our system, a Java program parses the title and generates an XSL stylesheet that does the extraction. Since XPath is included in XSL, portions of the title that adhere to the XPath syntax can be transplanted into the stylesheet. This has lead to a fast implementation of the topic processor.

In the title of a CAS topic, there is a *core path expres-*

sion that consists of the concatenation of all the context elements. There are also several *filter path expressions,* where each one is a path expression that appears in some filter concatenated with all the context elements that precede it. The last element of the core path expression and the last element on any filter path expression are called *target elements.* For example, consider the following CAS title:

```
//a[about(.//b, '...')]//c[about(.//d, '...')]
```

The core path expression is `//a//c` and `c` is the target element of this path. The fragments that eventually will be returned as answers for this title are `c` elements. The filter path expressions are `//a//b`, with the target element `b`, and `//a//c//d`, with the target element `d`. Fragments that are `b` and `d` elements should be extracted in order to check the conditions that are specified in the `about` clauses.

Extracting fragments for the target elements and checking that each one satisfies its corresponding condition is not quite enough. In order for the rankers to work correctly, it is important to know whether a fragment extracted for a `b` element is related to a fragment extracted for a `d` element, in the sense that both have the same `a` element as an ancestor. Therefore, the XSL stylesheet extracts fragments for the target elements in a manner that preserves the original hierarchy among these fragments. Essentially, the stylesheet has a sequence of nested loops, one for each target element. The nesting of the loops follows the hierarchy dictated by the the core and filter path expressions. Each loop extracts all the fragments for its corresponding target element. Each extracted fragment is assigned a level number, which is the level of nesting of its corresponding loop. For example, in the above title, the extracted `d` elements are descendants of the extracted `c` elements and, hence, will have a larger level number.

The XSL stylesheet is applied to all the documents that passed the filtering phase and it produces a new XML document, D_T, that contains for each extracted fragment *(1)* the URL of the parent document, *(2)* the path of the fragment in the document, *(3)* the stems of keywords from the title that appear in the fragment, *and (4)* the fragment itself. The XML file D_T is given to each ranker. Note that the `about` predicate can be evaluated by the rankers, since the relevant fragments are in D_T. The ranking of fragments and the final merging are explained in the next section.

6. RANKING THE RESULTS

We implemented five rankers, namely *word-number ranker*, *idf ranker*, *tf-idf ranker*, *proximity ranker* and *similarity ranker*. Each ranker gives scores to the fragments that are listed in the XML file D_T. This section describes the five rankers and how their results are merged.

6.1 Word-Number Ranker

Recall that T_o is the list of optional terms (i.e., not preceded by the $+$ or $-$ sign) from the title of a given topic T. Similarly, T_- is the list of terms that should not appear in the result (i.e., preceded by the $-$ sign). Given a fragment F, the number of optional terms that appear in F is $|T_o \cap F|$ and the number of unwanted terms in F is $|T_- \cap F|$. The score given to F by the word-number ranker is

$$|T_o \cap F| + 1 - \frac{\min(|T_- \cap F|, 10)}{10}.$$

Note that the score is increased when the number of optional terms appearing in the fragment F is increased and it is decreased when the number of unwanted terms in F is increased. Also note that the weight that is given to the appearance of a wanted term is an order of magnitude greater than the weight given to the appearance of an unwanted term. Moreover, there is a bound of 10 on the total number of unwanted terms that are taken into account.

6.2 Inverse-Document-Frequency (IDF) Ranker

We first give the intuition behind the idf ranker. Consider two fragments F_1 and F_2, such that $F_1 \cap T_+$ and $F_2 \cap T_+$ contain single keywords, w_1 and w_2, respectively. Also, assume that the intersection of F_1 and F_2 with T_- is empty. In this case, the word-number ranker returns the same score for F_1 and F_2. If, however, w_1 is a frequent word in the corpus and w_2 is a rare one, then F_2 should be given a higher score than F_1.

Let F be a given fragment. In the idf ranker, a rare keyword that appears in F has a greater effect on the score than a keyword that appears frequently in the corpus. The score of the idf ranker is the following sum of the idf values of the optional words and the unwanted words that appear in F.

$$\sum_{k \in \{T_o \cap F\}} idf(k) - \sum_{k \in \{T_- \cap F\}} idf(k)$$

Note that terms of T_+ are not considered by this ranker, since all the fragments contain them.

6.3 Tf-Idf Ranker

The tf-idf ranker uses a model similar to the vector-space model that is common in information retrieval [2]. We have modified the basic technique so that the weights given to tags will be incorporated in the computation of the ranker's score.

Let T be a given topic and let F be a fragment. We assume that all the words in T and in F are stemmed. We also assume that all the stop words are removed from F and T. The score given by the ranker to F with respect to T is computed using a variation of the standard *tfidf* (term frequency, inverse document frequency) method. Next, we briefly describe *tfidf* and how it is computed in our system.

Let k be a term. The *term frequency (tf)* of k in F is the number of occurrences of k in F (denoted as $occ(k, F)$), divided by the maximal number of occurrences in F of any term. That is,

$$tf(k, F) = \frac{occ(k, F)}{\max\{occ(k', F) \mid k' \in F\}} .$$

Note that a term is likely to have a larger term frequency in a small document than in a bigger one.

The inverse document frequency of k, $idf(k)$, has been defined in Section 4. The *tfidf* of a term k w.r.t. a fragment F, denoted by $tfidf(k, F)$, is $tf(k, F) \times idf(k)$. Note that by taking a log in the *idf* factor, the overall importance of the *tf* factor in *tfidf* is increased.

In our system, each tag has a weight. The default weight is 1. A user can modify the weight of any tag. The *accumulated weight* of a word w in an XML file X is the multiplication of the weights of all the tags of the elements of X in which w is nested. That is, the accumulated weight of w is produced by multiplying all the weights of the tags of elements on the path from the root of X to w. The effect of the weight of tags on the computation of *tfidf* is as follows. For each occurrence of k in F, instead of increasing $occ(k, F)$ by 1, the value of $occ(k, F)$ is increased by the accumulated weight of k for that occurrence.

The value of $tfidf(k, F)$ is normalized as follows.

$$w(k, F) := \frac{tfidf(k, F)}{\sqrt{\sum_{k' \in F} tfidf(k', F)^2}}$$

By definition, $w(k, F)$ is 0 if k does not appear in F. We denote by K the set of all the keywords appearing in the corpus. Each fragment F in the corpus is associated with a vector V_F of size $|K|$. For each keyword k of K, the vector V_F has an entry $V_F[k]$ that holds $w(k, F)$.

For each topic T, we define V_T to be the following vector.

$$V_T[k] = \begin{cases} 1 & \text{if } k \in T_+ \cup T_o \\ -1 & \text{if } k \in T_- \\ 0 & \text{otherwise} \end{cases}$$

The score given to the fragment F by the ranker is the cosine between V_T and V_F. The value of this cosine is proportional to the following sum:

$$\sum_{k \in K} V_F[k] \times V_T[k] = \sum_{k \in T} V_F[k] \times V_T[k]$$

Note that the above equality holds because $V_T[k] = 0$ if $k \notin T$.

6.4 Proximity Ranker

6.4.1 Lexical Affinities for Text Retrieval

The idea behind the proximity ranker is to use *lexical affinities* (abbr. LA) of words. The ranker takes advantage of the correlation between words that appear in a single phrase in a certain proximity.

The notion of lexical affinities for text retrieval was first introduced by Saussure [13]. Later, it was developed by Maarek and Smadja [10], in the context of information retrieval.

Essentially, the ranker works as follows. Given a topic T containing the terms $t_1, \ldots, t_n$, the ranker creates a list that contains all possible pairs of distinct words (t_i, t_j), such that $t_i < t_j$ (words are compared lexicographically). For each fragment F, whenever the ranker finds in F an occurrence of a pair (t_i, t_j) in a single sentence, the score given to F is increased. Different increasing policies can be used.

6.4.2 Lexical Affinities for XML Retrieval

The following explains how LA retrieval is adapted to XML, in general, and to our system, in particular.

Two words that appear very far from each other should not be considered as a LA. A maximal distance must be defined, such that when exceeded, the two words are not considered to be correlated. Martin [11] showed that 98% of LA's relate words that are separated by at most five words within a single sentence. Maarek and Smadja [10] used this result by searching for co-occurrences in a sliding window (within a single sentence) of size 5. We have adapted this result to the context of XML as explained below.

In XML, structure and content are combined. Due to this lack of separation between structure and content, an XML file can have a logical unit of text in which the text does not appear in a sentence delimited by full stops, but rather delimited by tags. For example, consider the following XML fragment.

```
<author>John Washington</author>
<address>New Jersey State</address>
```

The absence of a full stop between Washington and New Jersey State could be mistakenly interpreted as a case where Washington State is a LA. In order to avoid such mistakes, we consider a closing tag followed by an opening tag as a delimiter of a logical unit.

When looking for lexical affinities in a topic (i.e., query), special attention must be paid to the structure of the topic in order to avoid an attempt to pair words that do not appear under the same tag. For words that are not under the same tag, a LA should not be created. For example, consider the following topic title.

```
//article//fm[
 (about(.//tig, '+software +architecture')
  or about(.//abs, '+software +architecture'))
  and about(., '-distributed -Web')]
```

In this topic, the pairs "software architecture" and "distributed Web" should be considered as LA's. The pairs "distributed architecture" and "software Web" should not be considered as LA's.

For words that appear under the same tag, but some of them in quotation marks, the LA's in quotation marks are given a larger weight. For example, consider the following topic.

```
/article[about(./fm/abs,
'"information retrieval" "digital libraries"')]
```

The pairs "information retrieval," "digital libraries," "information digital," "information libraries," "retrieval digital" and "digital libraries" are all considered as LA's. However, the occurrences of "information retrieval" or "digital libraries" in a fragment get a larger weight than the occurrences of "retrieval digital" or "digital libraries."

6.5 Similarity Ranker

The idea behind the similarity ranker is that if two words appears very frequently in proximity in the corpus, then they should be considered as related concepts. For example, if we find that "SQL" and "databases" are two words that frequently appear together, then we may conclude that the two words are closely related. Therefore, when looking for documents about databases, we may as well search for documents about SQL.

Let F be a fragment of a document D. F_T denotes the terms appearing either in F, in the title of D or in the abstract of D. As usual, T denotes the terms in the title of a given topic. The similarity ranker computes the score of F w.r.t. the given topic T according to the formula

$$\prod_{k \in T} \sum_{w \in F_T} (tw(tag) * D(k, w))$$

where tag is the tag with the largest weight among those containing w. The similarity ranker uses the keyword-distance index in order to get the value of the distance $D(k, w)$. The tag index is used in order to get the value of $tw(tag)$.

This ranker can be seen as an automatic query refinement. It differs from the work of Jing and Croft [8], since we do not use a probabilistic approach. It also differs from the work of Carmel et al. [5], since our refinement uses a global analysis of the *whole* corpus and assigns weights to all the co-occurrences in the fragment, rather than just to a limited number of LA's.

6.6 Merging the Results of the Rankers

A crucial issue is to determine the relative weight of each ranker in the final phase of merging the results of the various ranker. Tackling this issue requires extensive experimentation with the system. So far, only a rudimentary merger has been implemented and it is based on the simple idea of merging the results by lexicographically sorting the scores of the five rankers. The relative positions of the five rankers in the lexicographic sort is given in a configuration file and can be easily modified by the user through a browser.

We have experimented with different orders of the rankers; in all of them, the word-number ranker was first and idf ranker was second. Results were produced for the following three orders of the rankers:

- Word Number, Idf, Proximity, Similarity, Tf-Idf.
- Word Number, Idf, Similarity, Proximity, Tf-Idf.
- Word Number, Idf, Tf-Idf, Proximity, Similarity.

We always chose word number and idf to be the first and second rankers, since early experiments with the system indicated that it gave the best results. The proximity ranker, the similarity ranker and the tf-idf ranker were essentially used to tune the ranking of the first two rankers.

The following two restrictions were applied to the creation of the XML file that contains the final ranking of the fragments. First, the final result is limited to 1500 fragments. Secondly, at most 5 fragments from any single document could appear in the final result. These limitations could be easily modified by the user.

7. CONCLUSION AND FUTURE WORK

The main contribution of our work is a design of an extensible system that is capable of combining different types of rankers in a manner that takes into account both the structure and the content of the documents. Traditional as well as new information-retrieval techniques can be incorporated into our system, and the ranking score of each technique can be easily modified to include the weights assigned to tags. Our system is also extensible in the sense that it can be easily adapted to changes in the formal syntax of titles, due to the implementation of the topic processor by means of XSL.

Two major issues remain for future work. One is improving the efficiency of the system. The second is improving the quality (i.e., recall and precision) of the results. This would necessitate extensive experimentation with the current rankers as well as new ones. In particular, we plan to modify the merger so that it would use a single formula to aggregate the scores of the various rankers, rather than sorting those scores lexicographically. Towards this end, further experimentation is needed in order to find the optimal weight of each ranker relative to the other rankers.

Acknowledgment

This research was supported by The Israel Science Foundation (Grant 96/01).

8. REFERENCES

[1] http://www.google.com.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press, 1999.

[3] G. Bahlotia. Keyword searching and browsing in databases using banks. In *ICDE Conference*, 2002.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[5] D. Carmel, E. Farchi, Y. Petrushka, and A. Soffer. Automatic query refinement using lexical affinities with maximal information gain. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, 283-290*, Tampere, Finland, June 2002.

[6] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic search engine for XML. In *Proc. 2003 VLDB International Conference on Very Large Databases*, Berlin (Germany), 2003.

[7] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *Proc. 2003 ACM SIGMOD International Conference on Management of Data*, San Diego (California), June 2003.

[8] Y. Jing and W. Croft. An association thesaurus for information retrieval. In *Proc. of RIAO-94, 4th International Conference "Recherche d'Information assistee par Ordinateur", 146-160*, New York (NY), 1994.

[9] G. Kazai, M. Lalmas, and S. Malik. Inex'03 guidelines for topic development, May 2003.

[10] Y. Maarek and F. Smadja. Full text indexing based on lexical relations: An application: Software libraries. In *Proceedings of the 12th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 189-206*, Cambridge, MA, June 1989.

[11] W. Martin, B. Al, and P. van Sterkenburg. On the processing of a text corpus: from textual data to lexicographical information. In *Lexicography: Principles and Practice, Ed. R.R.K Hartman: Principles and Practice, Ed. R.R.K Hartman, Applied Language Studies Series, Academic Press*, London, 1983.

[12] M. Porter. An algorithm for suffix stripping. In *Program*, chapter 14(3), pages 130–137. 1980.

[13] F. D. Saussure. *Cours de Linguistique generale, Quatrieme edition*. Librairie Payot, Paris, France, 1949.

The TIJAH XML-IR system at INEX 2003 (DRAFT)

J.A. List[1] V. Mihajlovic[2] A.P. de Vries[1] G. Ramírez[1]
D. Hiemstra[2]

[1]CWI
P.O. Box 94079
1090GB Amsterdam
The Netherlands
{jalist, arjen, georgina}@cwi.nl

[2]CTIT
P.O. Box 217
7500 AE Enschede
The Netherlands
{vojkan,hiemstra}@cs.utwente.nl

ABSTRACT

This paper discusses our participation in INEX (the Initiative for the Evaluation of XML Retrieval) using the TIJAH XML-IR system. TIJAH's system design follows a 'standard' layered database architecture, carefully separating the conceptual, logical and physical levels. At the conceptual level, we classify the INEX XPath-based query expressions into three different query patterns. For each pattern, we present its mapping into a query execution strategy. The logical layer exploits *probabilistic region algebra* as the basis for query processing. We discuss the region operators used to select and manipulate XML document components. The logical algebra expressions are mapped into efficient relational algebra expressions over a physical representation of the XML document collection using the 'pre-post numbering scheme'. The paper concludes with a preliminary analysis of the evaluation results of the submitted runs.

1. INTRODUCTION

This paper describes our research for INEX 2003 (the Initiative for the Evaluation of XML Retrieval). We participated with the TIJAH XML-IR retrieval system, a research prototype built on top of the MonetDB database kernel [1]. Key feature of the TIJAH system is its layered design, following the basic system architecture of relational database management systems.

Traditional information retrieval systems represent a document as a 'bag-of-words'. Inverted file structures provide the basis for implementing a retrieval system for such 'flat' documents. In the case of structured documents however, we think designing the retrieval system following 'the database approach' is best to keep the more complex data representation manageable.

The main characteristic of the database approach is a strong separation between conceptual, logical and physical levels, and the usage of different data models and query languages at each of those levels [18]. In relational database systems, a significant benefit of this *data abstraction* (through the separation between the levels in database design) is to enable query optimization. A SQL query (a 'calculus expression') at the conceptual level is first translated into relational algebra. The algebraic version used at the logical level is then rewritten by the query optimizer into an efficient physical query plan. The physical algebra exploits techniques like hashing and sorting to improve efficiency [8].

For XML-IR systems, following this separation in layers gives another, additional advantage: by choosing the appropriate level of abstraction for the logical level, the development of probabilistic techniques handling structural information is simplified, and kept orthogonal to the rest of the system design. Section 3 details our approach, based on a probabilistic extension of text region algebras.

The paper is organized along the layers of the TIJAH system design. The following Section describes the query language used at the conceptual level, identifies three patterns in the INEX topic set, and explains how the language modelling approach to information retrieval is used for the `about` operator. Section 3 presents a probabilistic region algebra for expressing the three query patterns. Section 4 explains how the algebraic expressions are mapped into efficient relational algebra expressions over a physical representation of the XML document collection using the 'pre-post numbering scheme'. We conclude with a discussion of the experiments performed with our approach for the three INEX search tasks.

2. CONCEPTUAL LEVEL

For the conceptual level, we used the INEX query language, as proposed by the INEX Initiative in 2002. The INEX query language extends XPath with a special *about*-function, ranking XML elements by their estimated relevance to a textual query. As such, the invocation of the *about* function can be regarded as the instantiation of a retrieval model.

The retrieval model used for the *about*-function is essentially the same as that used at INEX 2002 [12, 14]. We calculate the probability of complete relevance of a document component assuming independence between the probability of relevance on exhaustivity and the probability of relevance on specificity.

The probability of relevance on exhaustivity, $P(R_E)$, is estimated using the language modelling approach to information retrieval [11]. Instead of document frequency, we have used collection frequencies for the background model. The probability of relevance on specificity, $P(R_S)$, is assumed to be directly related to the component length (following a log-normal distribution). Its steep slope at the start discounts the likelihood that very short document components are relevant. Its long tail reflects that we do not expect long document components to be focused on the topic of request either.

The language model as used by our system disregards structure within a document component, i.e., the model treats a document component as a 'flat-text' document. This model property, and an informal inspection of the INEX 2003 topic list, led us to use only a subset of possible location step axes within an *about* function call; we only used the *descendant-or-self::qname* location step axis. Allowing other axes, like *sibling::qname* or *following::qname* requires correct probabilistic modeling for estimating probabilities in the language model, which our model did not offer at the time of evaluation.

Table 1: SCAS and VCAS pattern set. Note that *xp*, *xp2*, *axp*, *axp1* and *axp2* are location steps, and *'t/p'* denotes any set of *terms* or *phrases* to perform the search.

Pattern	Pattern definition
P_1	xp[about(axp, 't/p')]
P_2	xp[about(axp1, 't1/p1') AND about(axp2, 't2/p2')]
	xp[about(axp1, 't1/p1') OR about(axp2, 't2/p2')]
P_3	xp[about(axp1, 't1/p1')]/xp2[about(axp2, 't2/p2')]
	xp[about(axp1, 't1/p1')]//xp2[about(axp2, 't2/p2')]

Since we did not have an automatic query processing facility, we processed the queries manually but in a mechanic fashion. Processing the INEX query patterns takes place in two steps: 1) classify the query into (a sequence of) three basic *query patterns* (shown in Table 1), and, 2) create a query plan to process the queries. The query patterns are visualized in Figure 1.

The basic pattern for all XPath based queries is the single *location step*, as defined in [7], augmented with an *about* function call (pattern P_1 in Table 1). When referring to e.g., *xp*, we refer to the nodeset representing the location step *xp*; in other words, a path leading to a certain location (or node) in the XML syntax tree. The first query pattern consists of one location step to identify the nodes to be retrieved, ranked by an about expression over a nodeset reached by a second location step. The two other (more complex) patterns P_2 and P_3 are essentially multiple interrelated instances of the basic pattern P_1. The XPath location steps may also apply (Boolean) predicate filters, e.g. selecting nodes with a particular value range for `yr`.

3. LOGICAL LEVEL

The logical level is based on a probabilistic region algebra. Region algebra was introduced by Burkowski [2], Clarke et al. [3], and Tova and Milo [4]. The aim of the earliest text region algebra approaches has been to enable structured text search. Later, it has been applied to related tasks as well, including search on nested text regions [13], processing of structured text [17], and ranked retrieval from structured text documents [15].

The basic idea behind region algebra approaches is the representation of text documents as a set of 'extents', where each extent is defined by its starting and end position. The application of the idea of text extents to XML documents is straightforward. If we regard each XML document instance

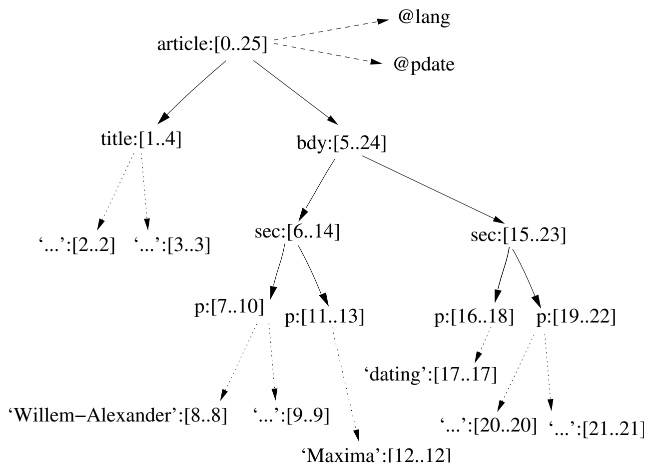


Figure 2: Example XML syntax tree with start and endpoint assignment.

as a linearized string or a set of *tokens* (including the document text itself), each component can then be considered as a text region or a contiguous subset of the entire linearized string. Therefore, a text region a can be identified by its starting point s_a and ending point e_a within the entire linearized string. Figure 2 visualizes an example XML document (as a syntax tree) with the start point and end point numbering for the nodes or regions in the tree. As an example, the *bdy*-region corresponds to (closed) interval [5..24].

Let us introduce the basic set of region operators. We use capital letters (A, B, C) to denote the region sets, and their corresponding non-capitals to denote regions in these region sets (a, b, c). The operators take region sets as input and give a result which is again a region set. The definition of region operators is given in Table 2. Interval operator $I(t)$ returns the region set representing the occurrences of term t as a content word in the XML document; note that it gives a result set in which $s_a = e_a$ for every region, assuming t is a single term and not a phrase. Location operator $L(xp)$ denotes the sequential application of XPath location steps, i.e., axis- and node-tests (a definition of axis- and node-tests can be found in [16]). Optionally, location step operator L also processes predicate tests on node or attribute values specified in the XPath expression.

Table 2: Region Algebra Operators.

Operator	Operator definition
$I(t)$	$\{a \| s_a, e_a$ are pre and post index of term t$\}$
$L(xp)$	$C = XPath(xp)$
$A \rhd B$	$\{a \| a \in A \wedge b \in B \wedge s_a \leq s_b \wedge e_a \geq e_b\}$
$A \lhd B$	$\{a \| a \in A \wedge b \in B \wedge s_a \geq s_b \wedge e_a \leq e_b\}$
$A \bigtriangleup B$	$\{c \| c \in A \wedge c \in B\}$
$A \bigtriangledown B$	$\{c \| c \in A \vee c \in B\}$

Table 3 expresses the patterns identified in the previous Section using region algebra operators (ignoring ranking for now). Pattern 1 distinguishes between term (t) and phrase expressions ($p = \{t_1, t_2, ..., t_n\}$). Patterns 2 and 3 are rewritten into several interrelated instances of pattern 1. Table 4

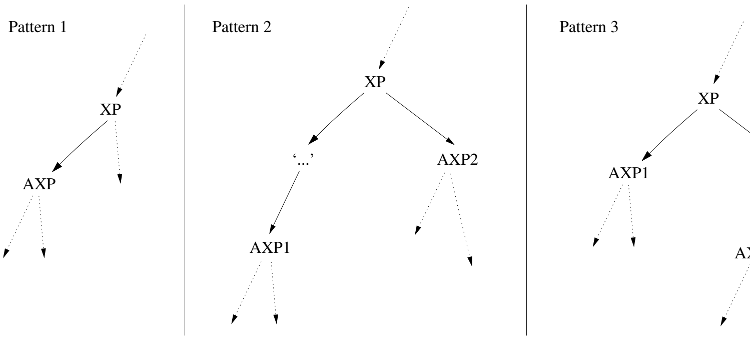


Figure 1: Example instances of the three defined patterns.

Table 3: Pattern definitions based on pure region algebra operators.

Pattern	Algebraic expression
$P_1(xp, axp)$	$L(xp) \rhd (L(axp) \rhd I(t))$ $L(xp) \rhd ((L(axp) \rhd I(t_1)) \vartriangle (L(axp) \rhd I(t_2)) \vartriangle ... \vartriangle (L(axp) \rhd I(t_n)))$
$P_2(xp, axp1, axp2)$	$P_1(xp, axp1) \vartriangle P_1(xp, axp2)$ $P_1(xp, axp1) \triangledown P_1(xp, axp2)$
$P_3(xp1, xp2, axp1, axp2)$	$P_1(xp2, axp2) \lhd P_1(xp1, axp1)$

introduces a probabilistic extension of the pure region algebra operators. In order to introduce ranking, we extend the notion of region with its *relevance score*; i.e., every region a has an associated relevance score p_a. In cases where pure region algebra operators are used, the value of the introduced relevance score is equal to a predefined default value (e.g., $p_a = 1$) for each resulting region in a region set.

Table 5 gives the probabilistic region algebra expressions corresponding to the INEX query patterns identified before. The *tp1* is used to denote *'t1/p1'* or the combination of *'t1/p1'* and *'t2/p2'* (the choice between these options is made at the conceptual level). Similarly, *tp2* is either *'t2/p2'* or a combination of *'t2/p2'* and *'t1/p1'*.

Expressing query plans using the operators given in Table 4 preserves *data independence* between the logical and the physical level of a database. Similarly, these operators enable the separation between the structural query processing and the underlying probabilistic model used for ranked retrieval: a design property termed *content independence* in [6]. The instantiation of these probabilistic operators is implementation dependent and does not influence the global system architecture. This gives us the opportunity to change the probabilistic model used or to modify the existing model while keeping the system framework, creating the opportunity to compare different probabilistic models with minimal implementation effort.

4. PHYSICAL LEVEL

The physical level of the TIJAH system relies on the MonetDB binary relational database kernel [1]. This Section details implementation and execution strategy for each of the patterns.

The text extents used at the logical level are represented by XML text regions at the physical level, and encoded using a preorder/postorder tree encoding scheme, following [9, 10]. The XML text regions are stored as three-tuples $\{ s_i, e_i, t_i \}$, where:

- s_i and e_i represent the start and end positions of XML region i;
- t_i is the (XML) tag of each region.

The set of all XML region tuples is named the *node index* $\mathcal{N}$. Index terms present in the XML documents are stored in a separate relation called the *word index* $\mathcal{W}$. Index terms are considered text regions as well, but physically the term identifier is re-used as both start and end position to reduce memory usage. The physical layer has been extended with the text region operators shown in Table 6. Boolean predicate filters are always applied first. For further details on this indexing scheme, refer to [5, 14].

4.1 Pattern 1

Pattern 1 for the VCAS scenario Processing pattern 1 in Table 1 requires two basic steps: relating nodesets *xp* and *axp* to each other, and processing the *about* operator. Nodesets *xp* and *axp* must have a parent - descendant[1] structural relationship. So, the pattern is processed as follows (visualized in Figure 3):

[1]Parent - child relationships are considered a specific variant of parent - descendant relationships.

Table 6: Text region operators at the physical level.

Operator	Definition
$a \supset b$	$true \iff s_b > s_a \wedge e_b < e_a$
$a \subset b$	$true \iff s_a > s_b \wedge e_a < e_b$
$A \bowtie_{\supset} B$	$\{(s_a, s_b)\vert\ a \leftarrow A,\ b \leftarrow B,\ a \supset b\}$
$A \bowtie_{\subset} B$	$\{(s_a, s_b)\vert\ a \leftarrow A,\ b \leftarrow B,\ a \subset b\}$

Table 4: Probabilistic region algebra operators. Note that the "ranked containing" and "ranked and" operators are used to define the *about*-function.

Operator	Operator description	Operator usage examples
$A \triangleright B$	ranked containing (based on LM)	$L(axp) \triangleright I(t)$
$A \trianglerighteq B$	average containing	$L(xp) \trianglerighteq (L(axp) \triangleright I(t))$
$A \Delta B$	ranked and (based on LM)	$L(xp) \trianglerighteq ((L(axp) \triangleright I(t_1))\Delta(L(axp) \triangleright I(t_2)))$
$A \trianglelefteq B$	average contained	$(L(xp1) \trianglerighteq (L(axp1) \triangleright I(t_1))) \trianglelefteq (L(xp2) \trianglerighteq (L(axp2) \triangleright I(t_2)))$
$A \bigtriangleup B$	complex and	$(L(xp) \trianglerighteq (L(axp1) \triangleright I(t_1))) \bigtriangleup (L(xp) \trianglerighteq (L(axp2) \triangleright I(t_2)))$
$A \bigtriangledown B$	complex or	$(L(xp) \trianglerighteq (L(axp1) \triangleright I(t_1))) \bigtriangledown (L(xp) \trianglerighteq (L(axp2) \triangleright I(t_2)))$

Table 5: Pattern definitions based on probabilistic region algebra operators.

Pattern	Algebraic expression
$P_1(xp, axp, t)$ $P_1(xp, axp, p)$	$L(xp) \trianglerighteq (L(axp) \triangleright I(t))$ $L(xp) \trianglerighteq ((L(axp) \triangleright I(t_1))\Delta(L(axp) \triangleright I(t_2))\Delta...\Delta(L(axp) \triangleright I(t_n)))$
$P_2(xp, axp1, axp2, tp1, tp2)$	$P_1(xp, axp1, tp1) \bigtriangleup P_1(xp, axp2, tp2)$ $P_1(xp, axp1, tp1) \bigtriangledown P_1(xp, axp2, tp2)$
$P_3(xp1, xp2, axp1, axp2, tp1, tp2)$	$P_1(xp2, axp2, tp2) \trianglelefteq P_1(xp1, axp1, tp1)$

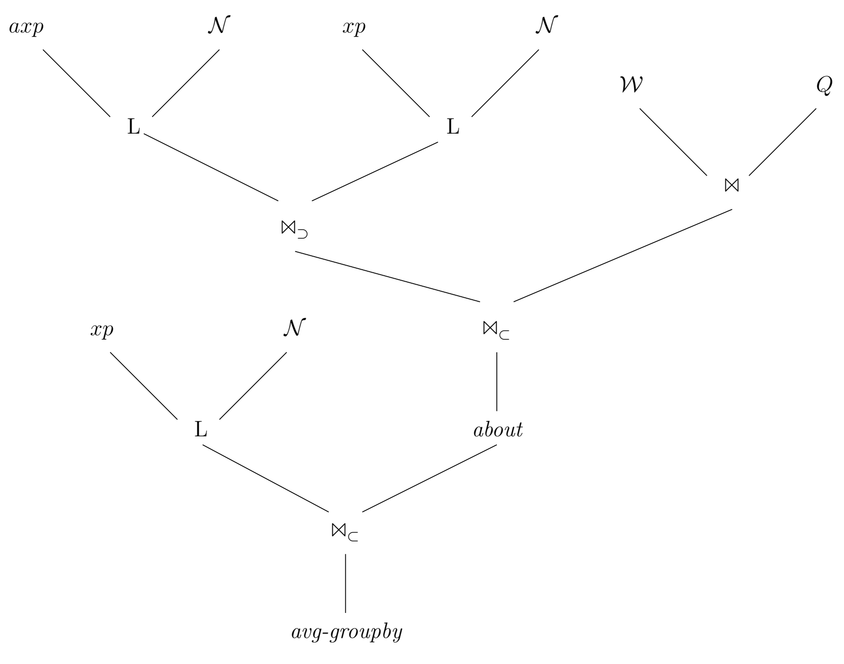


Figure 3: Physical query plan for pattern 1.

- Determine the correct *axp* nodeset for ranking. On the physical level, this is done by executing a containment join between the nodesets *xp* and *axp*: $axp \bowtie_{\subset} xp$. The result of this containment join is *cxp* or the set of those nodes of *axp* which are contained within nodes in *xp*;
- Perform the *about* operation on the nodes in *cxp* (the combination of $\triangleright$ and Δ operators on the logical level);
- Return the ranking for the *xp* nodeset, based on the rankings of the nodes present in *cxp*. Note that it is possible that the ranking returns a ranking for multiple *axp* descendant nodes for a single *xp* node (e.g., multiple sections within an article). In that case, we take the **average** as the final score for the *xp* node in question. This step is the physical equivalent of the logical $\triangleright$ (one descendant of the type of *axp*) or logical $\trianglerighteq$ (multiple descendants of the type of *axp*) operator.

Pattern 1 for the SCAS scenario The processing of pattern 1 for the SCAS scenario does not differ from the processing performed for the VCAS scenario. The containment join will automatically remove those *xp* nodes not containing one or more *axp* nodes. This ensures only the 'correct' *axp* nodes, those within a node from the *xp* nodeset, will be ranked.

4.2 Pattern 2

Pattern 2 for the VCAS scenario For the processing of pattern 2 for the VCAS scenario, we assume that conjunctions and disjunctions specified in the query relate to the structure, and never to the query terms. In case nodesets *axp1* and *axp2* are equal, the pattern is rewritten to a pattern 1. If the nodesets *axp1* and *axp2* are not equal, it is possible these nodesets represent completely different parts of the (sub)tree below *xp*, as depicted in Figure 1. In path-based terms, if the (sub)tree starting at *xp* does not contain both paths *axp1* and *axp2*, that *xp* tree cannot be relevant in the strict query scenario.

However, for a more vague query scenario, we argue that the absence of a descendant node does not render the requested (ancestor) target node irrelevant completely. Consider the following expression:

```
/article[
    about(./abstract, 'information retrieval')
            AND about(.//section, 'XML data')
    ]
```

If an article contains no abstract, but it does score on 'XML data' in one or more of the sections, the question is whether the article is completely irrelevant. For a vague retrieval scenario this might not be the case. Therefore, we decided to process these expression types as follows. We split up the expression into a series of pattern 1 expressions, and combine the results of the individual pattern 1 executions. The example above is split up into the following two pattern 1 expressions:

```
- /article[about(./abs, 'information retrieval XML data')]
- /article[about(.//sec, 'information retrieval XML data')]
```

Both subpatterns are processed as pattern 1. The two resulting nodesets need to be combined for a final ranking. An intuitive combination function for the $\triangle$ operator is taking the **minimum** of the (non-zero) descendant scores, and for the $\triangledown$ operator the **maximum**. Note that, alternatively, a more formal probabilistic choice would be to use product and sum instead of minimum and maximum; whether this yields better results is an open question for further research.

Pattern 2 for the SCAS scenario For the SCAS scenario, all of the descendant nodes present in *axp1* and *axp2* need to be present in the context of an *xp* node. In path-based terms: if the path *xp* does not contain both a path *axp1* and a path *axp2*, the path *xp* cannot be relevant. We filter out those *xp* paths, not containing both the *axp1* and *axp2* paths. This additional filtering step and the choice of operator to implement the complex 'and' ($\triangle$) and 'or' ($\triangledown$) operators define together the difference between strict and vague scenarios.

4.3 Pattern 3

Pattern 3 for the VCAS scenario Pattern 3 can be processed like pattern 2, except for the fact that the target element now lies deeper in the tree. We process this pattern by first splitting it up into multiple instances of pattern 1:

```
- xp[about(axp1, 't1/p1 t2/p2')]
- xp/xp2[about(axp2, 't1/p1 t2/p2')]
```

The pattern 1 execution already provides for aggregation of scores of a set of nodes of the same type, within a target element. The question remains however how to combine the scores of the nodes present in nodesets */xp/axp1* and */xp/xp2/axp2*. Like before, these nodesets can represent nodes in completely different parts of the (sub)tree.

Based on the observation that the user explicitly asks for the nodes present in the */xp/xp2* nodeset, we decided to use the rankings of those nodes as the final rankings. The first about-predicate reduces nodeset *xp* to those nodes for which a path *axp1* exists. For the vague scenario however, we argue that absence or presence of *axp1* does not really influence target element relevance (similar to pattern 2 in subsection 4.2).

Summarizing, the first *about*-predicate in the pattern mentioned at the start of this subsection is dropped, rewriting the resulting pattern to a pattern 1 instance:

```
/xp/xp2[about(axp2, 't1/p1 t2/p2')]
```

This results in the following execution strategy for pattern 3 under the VCAS scenario: remove all *about*-predicates from all location steps, except for the *about*-predicate on the target element.

Pattern 3 for the SCAS scenario The processing of pattern 3 for the SCAS scenario is stricter in the sense that we can not simply drop intermediate *about*-predicates, as we did for the VCAS scenario. The general procedure consists of 1) splitting up the pattern into separate location steps and

2) structural correlation of the resulting nodesets of each location step. The target elements are ranked by their corresponding *about*-predicate only; thus, ignoring the scores produced for the other *about*-clauses in the query. Like in pattern 1, the target element can have multiple descendants; in that case, the descendants' scores are averaged to produce the target element scores.

As an example, consider the following expression:

```
/article[about(./abstract, 't1/p1')]
        //section[about(./header, 't2/p2')]
        //p[about(., 't3/p3')]
```

We first split up the above expression into:

```
- /article[(about(./abstract, 't1/p1 t2/p2 t3/p3')]
- //section[about(./header, 't1/p1 t2/p2 t3/p3')]
- //p[about(., 't1/p1 t2/p2 t3/p3')]
```

All of the patterns above produce intermediate result nodesets that have to be structurally correlated to each other. We can choose to perform a top-down correlation sequence, or a bottom-up correlation sequence consisting of containment joins. The choice between a top-down or bottom-up sequence can be an optimization decision, made at runtime by the retrieval system. For example, if a collection contains many paragraph elements, not contained within article elements, the system might decide to limit the amount of unnecessary executed *about*-predicates by choosing a top-down approach. In the current implementation, the patterns are always processed top-down.

5. EXPERIMENTS

For the content only (CO) topics, we designed three experimentation scenarios. The first scenario represents the baseline scenario of 'flat-document' retrieval, i.e. retrieval of documents which possess no structure. After examination of the document collection, we decided to perform retrieval of article-components. The second scenario regarded all subtrees in the collection as separate documents. For the third scenario we re-used the result sets of the second run and used a log-normal distribution to model the quantity dimension. To penalize the retrieval of extremely long document components (this in contrast with the language model that assigns a higher probability to longer documents), as well as extremely short document components, we set the mean at 2516. Experiments for INEX 2002 showed that 2516 words was the average document component length of relevant document components according to the strict evaluation function used in INEX 2002. Table 7 gives a summary of our experimentation scenarios.

For both the SCAS (strict content-and-structure) and VCAS (vague content-and-structure), we submitted one run each (not mentioned in Table 7); the topics executed according to the conceptual, logical and physical SCAS and VCAS pattern rulesets as detailed in the previous Sections. The recall-precision graphs of our submitted CO runs are presented in Figures 4a–f and Figures 5a–f (for the overlapping metric). We have included both the strict and generalized evaluations.

Table 7: CO experimentation scenarios; note that we used a length of 2516 as preferred component length in scenario 3. The experiments for INEX 2002 showed 2516 was the average document component length of relevant components according to the strict evaluation function used in INEX 2002.

Scenario	Retr. Unit	Dimension(s)
V_1^{CO}	$\{tr('article')\}$	*topicality*
V_2^{CO}	$\{tr('*')\}$	*topicality*
V_3^{CO}	$\{tr('*')\}$	*top., quant.(2516)*

6. CONCLUSIONS AND FUTURE WORK

Our participation in INEX can be summed up as an exercise in applying current and state of the art information retrieval technology to a structured document collection. We described a relatively straightforward approach to simplify the implementation of retrieval models that combine structural and content properties. We hope to take advantage of this flexibility to a larger extend in our future research, as the current approach to retrieval has only used a small proportion of all the structural information present in XML documents. Other research includes more extensive experimentation in the area of relevance feedback, and develop a different normalization mechanism to remove the bias of the language model on short components.

7. REFERENCES

[1] P. Boncz. *Monet: a Next Generation Database Kernel for Query Intensive Applications.* PhD thesis, CWI, 2002.

[2] F.J. Burkowski. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Texts. In *Proceedings of the 15th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–125, 1992.

[3] C.L.A. Clarke, G.V. Cormack, and F.J. Burkowski. An Algebra for Structured Text Search and a Framework for its Implementation. *The Computer Journal*, 38(1):43–56, 1995.

[4] M. Consens and T. Milo. Algebras for Querying Text Regions. In *Proceedings of the ACM Conference on Principles of Distributed Systems*, pages 11–22, 1995.

[5] A. P. de Vries, J. A. List, and H. E. Blok. The Multi-Model DBMS Architecture and XML Information Retrieval. In H. M. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors, *Intelligent Search on XML*, volume 2818 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI), Springer-Verlag*, pages 179–192. Springer-Verlag, Berlin, New York, etc., August 2003.

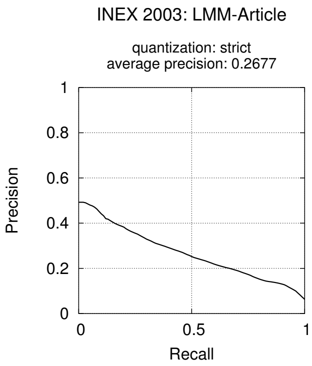
[6] A.P. de Vries. Content independence in multimedia databases. *Journal of the American Society for Information Science and Technology*, 52(11):954–960, September 2001.

[7] M.Fernández et al. XML Path Language (XPath 2.0). Technical report, W3C, 2003.

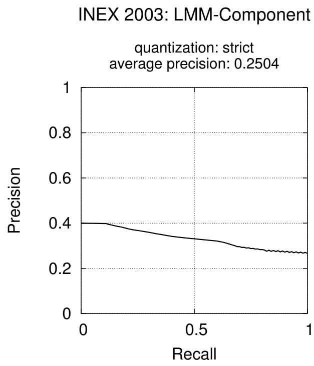
[8] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[9] T. Grust. Accelerating XPath Location Steps. In *Proceedings of the 21st ACM SIGMOD International Conference on Management of Data*, pages 109–120, 2002.

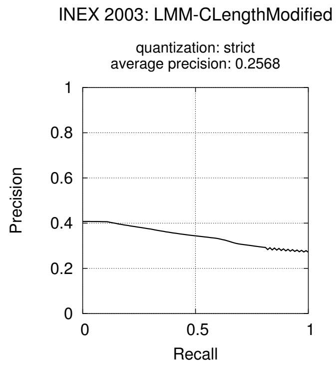
[10] Torsten Grust and Maurice van Keulen. Tree Awareness for Relational DBMS Kernels: Staircase Join. In H. M.



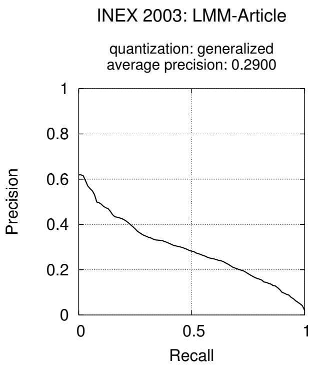
(a) Scenario 1 CO (strict)



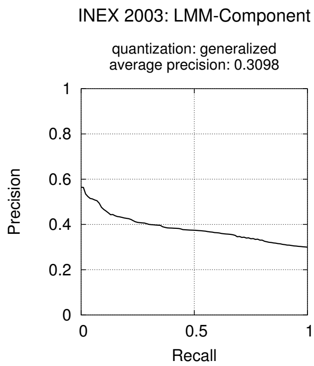
(b) Scenario 2 CO (strict)



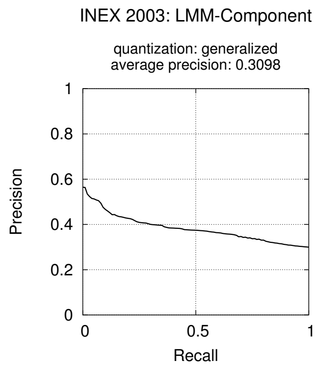
(c) Scenario 3 CO (strict)



(d) Scenario 1 CO (generalized)



(e) Scenario 2 CO (generalized)



(f) Scenario 3 CO (generalized)

Figure 4: CO recall-precision graphs for the simple metric (top row: strict evaluation, bottom row: generalized evaluation).

Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors, *Intelligent Search on XML*, volume 2818 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 179–192. Springer-Verlag, Berlin, New York, etc., August 2003.

[11] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, Twente, The Netherlands, 2000.

[12] D. Hiemstra. A database approach to content-based XML retrieval. In *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval*, 2002.

[13] J. Jaakkola and P. Kilpelainen. Nested Text-Region Algebra. Technical Report C-1999-2, Department of Computer Science, University of Helsinki, 1999.

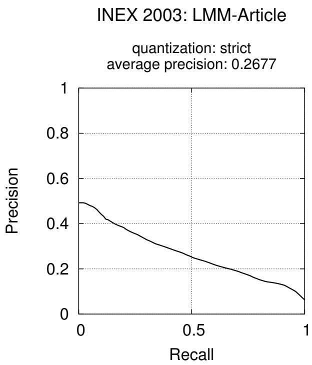
[14] J.A. List and A.P. de Vries. CWI at INEX 2002. In *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval*, 2002.

[15] Katsuya Masuda. A Ranking Model of Proximal and Structural Text Retrieval Based on Region Algebra. In *Proceedings of the ACL-2003 Student Research Workshop*, 2003.

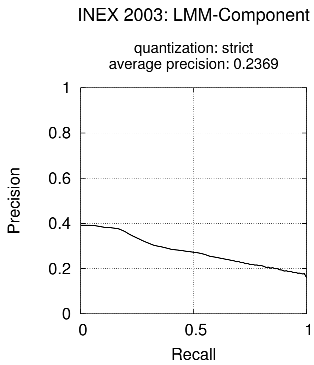
[16] V. Mihajlovic, D. Hiemstra, and P. Apers. On Region Algebras, XML Databases, and Information Retrieval. In *Proceedings of the 4th Dutch-Belgian Information Retrieval Workshop, to apear*, 2003.

[17] R.C. Miller. *Light-Weight Structured Text Processing*. PhD thesis, Computer Science Department, Carnegie-Mellon University, 2002.

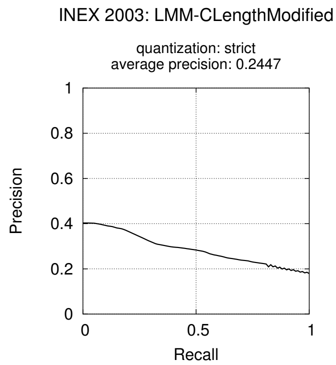
[18] D. Tsichritzis and A. Klug. The ANSI/X3/SPARC DBMS framework report of the study group on database management systems. *Information systems*, 3:173–191, 1978.



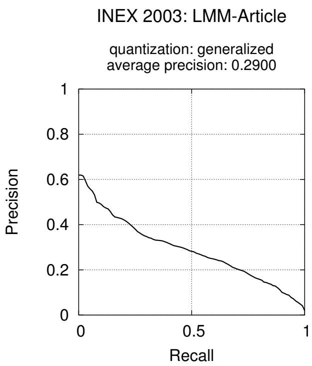
(a) Scenario 1 CO (strict)



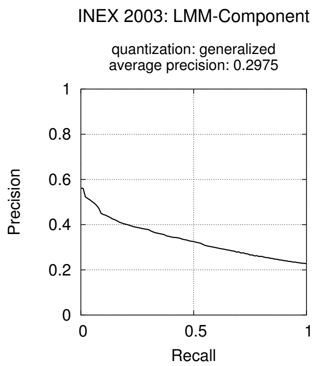
(b) Scenario 2 CO (strict)



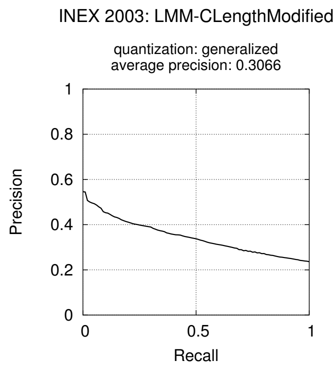
(c) Scenario 3 CO (strict)



(d) Scenario 1 CO (generalized)



(e) Scenario 2 CO (generalized)



(f) Scenario 3 CO (generalized)

Figure 5: CO recall-precision graphs for the overlapping metric (top row: strict evaluation, bottom row: generalized evaluation).

The University of Amsterdam at INEX 2003

Jaap Kamps Maarten de Rijke Börkur Sigurbjörnsson

Language & Inference Technology Group, University of Amsterdam
Nieuwe Achtergracht 166, 1018 WV Amsterdam, The Netherlands
E-mail: {kamps, mdr, borkur}@science.uva.nl

ABSTRACT

This paper describes the INEX 2003 participation of the Language & Inference Technology group of the University of Amsterdam. We participated in all three of the tasks, content-only, strict content-and-structure and vague content-and-structure.

1. INTRODUCTION

One of the recurring issues in XML retrieval is finding the appropriate unit of retrieval. For the content-only (CO) task at INEX 2002, we only submitted runs in which whole articles were the unit of retrieval [3]. Much to our surprise, retrieving articles turned out to be a competitive strategy. In [5] we experimented with going below the article level and returning elements. Our experiments showed that a successful element retrieval approach should be biased toward retrieving large elements. For the content-only task this year our aim was to experiment further with this size bias, in order to try to determine what is the appropriate unit of retrieval.

For the strict content-and-structure (SCAS) task the unit of retrieval is usually explicitly mentioned in the query. Our research question for the content-only task does therefore not carry over to the strict content-and-structure task. At INEX 2002, we experimented with assigning an RSV score to elements satisfying an XPath expression. This year we experiment further with the same idea, although our scoring methods are quite different from those of last year.

The vague content-and-structure (VCAS) task is a new task and we could not base our experiments on previous experience. Since the definition of the task was underspecified, our aim for this task was to try to find out what sort of task this was. We experimented with a content-only approach, strict content-and-structure approach and article retrieval approach.

All of our runs were created using the `FlexIR` retrieval system developed by the Language and Inference Technology group. We use a multinomial language model for the scoring of retrieval results.

The structure of the remainder of this paper is as follows. In Section 2 we describe the setup used in our experiments. In Section 3 we explain the submitted runs for each of the three tasks, CO in 3.1, SCAS in 3.2 and VCAS in 3.3. Results are presented and discussed in Section 4 and in Section 5 we draw initial conclusions from our experiments.

2. EXPERIMENTAL SETUP

2.1 Index

We adopt an IR based approach to XML retrieval. We created our runs using two types of inverted indexes, one for XML articles only and another for all XML elements.

Article index

For the article index, the indexing unit is a whole XML document containing all the terms appearing at any nesting level within the ⟨article⟩ tag. This is thus a traditional inverted index as used for standard document retrieval.

Element index

For the element index, the indexing unit can be any XML element (including ⟨article⟩). For each element, all text nested inside it is indexed. Hence the indexing units overlap (see Figure 1). Text appearing in a particular nested XML element is not only indexed as part of that element, but also as part of all its ancestor elements.

The article index can be viewed as a restricted version of the element index, where only elements with tag-name ⟨article⟩ are indexed.

Both indexes were word-based, no stemming was applied to the documents, but the text was lower-cased and stop-words were removed using the stop-word list that comes with the English version on the Snowball stemmer [8]. Despite the positive effect of morphological normalization reported in [3], we decided to go for a word-based approach. Some of our experiments have indicated that high precision settings are desirable for XML element retrieval [4]. Word-based approaches have proved very suitable for achieving high precision.

2.2 Query processing

Two different topic formats are used, see Figure 2 for one of the CO topics, and Figure 3 for one of the CAS topics. Our queries were created using only the terms in the ⟨title⟩ and ⟨description⟩ parts of the topics. Terms in the ⟨keywords⟩ part of the topics have proved to significantly improve retrieval effectiveness [4]. The keywords, which are used to assist during the assessment stage, are often based on human inspection of relevant documents during the topic creation. Using them would have meant a violation of the

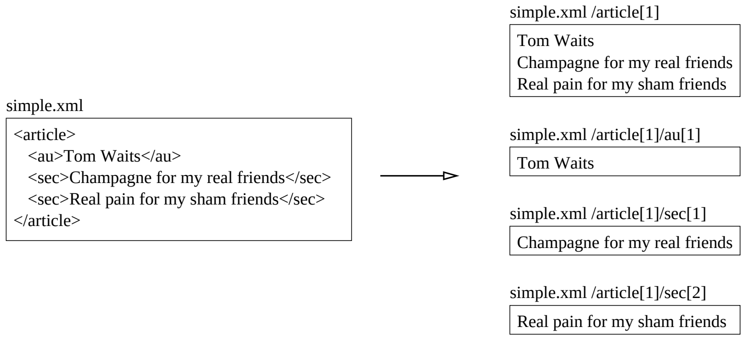


Figure 1: Simplified figure of how XML documents are split up into overlapping indexing units

assumptions underlying a fully automatic run, so we decided not to use them. Our system does not support +, - or phrases in queries. Words and phrases bound by a minus were removed, together with the minus-sign. Plus-signs and quotes were simply removed.

Like the index, the queries were word-based, no stemming was applied but the text was lower-cased and stopwords were removed.

For some of our runs we used queries expanded by blind feedback. We considered it safer to perform the blind feedback against the article index since we do not know how the overlapping nature of the element index affects the statistics used in the feedback procedure. We used a variant of Rocchio feedback [6], where the top 10 documents were considered relevant; the top 501-1000 were considered non-relevant; and up to 20 terms were added to the initial topic. Terms appearing in more that 450 documents were not considered as feedback terms. An example of an expanded query can be seen in Figure 2*c*.

Task specific query handling will be further described as part of the run descriptions in the following section.

2.3 Retrieval model

All our runs use a multinomial language model with Jelinek-Mercer smoothing [2]. We estimate a language model for each of the elements. The elements are then ranked according to the likelihood of the query, given the estimated language model for the element. To account for data sparseness we estimate the element language model by a linear interpolation of two language models, one for the element and another for the collection:

$$P(E|Q) = P(E) \cdot \prod_{i=1}^{k} (\lambda \cdot P_{mle}(t_i|E) + (1-\lambda) \cdot P_{mle}(t_i|C)), \quad (1)$$

where Q is a query made out of the terms $t_1, \ldots, t_k$; E is a language model of an element; C is a language model of the collection; and λ is the interpolation factor (smoothing parameter). We estimate the language models, $P_{mle}(\cdot|\cdot)$ using maximum likelihood estimation. For the collection model we use element frequencies. Assuming a uniform prior probability of elements being relevant, our basic scoring formula for an element E and a query $Q = (t_1, \ldots, t_k)$ is therefore

$$s(E,Q) = \sum_{i=1}^{k} \log\left(1 + \frac{\lambda \cdot \mathrm{tf}(t_i,E) \cdot (\sum_t \mathrm{df}(t))}{(1-\lambda) \cdot \mathrm{df}(t_i) \cdot (\sum_t \mathrm{tf}(t,E))}\right), \quad (2)$$

where $\mathrm{tf}(t,E)$ is the frequency of term t in element E, $\mathrm{df}(t)$ is the element frequency of term t and λ is the smoothing parameter. In most cases we base the probability $P(E)$ on the element length. That is, we add a length prior to the score:

$$\mathrm{lp}(E) = \log\left(\sum_t \mathrm{tf}(t,E)\right). \quad (3)$$

For an exact description of how we apply this length prior, see individual run descriptions in Section 3.

The smoothing parameter λ played a crucial role in our submissions. In [4] we reported on the effect of smoothing on the unit of retrieval. The results obtained there suggested that there was a correlation between the value of the smoothing parameter and the size of the retrieved elements. The average size of retrieved elements increases dramatically as less smoothing (a higher value for the smoothing parameter λ) was applied. Further descriptions on how we tried to exploit this size-smoothing relation are provided in the individual run descriptions.

Smoothing is not the only method applicable to eliminate the small elements from the retrieval set. One can also simply discard the small elements when building the index. Elements containing text that is shorter than a certain cut-off value can be ignored when the index is built. In some of our runs we imitated such index building by restricting our view of the element index to a such a cut-off version. Further details will be provided in the description of individual runs in the next section.

3. RUNS

3.1 Content-Only task

In [5] we tried to answer the question of what is the appropriate unit of retrieval for XML information retrieval. A general conclusion was that users have a bias toward large elements. With our runs for the content-only task we pursued this issue further.

We create a language model for each XML-element. As described in the previous section, since small elements do not provide a large sample space, we get very sparse statistics. We therefore smooth our statistics by combining the element language model with a language model for the whole collection. Zhai and Lafferty [10] argue that bigger documents require less smoothing than smaller documents. A similar effect was witnessed in [4], where less smooth language models retrieved larger elements than more smooth language models.

Increasing the value of λ in the language model causes an occurrence of a term to have an increasingly bigger impact. As a result, the elements with more matching terms are favored over elements with fewer matching terms. In the case of our overlapping element index, a high lambda gives us an article biased run, whereas a low lambda introduces a bias toward smaller elements (such as sections and paragraphs).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="103" query_type="CO" ct_no="50">
 <title>UML formal logic</title>
 <description>Find information on the use of formal logics
  to model or reason about UML diagrams.</description>
 <narrative>...</narrative>
 <keywords>...</keywords>
</inex_topic>
```

(a) Original topic

```
.i 103
uml formal logic find information use formal logics model
reason uml diagrams
```

(b) Cleaned query (TD)

```
.i 103
uml formal logic find information use formal logics model
reason uml diagrams booch longman rumbaugh itu jacobson
wiley guards ocl notations omg statecharts formalism
mappings verlag sdl documenting stereotyped semantically
sons saddle
```

(c) Expanded query (TD+blind feedback)

Figure 2: CO Topic 103

In our runs we scored elements by combining evidence from the element itself, $s(e)$, and evidence from the surrounding article $s(d)$, using the scoring formula

$$s_{comb}(e) = \text{lp}(e) + \alpha \cdot s(d) + (1 - \alpha) \cdot s(e) \quad (4)$$

where $s(\cdot)$ is the score function from Equation 2 and $\text{lp}(\cdot)$ is the length prior from Equation 3.

We submitted the following runs for the CO task. Since we wanted to experiment with the element size bias, we wanted to compare two runs, one with considerable smoothing ($\lambda = 0.2$) and another with considerably less smoothing ($\lambda = 0.9$).

UAmsI03-CO-lambda=0.9

In this run we set the smoothing parameter λ to 0.9. This value of λ means that little smoothing was performed, which resulted in a run with a bias toward retrieving large elements such as whole articles.

UAmsI03-CO-lambda=0.2

In this run we set the smoothing parameter λ to 0.2 which means that a considerable amount of smoothing is performed. This resulted in a run with a bias toward retrieving elements such as sections and paragraphs.

UAmsI03-CO-lambda=0.5

Here we went somewhere in between the two extremes above by setting $\lambda = 0.5$. Furthermore, we required elements to be either articles, bodies or nested within the body.

All runs used the same combination value $\alpha = 0.4$ in the scoring formula (4), a value chosen after experimenting with the INEX 2002 collection. Only elements longer than 20 terms were considered. Very short pieces of text are themselves not likely to be very informative. One straightforward way to make sure those short elements are not retrieved, is to remove them from the index. The value for the size threshold was justified by experiments on the INEX 2002 collection.

As described previously, queries were created using the terms from the title and description; they were not stemmed but stop-words were removed (See Figure 2*b*). The queries were expanded using blind feedback (See Figure 2*c*). The parameters for the feedback were based on experiments with the INEX 2002 collection. In our score calculations we used the overlapping element index as a basis for the collection language model. In our combinations of article and element scores we did not do any normalization of scores.

3.2 Strict Content-And-Structure task

For the Strict Content-and-structure (SCAS) task the unit of retrieval is usually coded inside the topics. Our research question for the CO task does therefore not carry directly over to the SCAS task.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="76" query_type="CAS" ct_no="81">
 <title>//article[(./fm//yr='2000' OR
  ./fm//yr='1999') AND about(.,'"intelligent
  transportation system"')]//sec[about(.,
  'automation +vehicle')]</title>
 <description>Automated vehicle applications
  in articles from 1999 or 2000 about intelligent
  transportation systems.</description>
 <narrative>...</narrative>
 <keywords>...</keywords>
</inex_topic>
```

(a) Original topic

```
.i 76
intelligent transportation system automation
vehicle automated vehicle applications in
articles from 1999 or 2000 about intelligent
transportation systems
```

(b) Full content query (TD)

```
.i 76a article
intelligent transportation system
.i 76b sec
automation vehicle
```

(c) Partial content queries(T)

```
//article[about(., "76a")]//sec[about(.,"76b")]
```

(d) Fuzzy structure (T)

```
//article[./fm//yr='2000' or ./fm//yr='1999']//sec
```

(e) Strict structure (T)

Figure 3: CAS Topic 76

The CAS topics have a considerably more complex format than the CO topics (see Figure 3*a* for an example). The description part is the same, but the title has a different format. The CAS title is written in a language which is an extension of a subset of XPath [9]. We can view the title part of the CAS topic as a mixture of path expressions and filters. Our aim with our SCAS runs was to try to cast light on how these expressions and filters could be used to assign scores to elements. All our runs treat the filters in quite a strict fashion; the larger number the number of filters that are satisfied, the higher the ranking of an element. The difference between our three runs lies in the way we decide the ranking of results that satisfy the same number of filters.

More precisely, we consider the topic title of CAS topics to be split into path expressions and filters as follows.

$$\texttt{rootPath}[F_r \cup C_r \cup S_r]\texttt{targetPath}[F_e \cup C_e \cup S_e], \qquad (5)$$

where `rootPath` and `targetPath` are path expressions and F_r, C_r, S_r, F_e, C_e, S_e are sets of filters (to be explained below). The filters in the actual topics were connected with a boolean formula. We ignore this formula and only look at sets of filters. We distinguish between three types of filters.

Element filters (F) F is a set of filters that put content constraints on the current element, as identified by preceding path expression (`rootPath` or `targetPath`). Element filters have the format `about(.,'whatever')`

Nested filters (C) C is a set of filters that put content constraints on elements that are nested within the current element. Nested filters have the format `about(./path, 'whatever')`

Strict filters (S) S is a set of filters of the format `path op value`, where `op` is a comparison operator such as = or >=; and value is a number or a string.

As an example, the title part of Topic 76 in Figure 3*a* can be broken up into path expressions and filters such as:

$\texttt{rootPath} = \texttt{//article}$

$F_r = \{\texttt{about(., `"intelligent transportation system"')}\}$

$C_r = \emptyset$

$S_r = \{\texttt{./fm//yr=`2000', ./fm//yr=`1999'}\}$

$\texttt{targetPath} = \texttt{//sec}$

$F_e = \{\texttt{about(., `automation +vehicle')}$

$C_e = \emptyset$

$S_e = \emptyset$

We calculate the retrieval scores by combining 3 base runs. The base runs consist of an *article run*, a ranked list of articles answering the full content query (Figure 3*b*); an *element run*, a ranked list of target elements answering the full content query (Figure 3*b*); and a *filter run*, a ranked list of elements answering each of the partial content queries (Figure 3*c*). More precisely the base runs were created as follows.

Article run

We created an article run from the element index by filtering away all elements not having the tag-name ⟨**`article`**⟩. We used a value $\lambda = 0.15$ for the smoothing parameter. This is the traditional parameter settings for document retrieval. We used the full content query (Figure 3*b*), expanded using blind feedback. For each query we retrieved a ranked list of 2000 most relevant articles.

Element run

We created an element run in a similar fashion as for the CO task. Additionally, we filtered away all elements that did not have the same tag-name as the target tag-name (the rightmost part of the `targetPath`). For topics where the target was a '*' we considered only elements containing at least 20 terms. We did moderate smoothing by choosing a value of 0.5 for λ. We used the full content queries (Figure 3*b*), expanded using blind feedback. For each query we retrieved an exhaustive ranked list of relevant elements.

Filter run

We created an element run in a similar fashion as for the CO task, but using the partial content queries (Figure 3*c*). No blind feedback was applied to the queries. We filtered away all elements that did not have the same tag-name as the target tag-name of each filter. For filters where the target was a '*' we considered only elements containing at least 20 terms. We did minor smoothing by choosing the value 0.7 for λ. For each query we retrieved an exhaustive ranked list of relevant elements.

For all the base runs the length prior from equation 3 is added to the score.

From the base runs we created three runs which we submitted: one where scores are based on the element run; another where scores are based on the article run; and a third which uses a mixture of the element run, article run and filter run.

UAmsI03-SCAS-ElementScore

The articles appearing in the article run were parsed and their elements that matched any of the element- or nested-filters were kept aside as candidates for the final retrieval set. In other words, we kept aside all elements that matched the title XPath expression, where the about predicate returns the value `true` for precisely the elements that appear in the filter run. The candidate elements were then assigned a score according to the element run. Additionally, results that match all filters got 100 extra points. Elements that match only the target filters got 50 extra points. The values 100 and 50 were just arbitrary numbers used to guarantee that the elements matching all the filters were ranked before the elements only matching a strict subset of the filters. This can be viewed as a coordination level matching for the filter matching.

UAmsI03-SCAS-DocumentScore

This run is almost identical to the previous run. The only difference was that the candidate elements were assigned scores according to the article run instead of according to the element run.

UAmsI03-SCAS-MixedScore

The articles appearing in the article run are parsed in the same way as for the two previous cases. The candidate elements are assigned a score which is calculated by combining the RSV scores of the three base runs. Hence, the score of an element is a mixture of its own score, the score of the article containing it, and the scores of all elements that contribute to the XPath expression being matched. More precisely, the element score was calculated using the formula

$$\begin{aligned} RSV(e) = \alpha \cdot \left(s(r) + \sum_{f \in F_r} s(f) + \sum_{c \in C_r} \max s(c) \right) \\ + (1-\alpha) \cdot \left(s(e) + \sum_{f \in F_e} s(f) + \sum_{c \in C_e} \max s(c) \right), \quad (6) \end{aligned}$$

where F_r, C_r, F_e and C_e represent sets of elements passing the respective filter mentioned in Equation 5; $s(r)$ is the score of the article from the article run; $s(f)$ and $s(c)$ are scores from the filter

	MAP	p@5	p@10	p@20
$\lambda = 0.9$	0.1091	0.3308	0.2769	0.2250
$\lambda = 0.2$	**0.1214**	0.3231	**0.2923**	**0.2423**
$\lambda = 0.5$	0.1143	**0.3462**	**0.2923**	0.2346

Table 1: Results of the CO task

run; and $s(e)$ is the score from the element run. In all cases we set $\alpha = 0.5$. We did not have any training data to estimate an optimal value for this parameter. We did not apply any normalization to the RSVs before combining them.

For all the SCAS runs, the elements are also filtered using the strict filters (Figure 3*e*). Any filtering using tag-names used the tag equivalence relations defined in the topic development guidelines.

3.3 Vague Content-And-Structure task

Since the definition of the task was a bit underspecified, we did not have a clear idea about what this task was about. With our runs we tried to cast light on whether this task is actually a content-only task, a content-and-structure task or a traditional article retrieval task.

UAmsI03-VCAS-NoStructure
This is a run that is similar to our CO runs. We chose a value $\lambda = 0.5$ for the smoothing parameter. We used the full content queries, expanded by blind feedback. We only considered elements containing at least 20 terms.

UAmsI03-VCAS-TargetFilter
This run is more similar to our SCAS runs. We chose a value $\lambda = 0.5$ for the smoothing parameter. We used the full content queries, expanded by blind feedback. Furthermore, we only returned elements having the same tag-name as the rightmost part of `targetPath`. Where the target element was not explicitly stated (*-targets), we only considered elements containing at least 20 terms.

UAmsI03-VCAS-Article
This run is a combination of two article runs using unweighted combSUM [7]. The two runs differ in the way that one is aimed at recall but the other at high precision. The one that aims at recall used $\lambda = 0.15$ and the full content queries, expanded by blind feedback. The high precision run used $\lambda = 0.70$ and as queries only the text appearing in the filters of the topic title. The RSV values of the runs were normalized before they were combined.

For all the VCAS runs, the length prior from equation 3 was added to the score.

4. RESULTS

Our runs were evaluated using version 2003.004 of the evaluation software provided by the INEX 2003 organizers. We used version 2.3 of the assessments. All runs were evaluated using the strict quantization; i.e., an element is considered relevant if and only if it is highly exhaustive and highly specific.

4.1 Content-Only task

Table 1 shows the results of the CO runs. Figure 4 shows the precision-recall plots. The CO runs at INEX 2003 were evaluated using *inex_eval*, the standard precision-recall measure for INEX.

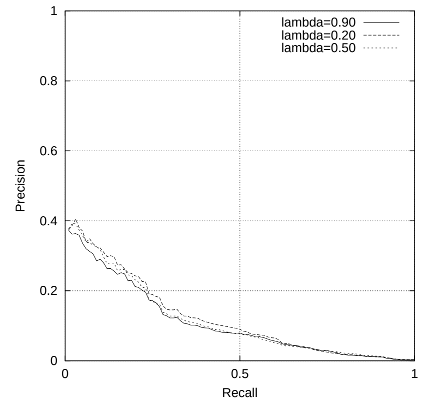


Figure 4: Precision-recall curves for our CO submissions, using the strict evaluation measure

	MAP	p@5	p@10	p@20
ElementScore	0.2650	**0.4273**	**0.3455**	0.2432
DocumentScore	0.2289	0.2909	0.2636	0.2136
MixedScore	**0.2815**	0.4000	0.3318	**0.2773**

Table 2: Results of the SCAS task

Furthermore, two other measures were developed, *inex_eval_ng(s)*, a precision recall measure that takes size of retrieved components into account; and *inex_eval_ng(o)*, which considers both size and overlap of retrieved components [1]. At the time when this report is written, a working version of the latter two measures had not been released. We will therefore only report on our results using the inex_eval measure.

It looks like our runs are very similar. There is not much difference in scoring and the graphs look the same. It seems that index size cut-off reduces the effect of the smoothing parameter reported in [4], where the absence of smoothing provided bias toward larger elements. Here the cut-off already eliminates the smallest elements and there is less need for extreme size bias.

According to the inex_eval measure, the run using $\lambda = 0.2$ has over all highest MAP score. The run using $\lambda = 0.5$ and filter out elements outside the $\langle$`bdy`$\rangle$ tag, gives slightly higher precision when 5 elements were retrieved. The run using $\lambda = 0.2$ does however catch up quite quickly. The runs seem to be so similar that any differences are unlikely to be statistically significant.

4.2 Strict Content-And-Structure task

Table 2 shows the results of the SCAS runs. Figure 5 shows the precision-recall plots. The run using the combination of element-, document- and filter-RSVs has higher MAP than the other two runs. The run based on element scores has slightly lower MAP than the combination run. The run based on document scores has the lowest MAP.

The run based on element scores outperforms the other two at low recall levels. We can see from the table that the element based run has the highest precision after only 5 or 10 documents have been retrieved. The combination run catches up with the element score based run once 20 documents have been retrieved. This indicates

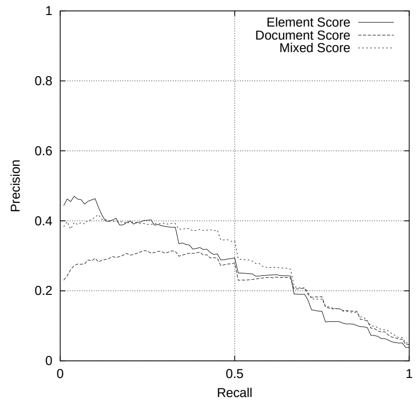


Figure 5: Precision-recall curves for our SCAS submissions, using the strict evaluation

	MAP	p@5	p@10	p@20
NoStructure	**0.1270**	**0.2880**	0.2520	**0.1980**
TargetFilter	0.0647	**0.2880**	**0.2640**	0.1960
Article	0.0627	0.2080	0.1800	0.1300

Table 3: Results of the VCAS task

that the coordination level matching for the filter matching, works well for initial precision, but is not as useful at higher recall levels.

4.3 Vague Content-And-Structure task

Table 3 shows the results of the VCAS runs. Figure 6 shows the precision-recall plots. Treating the VCAS task as a CO task, results in a higher MAP than either treating it as an SCAS task or an article retrieval task. The main difference lies in the recall. By limiting the set of returned elements, by considering the structure or retrieving articles, we discard elements that have a big potential of being relevant to the user. Hence we can never obtain maximal recall.

Looking at the precision at low recall levels, the difference between the two element retrieval runs is not so great. If we look at precision after 5, 10 and 20 elements have been retrieved, we see that treating the VCAS task as an SCAS task performs comparable to treating it as a CO task. The strict implementation of the task can even help early precision.

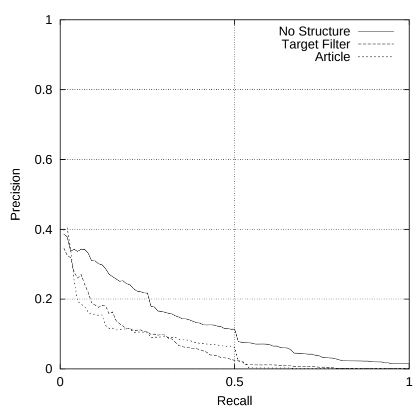


Figure 6: Precision-recall curves for our VCAS submissions, using the strict evaluation

5. CONCLUSIONS

This paper described our official runs for the INEX 2003 evaluation campaign. Our main research question was to further investigate the appropriate unit of retrieval. Although this problem is most visible for INEX's CO task, it also plays a role in the element and filter base runs for the CAS topics. With default adhoc retrieval settings, small XML elements dominate the ranks of retrieved elements. We conducted experiments with a number of approaches that aim to retrieve XML elements similar to those receiving relevance in the eyes of the human assessors. First, we experimented with a uniform length prior, ensuring the retrieval of larger sized XML elements [5]. Second, we experimented with Rocchio blind feedback, resulting in longer expanded queries that turn out to favor larger XML elements than the original queries. Third, we experimented with size cut-off, only indexing the element that contain at least 20 words. Fourth, we experimented with an element filter, ignoring elements occurring in the front and back matter of articles. Fifth, we experimented with smoothing settings, where the increase of the term importance weight leads to the retrieval of larger elements [4]. Finally, we combined approaches in various ways to obtain the official run submission. We plan to give an overview of the relative impact of these approaches in the final proceedings of INEX.

Our future research focuses on the question of what is the appropriate statistical model for XML retrieval. In principle, we could estimate language models from the statistics of the article index similar to standard document retrieval. An alternative is to estimate them from the statistics of the element index, or from a particular subset of the full element index. In particular, we smooth our element language model with collection statistics from the overlapping element index. Arguably, this may introduce biases in the word frequency and document frequency statistics. Each term appearing in an article usually creates several entries in the index. The overall collection statistics from the index may not best estimator for the language models. In our current research we investigate the various statistics from which the language models can be estimated.

6. REFERENCES

[1] N. Gövert, G. Kazai, N. Fuhr, and M. Lalmas. Evaluating the effectiveness of content-oriented XML retrieval. Technical report, University of Dortmund, Computer Science 6, 2003.

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

[3] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. The Importance of Morphological Normalization for XML Retrieval. In N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas, editors, *Proceedings of the First Workshop of the Initiaitve for the Evaluation of XML Retrieval (INEX)*, pages 41–48. ERCIM Publications, 2003.

[4] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. Topic Field Selection and Smoothing for XML Retrieval. In A. P. de Vries, editor, *Proceedings of the 4th Dutch-Belgian Information Retrieval Workshop*, pages 69–75. Institute for Logic, Language and Computation, 2003.

[5] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. XML Retrieval: What to Retrieve? In C. Clarke, G. Cormack, J. Callan, D. Hawking, and A. Smeaton, editors, *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 409–410. ACM Press, 2003.

[6] J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System — Experiments in Automatic Document Processing*. Prentice Hall, 1971.

[7] Joseph A. Shaw and Edward A. Fox. Combination of multiple searches. In D.K. Harman, editor, *Proceedings TREC-2*, pages 243–249. NIST, 1994.

[8] Snowball. The snowball string processing language, 2003. `http://snowball.tartarus.org/`.

[9] XPath. Xml path language, 1999. `http://www.w3.org/TR/xpath`.

[10] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM Press, 2001.

Bayesian Networks and INEX'03

Benjamin Piwowarski
LIP 6, Paris, France
bpiwowar@poleia.lip6.fr

Huyen-Trang Vu
LIP 6, Paris, France
vu@poleia.lip6.fr

Patrick Gallinari
LIP 6, Paris, France
gallinar@poleia.lip6.fr

ABSTRACT

We present a bayesian framework for XML document retrieval. This framework allows us to consider content only. We perform the retrieval task using inference in our network. Our model can adapt to a specific corpus through parameter learning and uses a grammar to speed up the retrieval process in big or distributed databases. We also experimented list filtering to avoid element overlap in the retrieved element list.

Keywords

Bayesian networks, INEX, XML, Focused retrieval, Structured retrieval

1. INTRODUCTION

The goal of our model is to provide a generic system for performing different IR tasks on collections of structured documents. We take an IR approach to this problem. We want to retrieve specific relevant elements from the collection as an answer to a query. The elements may be any document or document part (full document, section(s), paragraph(s), ...) indexed from the structural description of the collection. We consider the task as a *focused retrieval*, first described in [1, 7].

This year, we focused on *content only* (CO) queries since many questions still remain open for this specific task. The Bayesian Network (BN) model is is briefly described in section 2.1. We also present modifications with respect to the model we presented last year.

2. MODELS

The generic BN model used for the CO task was described in last year proceedings [8]. We only give here the main model characteristics. Our work is an attempt to develop a formal model for structured document access. Our model relies on bayesian networks and provides an alternative to other specific approaches for handling structured documents [6, 3, 4]. BN offer a general framework for taking into account relation dependencies between different structural elements. Those elements, which we call *doxels* (for Document Element) will be random variables in our BN.

We believe that this approach allows casting different access information tasks into a unique formalism, and that these models allow performing sophisticated inferences, e.g. they allow to compute the relevance of different document parts in the presence of missing or uncertain information. Compared to other approaches based on BN, we propose a general framework which should adapt to different types of structured documents or collections. Another original aspect of our work is that model parameters are learnt from data. This allows to rapidly adapt the model to different document collections and IR tasks.

Compared to last year baseline model, we have proceeded this year to different additions:

- We experimented with different weighting schemes for terms in the different doxels. Weight importance may be relative to the whole corpus of documents, to doxels labelled with the same tag, ...;
- We introduced a grammar for modelling different constraints on the possible relevance values of doxels in a same path ;
- For limiting the overlap of retrieved doxels, we introduced simple filtering techniques.

2.1 Bayesian networks

The BN structure we used directly reflects the document hierarchy, *i.e.* we consider that each structural part within that hierarchy as an associated random variable. The root of the BN is thus a "corpus" variable, its children the "journal collection" variables, etc. In this model, due to the conditional independence property of the BN variables, relevance is a local property in the following sense: if we know that the journal is (not) relevant, the relevance value of the journal collection will not bring any new information on the relevance of one article of this journal.

In our model, the random variable associated to a structural element can take three different values in the set $V = \{N, G, E\}$ which is related to the specificity dimension of the INEX'03 assessment scale:

N (for Not relevant) when the element is not relevant;

G (for too biG) when the element is marginally or fairly specific;

E (for Exact) when the element has an high specificity.

For any element e and for a given query, the probability $P(e = E|\text{query})$ gives us the *final* Retrieval Status Value (RSV) of this element. This value is used to order the different elements before returning the list of retrieved elements.

We considered two more types of random variables. The first one is the query need that is described as a vector of word frequencies. Note that this random variable is always observed (known). The second one is associated to *baseline* models and can take only two values : *relevant* and *not relevant*.

For a given query, a local relevance score is computed for each doxel via the baseline score models. This score only depends on the query and the doxel constraint without influence other doxels in the tree. Based on these local scores and on parameters, BN inference is then used to combine evidence and scores for the different doxels in the document model. For computing the local score, different models could be used. We used in our experiments simple retrieval methods and classical ones such as Okapi. The first one (*ratio*) compute for each element the value S_1:

$$S_1(\text{element}) = \frac{\sum_{\text{term}t} tf_{\text{query}}(t) \frac{tf_{\text{element}}(t)}{tf_{\text{parent}}(t)}}{\sum_{\text{term}t} tf_{\text{query}}(t)}$$

where tf_{parent} denotes the term frequency in the parent of the element, tf_{element} the term frequency within the element and tf_{query} within the query. The second one (*weight ratio*) is simply S_1 divided by a decreasing function of the element length:

$$S_2(\text{element}) = \frac{S_1(\text{element})}{\log(20 + \text{length}(\text{element}))}$$

where the length of the element is number of words that this element and its descendants contain. All those formulas and coefficient were determined empirically. The main advantages of these formulas are that they are naturally bounded (between 0 and 1) and that they can be computed *locally*. We can then simply define the probability that an element is relevant (R) for the first (resp. second) model M_1 (M_2) by:

$$P(M_i = R|\text{query}, \text{elementcontent}) = S_i \text{ with } i \in \{1, 2\}$$

We also tried to add the classical Okapi model, but as its RSV are harder to normalize, we were not able to integrate it with success into our BN framework.

In our model, the probability that element is in the state N, G or E depends on the parent state and on the fact that M_i has judged the element as relevant or not relevant (figure 1). We can then compute the probability using this formula for any element e and any state $v \in V$:

$$\begin{aligned} P(e = v|\text{query}) = \sum_{\substack{v_p \in V \\ r_1, r_2 \in \{R, \neg R\}}} & \theta_{c(e),v,v_p,r_1,r_2} \\ & \times P(e \text{ parent} = v_p) \\ & \times P(M_1 = r_1|\text{query}) \\ & \times P(M_2 = r_2|\text{query}) \end{aligned}$$

where θ is a learnt parameter that depends on the different states of the four random variables (element state, parent state, baseline model 1 and 2 relevance) and on the category $c(e)$ of the element. The categories used in our experiment

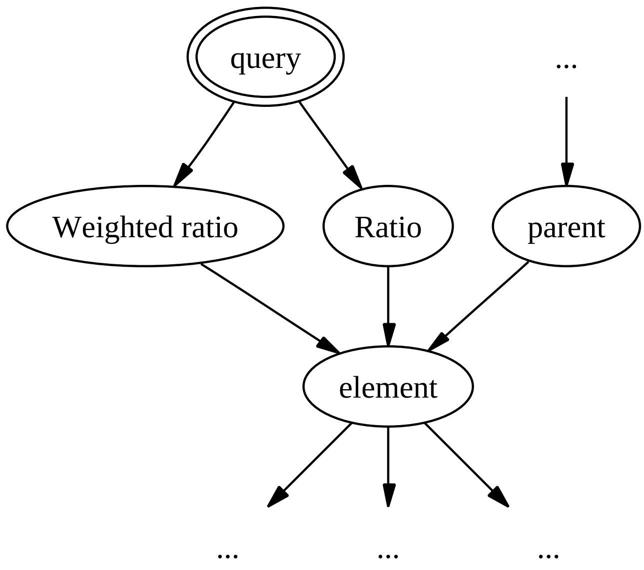


Figure 1: Bayesian Network model (detail view). The element state depends on the parent state and on the relevance of the element for the model *ratio* (M_1) and *weighted ratio* (M_2)

are shown in table 1. In our BN, scores are computed recursively with the above formula: we begin by the biggest doxels (INEX volumes) and then with smaller doxels (article, body paragraph, ...).

Adding a grammar to the BN

We used a grammar in order to add some constraint on the retrieval inference process. That grammar enables us to express coherence rules on scored doxels within the same document path:

- A non relevant element *may not* have a relevant descendant:

$$\forall c, r_1, r_2, \theta_{c,v,N,r_1,r_2} = 0 \text{ if } v \in \{G, E\}$$

- A highly specific (G) element has either a non relevant (N) or a highly specific (E) child

$$\forall c, r_1, r_2, \theta_{c,G,E,r_1,r_2} = 0$$

The main interest of this grammar is to provide us a way to make a decision about whether we can find an element which has a higher RSV in the set of descendants of a given element. Indeed, we can show that:

$$P(e = E|\text{query}) \leq P(p = E|\text{query}) + P(p = G|\text{query}) \quad (1)$$

where p is the parent of the doxel e.

Learning parameters

In order to fit a specific corpus, parameters are learnt from observations using the Expectation Maximization (EM) algorithm. An observation $O^{(i)}$ is a query with its associated

tags	category $c(e)$
ss, ss1, sec1	section
bib, bibl, ack, reviewers	misc
ip, ip1, ip2, ip3, bb, app, p1, p2	paragraph
figw, fig	figure
l1, l2, l3, l4, l5, l6, l7, l8, l9, la, lb, lc, ld, le, numeric-list, numeric-rbrace, bullet-list, index	list
index-entry, item-none, item-bold, item-both, item-bullet, item-diamond, item-letpara, item-mdash, item-numpara, item-roman, item-text	item
hdr, hdr2, hdr1, h3, h2, h2a, h1a, h1, h	header
bdy, article	container
* (any other tag)	other

Table 1: Element categories

relevance assessments (document/part is relevant or not relevant to the query). EM [2] optimizes the model parameters Θ with respect to the likelihood $\mathcal{L}$ of the observed data:

$$\mathcal{L}(O, \Theta) = \log P(O|\Theta)$$

where $O = \left\{O^{(1)}, \ldots, O^{(|O|)}\right\}$ are the N observations. Observations may or may not be *complete*, *i.e.* relevance assessments need not to be known for each structural element in the BN in order to learn the parameters. Each observation O_i can be decomposed into E_i and H_i where E_i corresponds to structural entities for which we know whether they are relevant or not, i.e. structural parts for which we have a relevance assessment. E_i is called the evidence. H_i corresponds to hidden observations, i.e. all other nodes of the BN.

In our experiment, we used for learning the 30 CO queries from INEX'02 and their associated relevance assessments.

2.2 Filtering

A Structured IR system has to cope with overlapping doxels, as it may for example return a section and a paragraph. In order to avoid duplicate information, it might be interesting to filter out the returned result in order to choose between different levels of granularity. We thus developed a simple filtering algorithm which we describe below. The basic idea is to remove an element when another element in the retrieved list contains or is contained by the element. For INEX'03, we chose a very simple filtering mainly motivated by intuition.

The filtering we chose removes some of the retrieved doxels in the list while preserving the relative ranking of other document components. Kazai et al. [5] had this idea with the so-called BEP[1]. We can consider our filtering step as an instance of BEP which doesn't take into account hyperlinks.

[1]Best Entry Point

Filtering is a necessary step for improving the effectiveness of Structured IR systems.

We tried the three following strategies:

Root oriented If a doxel appears on the retrieved list, its descendants in the document tree will not give any new information if they appear below in the list. We thus remove any element in the ranked list if an ancestor is higher in the list. This simple method favors large doxels which is in conflict with the CO objective (retrieve the most specific doxels as possible).

Leaf oriented This is the inverse of the previous approach. We remove an element from the list when there is a descendant higher. The limit of this method is that when the latest is not relevant, then all the other informations brought by the ancestor are lost for the user.

BEP BEP strategy cumulates root and leaf oriented filtering. That is, an element is kept only if there is neither descendant nor ancestor higher in the retrieved list.

We chose the "Root oriented" strategy for some of the official submissions for INEX'03.

3. EVALUATION

The definition of a measure is based on an hypothetical user behaviour. Hypothesis used in classical measures are subjective but do reflect a reality. In the Structured IR framework (SIR), we propose a measure that estimate the number of relevant doxels a user might see and that does depend on a specific user behaviour.

We made three specific hypothesis on the user behaviour. First, the user eventually consults the structural context (parent, children, siblings) of a returned doxel. This hypothesis is related to the inner structure of documents; Second, the specificity of a doxel influences the behaviour of the user. Third, The user will not use any hyperlink. More precisely, he will not jump to another document. This hypothesis is valid in the INEX corpus but can easily be removed in order to cope with hyperlinked corpora.

The measure we propose is the expectation of the number of relevant doxels a user sees when he consult the list of the k first returned doxels divided by the expectation of the number of relevant doxels a user see if he explores all the doxels of the database. We denote this measure by ERR (for Expected Ratio of Relevant documents). This measure is normalized so it can be averaged over queries.

4. CONCLUSION

We have described a new model for performing IR on structured documents. It is based on BN whose conditional probability functions are learnt from the data via EM. This model uses a grammar for restricting the allowed state of a doxel in our BN knowing the state of its parent. The BN framework has thus three advantages:

1. it can be used in distributed IR, as we only need the score of the parent element in order to compute the score of any its descendants;

2. it can use simultaneously different baseline models: we can therefore use specific models for non textual media (image, sound, ...) as another source of evidence;

3. Whole part of the corpus can be ignored when retrieving doxels using inequality (1).

The model has still to be improved, tuned and developed, and several limitations have still to be overcome in order to obtain an operational structured information retrieval system. In particular, we should improve the baseline models[2] and further experiments are thus needed for tuning the learning algorithms and for filtering.

5. REFERENCES

[1] Y. Chiaramella, P. Mulhem, and F. Fourel. A Model for Multimedia Information Retrieval. Technical report, IMAG, Grenoble, France, July 1996.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from incomplete data via de EM algorithm. *The Journal of Royal Statistical Society*, 39:1–37, 1977.

[3] N. Fuhr and T. Rölleke. HySpirit - a Probabilistic Inference Engine for Hypermedia Retrieval in Large Databases. In H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, Valencia, Spain, 1998. Springer, Berlin.

[4] T. Grabs and H.-J. Schek. ETH Zrich at INEX: flexible information retrieval from XML with PowerDB-XML. Dec. 2002.

[5] G. Kazai, M. Lalmas, and T. Rölleke. A Model for the Representation and Focussed Retrieval of Structured Documents based on Fuzzy Aggregation. In *String Processing and Information retrieval (SPIRE 2001) Conference*, Laguna de San Rafael, Chile, Sept. 2001.

[6] M. Lalmas. Dempster-Shafer's Theory of Evidence Applied to Structured Documents: Modelling Uncertainty. In *Proceedings of the 20th Annual International ACM SIGIR*, pages 110–118, Philadelphia, PA, USA, July 1997. ACM.

[7] M. Lalmas and E. Moutogianni. A Dempster-Shafer indexing for the focussed retrieval of a hierarchically structured document space: Implementation and experiments on a web museum collection. In *6th RIAO Conference, Content-Based Multimedia Information Access*, Paris, France, Apr. 2000.

[8] B. Piwowarski, G.-E. Faure, and P. Gallinari. Bayesian networks and INEX. In *Proceedings of the First Annual Workshop of the Initiative for the Evaluation of XML retrieval (INEX)*, DELOS workshop, Dagstuhl, Germany, Dec. 2002. ERCIM.

[2]We tried to use one of the Okapi variants but it gives a score that can not be easily normalised

IRIT at INEX 2003

Karen Sauvagnat
IRIT/SIG-RFI
118 route de Narbonne
31062 Toulouse cedex 4
+33-5-61-55-68-99
sauvagna@irit.fr

Gilles Hubert
IRIT/SIG-EVI
118 route de Narbonne
31062 Toulouse cedex 4
+33-5-61-55-74-35
hubert@irit.fr

Mohand Boughanem
IRIT/SIG-RFI
118 route de Narbonne
31062 Toulouse cedex 4
+33-5-61-55-74-16
bougha@irit.fr

Josiane Mothe
IRIT/SIG-EVI
118 route de Narbonne
31062 Toulouse cedex 4
+33-5-61-55-63-22
mothe@irit.fr

ABSTRACT
This paper describes the retrieval approaches proposed by IRIT in INEX'2003 evaluation. The primary approach uses Mercure system and different modules to perform content only and content and structure queries. The paper also discusses a second approach based on a voting method previously applied in the context of automatic text categorization.

Keywords
Information Retrieval, XML retrieval, connectionist model, voting method, automatic text categorization

1. INTRODUCTION
XML (eXtensible Markup Language) has recently emerged as a new standard for representation and data exchange on the Internet [29]. If this tendency goes on, XML will certainly become a universal format and HTML (Hypertext Markup Language) will disappear in aid of XML. Consequently, the information retrieval issue in XML collections becomes crucial.

A growing number of approaches are dealing with structured documents like XML. They can be divided into three main groups: database, XML-oriented specific approaches and IR approaches. The database community considers XML collections as databases, and tries to develop models for representing and querying documents, according to the content and the structure of these documents. Many languages have been developed for querying and updating these databases [1][18][24][30][11]. XML specific oriented approaches estimate the relevance of document parts according to the relevance of their structurally related parts. They are also named aggregation-based methods [8][15][7][13][16]. In IR approaches, traditional IR models are adapted to be used on structured collections [17][20][22].

In this paper, we present two IR approaches applied to structured documents retrieval, within the context of INEX'2003: the first approach uses Mercure information retrieval system, while the second one is based on a voting method used initially for automatic text categorization. Section 2 presents the INEX initiative. Section 3 describes the Mercure model, and the INEX search approach with Mercure system is reported in section 4. Section 5 and 6 present first the voting method defined in the context of categorization and then the adaptations we integrated within the INEX'2003 context.

2. THE INEX INITIATIVE
2.1 Collection
INEX collection, 21 IEEE Computer Society journals from 1995-2002, consists of 12 135 (when ignoring the volume.xml files) documents with extensive XML-markup. All documents respect the same DTD.

2.2 Queries
As last year, participants to INEX'2003 have to perform two types of queries. CO (Content Only) queries are requests that ignore the document structure and contain only content related conditions, e.g. only specify what a document/component should be about. CAS (Content and Structure) queries contain explicit references to the XML structure, and restrict the context of interest and/or the context of certain search concepts. Both CO and CAS topics are made up of four parts: topic title, topic description, narrative and keywords.

Within the ad-hoc retrieval task, three sub-tasks are defined: (1) the CO task, using CO queries, (2) the SCAS task, using CAS queries, for which the structural constraints must be strictly matched, (3) the VCAS task, also using CAS queries, but for which the structural constraints can be considered as vague conditions.

3. MERCURE SYSTEM
Mercure is a full-text information retrieval system based on a connectionist approach and modeled by a multi-layer network. The network is composed of a query layer (set of query terms), a term layer (representing the indexing terms) and a document layer [4].

Mercure includes the implementation of retrieval process based on spreading activation forward and backward through the weighted links. Queries and documents can be used either as inputs or outputs. The links between layers are symmetric and their weights are based on the *tf-idf* measure inspired by OKAPI [23] and SMART term weighting.

The query-term links are weighted as follows :

$$q_{ui} = \begin{cases} \dfrac{nq_u * qtf_{ui}}{nq_u - qtf_{ui}} \text{ if } (\mathrm{nq_u} > qtf_{ui}) \\ qtf_{ui} \text{ otherwise} \end{cases} \quad (1)$$

Where:

- q_{ui} : the weight of the term t_i in the query u at the stage s
- qtf_{ui}: the frequency of the query term t_i in the query u
- nq_u: the number of terms in the query u

The term-document link weights are expressed by :

$$d_{ij} = \frac{tf_{ij} * (h_1 + h_2 * \log(\frac{N}{n_i}))}{h_3 + h_4 * \frac{dl_j}{\Delta_l} + h_5 * tf_{ij}} \qquad (2)$$

Where:

- d_{ij} : term-document weight of term t_i and document d_j
- tf_{ij}: term frequency of t_i in the document d_j
- N: total number of documents
- n_i: number of documents containing term t_i
- h_1,h_2,h_3,h_4 and h_5: constant parameters
- Δl : average document length
- dlj :number of terms in the document dj

The query evaluation function computes the similarity between queries and documents.

Each term node computes an input value: $In(t_i) = q_{ui}^{(s)}$

and an activation value: $Out(t_i) = g(In(t_i))$, where g is the term layer activation function.

Each term node propagates then this activation value to the document nodes through the term-document links. Each document node computes an input value: $In(d_j) = \sum_i Out(t_i) * d_{ij}$ and an activation value: $Out(d_j) = g(In(d_j))$, where g is the document layer activation function.

Documents are then ranked by ascending order of their activation value.

The ranking function (activation) is modified to take into account term proximity in a document [14]. Thus, documents having close query terms compute a new input value:

$$In(d_j) = \sum_i (Out(t_i) * d_{ij}) * \sum_i \frac{\alpha}{prox_{i,i-1}} \qquad (3)$$

Where:

- α is a constant parameter so that $\frac{\alpha}{prox_{i,i-1}} \geq 1$. α is set to 4 for INEX'2003 experiments.
- $prox_{i,i-1}$ is the number of terms separating the query terms t_i and t_{i-1} in the window of α terms in the document. The query terms are ranked according to their position in the query text.

In other words, documents having close query terms (i.e. no more than α words separate two consecutive query terms in the document content) increase their input value.

In addition, we have implemented two modules that are used to process structured documents. The aim of these modules is to filter the most specific[1] and exhaustive[2] elements of the documents returned by Mercure [15].

[1] An element is specific to a query if all its information content concerns the query.

[2] An element is exhaustive to a query if the element contains all the required information.

The first module, which is *content-oriented*, deals with queries composed of simple keyword terms. It glances through documents retrieved by Mercure, and finds elements answering the queries in the most specific and exhaustive way. Element types that can be retrieved are pre-specified manually, according to the DTD of the documents. This module performs as follow: for each document retrieved by Mercure, it searches occurrences of query terms in all pre-specified elements. It returns the elements containing the greatest number of query terms. If more than *k* elements are supposed to be the most specific and exhaustive, the module returns the whole document.

The second module, which is *content-and-structure-oriented*, performs queries containing both explicit references to the XML structure and content constraints. These queries can be divided into two parts : a target element and a content constraint on this target element. As the content-oriented module, the second module browses documents returned by Mercure, and returns specific elements (e.g. target elements) containing the greatest number of query terms specified in the content constraints. If the target elements do not contain any of the terms of the content constraints, the document retrieved by Mercure is removed from the list of results.

Thus, the main difference between the two modules is the way they process the documents structure. In the content-oriented module, elements that can be returned are pre-specified manually without user intervention. The user only gives keywords and cannot express structural conditions in his query. Using the content-and-structure-oriented module, users explicitly give a target element and content constraints on this target element.

As a result for both modules, we obtain a ranked list of elements/documents.

4. THE INEX SEARCH APPROACH WITH THE MERCURE SYSTEM

4.1 Indexing the INEX database and the queries

The INEX collection was indexed in order to take into account term positions in the documents. Terms are stemmed with Porter algorithm and a stop-word list is used in order to remove non-significant terms from the index. No structural information is kept in the index.

Queries are then indexed in two different ways.

CO and CAS queries are first indexed using title and keywords fields, in order to build queries *for Mercure system*. Regarding CO queries, we simply remove terms preceded by minus (which means that the user does not want these terms appear in the results) and keep all the other terms. CAS queries are indexed using terms in the content constraints of the title field and terms of the keyword field. For both types of queries, terms are stemmed with the Porter algorithm and terms appearing in the stop-word list are removed, as it is done for the documents.

Then, CAS queries are re-indexed *for the content-and-structure-oriented module*. Indeed, as explained before, the content-and-structure-oriented module needs the target element of queries in order to process them. Let us take some examples of CAS queries:

Top.	Title field	Description
63	//article[about(.,'"digital library"') AND about (.//p,'+authorization +"access control" + security')]	Relevant documents are about digital libraries and include one or more paragraphs discussing security, authorization or access control in digital libraries.
66	>/article[.//yr <='2000'] //sec[about(.,'"search engines"')]	The user is looking for sections of articles published before 2000, which discuss search engines.
84	//p[about(.'overview "distributed query processing" join')]	The user wants paragraphs that give an overview about distributed query processing techniques with a focus on joins implementations.
90	//article[about(./sec,' +trust authentication "electronic commerce" e-commerce e-business marketplace') //abs[about(.,'trust authentication')]	The user wants to find abstracts or article that discuss automated tools for establishing trust between parties on the internet. The article should discuss applications of trust for authenticating parties in e-commerce.

Table 1: Examples of CAS queries

All the content constraints occurring in the *about* predicates are first indexed for Mercure system, even though they are not on the target element (in topics 63 and 90 for example). Targets elements (*article* for topic 63, *section* for topic 66, *paragraph* for topic 84 and *abstract* for topic 90) are then indexed for the content-and-structure-oriented module.

About 20% of the CAS topics (like topic 66) contain a constraint on the year of publication. This constraint is also stored and will be used to filter results of the content-and-structure-oriented module.

4.2 Retrieval

In both cases (CO queries and CAS queries), a first search is performed with Mercure search engine using the content part of the queries. As a result, a ranked list of 1000 documents is selected for each query. Then, the content-oriented module is used to process the document results of CO queries, and the content-and-structure-oriented module for CAS queries. Both modules return a ranked list of elements/documents, derived from the first ordered list of documents returned by Mercure system.

4.2.1 Retrieval with CO queries

According to the DTD, we have decided to allow the content-oriented module to return only *section* or *abstract* elements. Indeed, section and abstract elements are supposed to be large enough to be exhaustive and small enough to be specific.

If the content-oriented module finds more than two relevant elements (k =2) within a given document, the whole document is returned.

4.2.2 Retrieval with CAS queries

The content-and-structure-oriented module browses documents returned by Mercure, and returns target elements containing the greatest number of query terms specified in *all* the content constraints of CAS queries. If no occurrence of terms contained in the content constraints is found in target elements, the document returned by Mercure is removed from the results list. Indeed, the target element always have a content constraint.

Then, if the query contains a year constraint, elements returned by the content-and-structure-oriented module are filtered, according to the article publication date .

4.3 Submitted runs

The first goal of our experiments in INEX'2003 is to test whether a full-text information retrieval system can be easily adapted to structured retrieval and to evaluate how suitable are the full-text IR based techniques for such kind of retrieval. Our approach can be compared to the fetch and browse method proposed in [5]. No static structure is used a priori and so, all types of XML documents can be processed. The second goal of our experiments is to measure the effect of term positions in INEX query types.

Five runs performed with Mercure have been submitted to INEX'2003. The runs are labeled as follows:

- *pos* indicates that Mercure uses term positions to process queries (*Mercure2.pos_co_ti, Mercure2.pos_cas_ti, Mercure2.pos_vcas_ti*), otherwise runs are based on a Mercure simple search (*Mercure2.co_ti, Mercure2.cas_ti*).
- *Co (Mercure2.co_ti, Mercure2.pos_co_ti), cas (Mercure2.cas_ti, Mercure2.pos_cas_ti)*, and *vcas (Mercure2.pos_vcas_keyti)* indicate the sub-task type, e.g. CO, SCAS or VCAS
- *ti* indicates that only title field of queries was used (*Mercure2.co_ti, Mercure2.pos_co_ti, Mercure2.cas_ti, Mercure2.pos_cas_ti*) , whereas *keyti* indicates that the title and keywords fields were used (*Mercure2.pos_vcas_keyti)*.

4.4 First results

The following table shows the results of the five runs, in terms of average precision:

Run	**Strict quantization**		**Generalized quantization**	
	Average precision	**Rank**	**Average precision**	**Rank**
Mercure2.co_ti	0.0056	50/56	0.0088	48/56
Mercure2.pos_co_ti	0.0344	28/56	0.0172	41/56
Mercure2.cas_ti	0.0719	33/38	0.0612	34/38
Mercure2.pos_cas_ti	0.1641	25/38	0.1499	24/38
Mercure2.pos_vcas_keyti	NC	NC	NC	NC

Table 2: Results of the five runs performed with Mercure system

The first result that can be drawn from Table 2 is that runs using term positions are definitely better that simple search for both query types (CO and CAS). Average precision for runs using term positions (*Mercure2.pos_cas_ti* , *Mercure2.pos_vcas_keyti*, and *Mercure2.pos_co_ti*) is about four times higher than average precision of runs performed with a single Mercure search (*Mercure2.cas_ti* , *Mercure2.co_ti*).

4.5 Discussion and future works

Regarding this year experiments and results, some investigations have to be performed. First of all, for the CO task, elements that can be returned by the content-oriented module are pre-selected manually. These types of elements are not always necessarily the most exhaustive and specific: it depends on the way the DTD was understood by the document creators. Statistics [12] or aggregation methods [7] [13] may be used to find those elements automatically. Then, the content-and-structure-oriented module is not able to perform all the content and structural constraints. Indeed, it processes only content constraint on the target element and year constraints. For example, in topic 90, the first *about* predicate is on sections, whereas the target element is abstract: the module does not insure that the content constraint on sections is respected. However, topics such as topic 84 are fully treated. According to these remarks, the content-and-structure-oriented module seems to be more adapted to the VCAS task. For this purpose, the run *Mercure2.pos_vcas_keyti* was performed and submitted. Finally, query processing is relatively slow, because the modules have to browse all documents returned by Mercure in order to find relevant elements. Regarding these limitations, an indexing model taking into account the structural and content information of documents seems to be necessary.

Moreover, our approach uses the *idf* measure to compute a retrieval status value for documents (and then documents are browsed to return relevant elements). The *idf* measure is also used in [7] and [26], in order to directly return relevant elements. However, term occurrences in elements do not necessary follow a Zipf law [31]. The number of term repetitions can be (very) reduced in XML documents and *idf* is not necessarily appropriate [6][10]. The use *of ief (Inverse Element Frequency)* is proposed in [28] and [9]. An indexing scheme storing different IR statistics might be interesting on the INEX collection: thus, combinations of IR and XML-specific approaches could be tested.

5. A VOTING METHOD FOR INFORMATION RETRIEVAL

The approach proposed is derived from a process for categorisation of textual documents. This categorisation intends to link documents with pre-defined categories. Our approach focuses on categories organised as a taxonomy. The original aspect is that our approach involves a voting principle instead of a classical similarity computing.

Our approach associates each text with different categories as opposed to most of the other categorisation techniques. The association of a text to categories is based on the Vector Voting method [21]. This method relies on the terms describing each category and their automatic extraction from the text to be categorised. The voting process evaluates the importance of the association between a given text and a given category. This method is similar to the HVV method (Hyperlink Vector Voting) used within the Web context to compute the pertinence of Web page regarding the web sites referring to it [19]. In our context, the initial strategy considers that the more the category terms appear in the text, the more the link between the text and this category is strong.

The association principle between a document and categories is composed of different steps:

– Compute the profile of each category. In automatic categorisation, profiles correspond generally to a set of weighted terms [25][27] which can be obtained by training from previous categorised documents.

– Extract automatically the concepts describing a document and their importance for the document. The extraction is based on a set of rules to treat, for example, document tags, and processes to treat synonymy and to remove stop words.

– For each category of the hierarchy, compute a score with a voting function which measures the representativity of the category according to the text. Different functions can be used as voting function based on measures such as term importance in text and in hierarchy, text size, hierarchy size, number of terms describing a category that appear in the text.

– Sort the winning categories according to their score, and eventually select the best categories (for example, scores greater than a fixed threshold, or n greatest scores).

We have studied different voting functions whose results are presented in [2][3]. The voting function must take into account the importance in the document of each term describing the category, the discriminant power of each term describing the category, the category representativity within the document. The function providing the best results is described as follows :

$$Vote(E_H, D) = \sum_{\forall t \in E} \frac{F(t,D)}{S(D)} \cdot \frac{S(H)}{F(t,H)} \cdot e^{\frac{NT(E,D)}{NT(E)}} \qquad (1)$$

where

E_H corresponds the category E in the hierarchy H

D is a document

$\frac{F(t,D)}{S(D)}$	This factor measures the importance of the term t in the document D. F(t,D) corresponds the number of occurrences of the term t in the document D and S(D) corresponds to the size (number of terms) of D.
$\frac{S(H)}{F(t,H)}$	This factor measures the discriminant power the term t in the hierarchy H. F(T,H) corresponds to the number of occurrences of the term t in the hierarchy H and S(H) corresponds to size of H.
$\frac{NT(E,D)}{NT(E)}$	This factor measures the presence rate of terms representing the category in the text (importance of the category). NT(E) corresponds to the number of terms in the category E and NT(E,D) corresponds to the number of terms of the category E that appear in the document D

The above function (1) considers the two factors as equivalent: the importance of a term in the document and the discriminant power of this term in the hierarchy. Applying the exponential function to the third factor (i.e. the presence rate of terms representing the category in the text) aims at accentuate its importance.

The function is completed with the notion of *coverage*. The aim of the coverage is to ensure that only categories enough represented in a document will be selected for this document. The coverage is a threshold corresponding to the percentage of terms from a category that appear in a text . For example, a coverage of 50% implies that at least half of terms describing a category have to appear in the text of a document to be selected.

6. THE INEX SEARCH APPROACH WITH A VOTING METHOD

6.1 Evolution if the categorisation process

From the topic point of view, CO and CAS topics are constituted of different informative parts (title, keywords, description) that can be exploited to construct their profile. Although our method can use all the possible parts we first focused on to the title and keyword parts for the INEX'2003 experiments. For both topic types, stop words are removed and optionally terms can be stemmed using Porter algorithm.

For CAS topics, an additional step identify the structural constraints indicated in a topic. All the structural constraints defined on target elements of topics are taken into account and stored to be processed in a post categorisation step to filter the results issued from the categorisation step. Only, the results having expected xpaths are kept. About content structural constraints (e.g. about(.//p,'+authorization +"access control" +security') or .//yr <='2000') only constraints on the year of the article are taken into account and stored to filter the results. More complex content constraints have not been treated for INEX'2003. Next experiments are planned about the extension of the voting method to take into account such constraints.

From the INEX collection point of view, the documents are considered as sets of text chunks identified by xpaths. For each document, concepts are extracted automatically with the different xpaths identifying the chunks where they appear and their importance in the chunk is calculated. For INEX'2003 experiments all XML tags have been taken into account.

The voting method is applied without any modification. Topics are considered as categories to which document elements have to be assigned. The result is constituted of a list of topics associated to each chunk of text (identified by its xpath) for each document.

6.2 Experiments

Our experiments aim at evaluating the efficiency of the voting function and estimating the adaptations needed for the categorisation process in a context such as INEX'2003.

Four runs based on the voting method were submitted to INEX'2003. The main parameter that distinguishes the runs is to apply or not a coverage (C50 corresponds to apply a coverage of 50% i.e. half of the terms describing the topic must appear in the text to keep the topic, C0 corresponds to no coverage). No stemming process has been applied for the submitted although it can be added. The tcXX% parameter specifies that only the elements having a score over a given percentage of the best score will be kept (e.g. tc50% indicates that only the elements having a score over the half of the best score are kept in the result)..

6.3 Results

The following table shows the preliminary results of the four runs based on the voting method :

Run	**Strict quantization**		**Generalized quantization**	
	Average precision	**Rank**	**Average precision**	**Rank**
VotingNoStemTKCO tc75%C0nonorm	0.0012	54/56	0.0041	56/56
VotingNoStemTKVCAS C50nonorm	NC	NC	NC	NC
VotingNoStemTKSCAS tc50%C0nonorm	0.0626	34/38	0.0746	31/38
VotingNoStemTKVCAS tc50%C0nonorm	NC	NC	NC	NC

Table 3: Results of the 4 runs performed with the voting method

Results for VCAS topics are not yet known.

6.4 Discussion and future works

Regarding the experiment that were performed and the obtained results we can notice that:

- the voting method applied without coverage tends to promote short chunks of text that have only one term in common with the topic. Introducing coverage intends to correct this since short chunks of text that have several terms in common with the topic are less frequent than longer ones. We plan to study changes made to the voting function to evaluate their impact on results notably with regard to the size of text chunks.
- The elementary level has been considered to identify the different chunks of text. This choice leads to miss complex chunks of text constituted of different elementary chunks with high voting scores. A rebuilding of complex chunk should be integrated in the process.
- Structural constraints defined on the content of topics have not been taken into account. This aspect constitutes the main axis of study to extend the voting method. The main idea is to integrate the constraint when computing the voting score in order to promote relevant text chunks regarding content which respect the structural constraints without eliminating relevant chunks (regarding content) but that do not satisfy the constraints.

7. ACKNOWLEDGMENTS

We thank Cécile Laffaire for her participation in this work.

8. REFERENCES

[1] Abiteboul, S. , Quass, D., Mc Hugh, J., Widom, J., Wiener, J-L.. The Lorel query language for semi-structured data. International Journaf on Digital Libraries, 1(1), 68-88, 1997.

[2] Augé, J., Englmeier, K., Hubert, G., Mothe, J. Catégorisation automatique de textes basée sur des hiérarchies de concepts , 19^{ième} Journées de Bases de Données Avancées, Lyon, p. 69-87, 2003.

[3] Augé, J., Englmeier, K., Hubert, G., Mothe, J. Classification automatique de textes basée sur des hiérarchies de concepts , Veille stratégique, scientifique et technologique, Barcelone, p. 291-300, 2001.

[4] Boughanem, M., Chrisment, C., Soule-Dupuy , C. Query modification based on relevance back-propagation in ad-hoc environment. Information Processing and Management, 35 (1999), 121-139, 1999.

[5] Chiaramella, Y. , Mulhem, P. , Fourel, F. A model for multimedia search information retrieval. Technical report, Basic Research Action FERMI 8134, 1996.

[6] Fuhr, N., Gövert, N., Röelleke, T. Dolores: a system for logic based retrieval of multimedia objects. In Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1998, Melbourne, Australia, pages 257-265. ACM Press, 1998.

[7] Fuhr, N., Grossjohann, K. XIRQL: A query Language for Information Retrieval in XML Documents. In Proceedings of the 24th annual ACM SIGIR conference on research and development in Information Retrieval, New Orleans, USA, pages 172-180. ACM Press, 2001.

[8] Fuller, M., Mackie, E., Sacks-Davis, R., Wilkinson, R. Structural answers for a large structured document collection. In Proc. ACM SIGIR, pp. 204-213. Pittsburgh, 1993.

[9] Grabs, T., Sheck, H.J. ETH Zürich at INEX : Flexible Information Retrieval from XML with PowerDB-XML. In INEX 2002 Workshop Proceedings, p. 35-40, Germany, 2002.

[10] Grabs, T. Storage and retrieval of XML documents with a cluster of databases systems. PhD thesis, Swiss Federal Institute of Technology Zurich, 2003.

[11] Grosjohann, K. Query Formulation and Result Visualization for XML Retrieval . In Proc of the SIGIR 2000 Workshop on XML and Information Retrieval. Athens, Greece, 2000.

[12] Hatano, K., Kinutani, H., Watanabe, M. An Appropriate Unit of Retrieval Results for XML Document Retrieval. In INEX 2002 Workshop Proceedings, p. 66-71, Germany, 2002.

[13] Kazai, G., Lalmas, M., Roelleke, T. Focused document retrieval, 9th International Symposium on string processing and information retrieval, Lisbon, Portugal, September 2002.

[14] Kean, E.M. The use of term position devices in ranked output experiments, Journal of Documentation, v.47 n.1, p.1-22, March 1991 .

[15] Lalmas, M. Dempster-Shafer's theory of evidence applied to structured documents : Modeling uncertainty. In Proc. ACM-SIGIR, pp. 110-118. Philadelphia, 1997.

[16] Lalmas, M., Roelleke, T. Four-valued knowledge augmentation for structured document retrieval. International Journal of Uncertainty, Fuziness and Knowledge-based systems (IJUFKS), Special issue on Management of uncertainty and imprecision in multimedia information systems, 11(1), 67-86, February 2003.

[17] Larson, R. R. . Cheshire II at INEX : Using a hybrid logistic regression and Boolean model for XML retrieval. In INEX 2002 Workshop Proceedings, p. 2-7, Germany, 2002

[18] Levy, A., Fernandez, M., Suciu, D., Florescu, D., Deutsch A. . XML-QL : A query language for XML. World Wide Web Consortium technical report, Number NOTE- xml-ql-19980819, 1998.

[19] Li, Y. Toward a qualitative search engine , IEEE Internet Computing, vol. 2, n° 4, p. 24-29, 1998.

[20] Ogilvie, P., Callan, J. : Languages models and structured documents retrieval. In INEX 2002 Proceedings, p. 18-23, Germany, 2002.

[21] Pauer, B., Holger, P. Statfinder . Document Package Statfinder, Vers. 1.8, may 2000.

[22] Piwowarski, B., Faure, G.E. , Gallinari, P. . Bayesian Networks and INEX. In INEX 2002 Workshop Proceedings, p. 7-12, Germany, 2002.

[23] Robertson, SE , Walker, S. Okapi/Keenbow at TREC-8. In Proceedings of the TREC-8 Conference, National Institute of Standards and Technology, pages 151-161, 2000.

[24] Robie , J., Lapp, J., Schach, D. XML Query Language (XQL). Proceedings of W3C QL'98 (Query Languages 98). Massachussets, 1998.

[25] Salton, G., « The SMART Retrieval System », Experiments in automatic document processing, Prentice Hall Inc., Englewood Cliffs, NL, 1971.

[26] Theobald, A., Weikum, G. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In C. S. Jensen, K. G. Jeffery, J. Pokorny, S. Saltenis, E. Bertino, K. Bäohm, and M. Jarke, editors, Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, volume 2287 of Lecture Notes in Computer Science, pages 477-495. Springer, 2002.

[27] Van Rijsbergen, K. Information Retrieval . Butterworths, London, Second Edition, 1979. http://www.dcs.gla.ac.uk/Keith/Preface.html

[28] Wolff, J.E., Flörke, H., Cremers, A.B. Searching and browsing collections of structural information. In Proc of IEEE advances in digital libraries, pp. 141-150. Washington, 2000.

[29] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/REC-xml , Oct. 2000.

[30] World Wide Web Consortium. Xquery 1.0: an XML query language. http://www.w3.org/TR/xquery/ , Aug. 2003.

[31] Zipf, G. Human Behaviour and the Principle of Least Effort. Addison-Wesley, 1949.

Using Language Models for Flat Text Queries in XML Retrieval

Paul Ogilvie, Jamie Callan
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA USA
{pto,callan}@cs.cmu.edu

ABSTRACT

This paper presents a language modeling system for ranking flat text queries against a collection of structured documents. The system, built using Lemur, produces probability estimates that arbitrary document components generated the query. This paper describes storage mechanisms and retrieval algorithms for the evaluation of unstructured queries over XML documents. The paper includes retrieval experiments using a generative language model on the content only topics of the INEX testbed, demonstrating the strengths and flexibility of language modeling to a variety of problems. We also describe index characteristics, running times, and the effectiveness of the retrieval algorithm.

1. INTRODUCTION

Language modeling has been studied extensively in standard Information Retrieval in the last few years. Researches have demonstrated that the framework provided by language models has been powerful and flexible enough to provide strong solutions to numerous problems, including ad-hoc information retrieval, known-item finding on the Internet, filtering, distributed information retrieval, and clustering.

With the success of language modeling for this wide variety of tasks and the increasing interest in studying structured document retrieval, it is natural to apply the language modeling framework to XML retrieval. This paper describes and presents experiments using one way the generative language model could be extended to model and support queries on structured documents. We model documents using a tree-based language model. This is similar to many previous models for structured document retrieval [1][2][3][6][7][10], but differs in that language modeling provides some guidance in combining information from nodes in the tree and estimating term weights. This work is also similar to other works using language models for XML retrieval [5][9], but differs in that we also present context-sensitive language model smoothing and an implementation using information retrieval style inverted lists rather than a database.

The next section provides background in language modeling in information retrieval. In Section 3 we present our approach to modeling structured documents. Section 4 describes querying the tree-based language models presented in the previous section. In Section 5we describe the indexes required to support retrieval and the retrieval algorithms. We describe the experiment setup and indexes used for INEX 2003 in Section 6. Section 7 describes experimental results. We discuss relationships to other approaches to structured document retrieval in Section 8, and Section 9 concludes the paper.

2. LANGUAGE MODELS FOR DOCUMENT RETRIEVAL

Language modeling applied to information retrieval problems typically models text using unigram language models. Unigram language models are similar to bags-of-words representations, as word order is ignored. The unigram language model specifically estimates the probability of a word given some text. Document ranking typically is done one of two ways: by measuring how much a query language model diverges from document language models [8], or by estimating the probability that each document generated the query string. Since we use the generative language model for our experiments, we will not describe the divergence based approaches here.

2.1 The Generative Language Model

The generative method ranks documents by directly estimating the probability of the query using the texts' language models [12][4][14][15]:

$$\mathrm{P}(\mathrm{Q}|\theta_\mathrm{T}) = \prod_{w \in Q} \mathrm{P}(w|\theta_\mathrm{T})^{qtf(w)}$$

where Q is the query string, and θ_T is the language model estimated for the text, and $qtf(w)$ is the query term frequency of the term. Texts more likely to have produced the query are ranked higher. It is common to rank by the log of the generative probability as it there is less danger of underflow and it produces the same orderings:

$$\log(\mathrm{P}(\mathrm{Q}|\theta_\mathrm{T})) = \sum_{w \in Q} qtf(w) \log \mathrm{P}(w|\theta_\mathrm{T})$$

Under the assumptions that query terms are generated independently and that the query language model used in KL-divergence is the maximum-likelihood estimate, the generative model and KL divergence produce the same rankings [11].

2.2 The Maximum-Likelihood Estimate of a Language Model

The most direct way to estimate a language model given some observed text is to use the maximum-likelihood estimate, assuming an underlying multinomial model. In this case, the maximum-likelihood estimate is also the empirical distribution. An advantage of this estimate is that it is easy to compute. It is very good at estimating the probability distribution for the language model when the size of the observed text is very large. It is given by:

$$\mathrm{P}(w|\theta_\mathrm{T}) = \frac{freq(w,\mathrm{T})}{|\mathrm{T}|}$$

where T is the observed text, *freq*(w, T) is the number of times the word w occurs in T, and |T| is the length in words of T. The maximum likelihood estimate is not good at estimating low frequency terms for short texts, as it will assign zero probability to those words. This creates a problem for estimating document language models in both KL divergence and generative language model approaches to ranking documents, as the log of zero is negative infinity. The solution to this problem is smoothing.

2.3 Smoothing

Smoothing is the re-estimation of the probabilities in a language model. Smoothing is motivated by the fact that many of the language models we estimate are based on a small sample of the "true" probability distribution. Smoothing improves the estimates by leveraging known patterns of word usage in language and other language models based on larger samples. In information retrieval smoothing is very important [15], because the language models tend to be constructed from very small amounts of text. How we estimate low probability words can have large effects on the document scores. In addition to the problem of zero probabilities mentioned for maximum-likelihood estimates, much care is required if this probability is close to zero. Small changes in the probability will have large effects on the logarithm of the probability, in turn having large effects on the document scores. Smoothing also has an effect similar to inverse document frequency [4], which is used by many retrieval algorithms.

The smoothing technique most commonly used is linear interpolation. Linear interpolation is a simple approach to combining estimates from different language models:

$$\mathrm{P}(w|\theta) = \sum_{i=1}^{k} \lambda_i \mathrm{P}(w|\theta_i)$$

where k is the number of language models we are combining, and λ_i is the weight on the model θ_i. To ensure that this is a valid probability distribution, we must place these constraints on the lambdas:

$$\sum_{i=1}^{k} \lambda_i = 1 \quad \text{and for } 1 \le i \le k,\ \lambda_i \ge 0$$

One use of linear interpolation is to smooth a document's language model with a collection language model. This new model would then be used as the smoothed document language model in either the generative or KL-divergence ranking approach.

2.4 Another Characterization

When we take a simple linear interpolation of the maximum likelihood model estimated from text and a collection model, we can also characterize the probability estimates as:

$$\mathrm{P}(w|\theta_{\mathrm{T}}) = \begin{cases} \mathrm{P_{seen}}(w|\theta_{\mathrm{T}}) & \textit{if } w \in \mathrm{T} \\ \mathrm{P_{unseen}}(w|\theta_{\mathrm{T}}) & \textit{otherwise} \end{cases}$$

where

$$\mathrm{P_{seen}}(w|\theta_{\mathrm{T}}) = (1-\omega)\mathrm{P_{MLE}}(w|\theta_{\mathrm{T}}) + \omega \mathrm{P}(w|\theta_{collection})$$

and

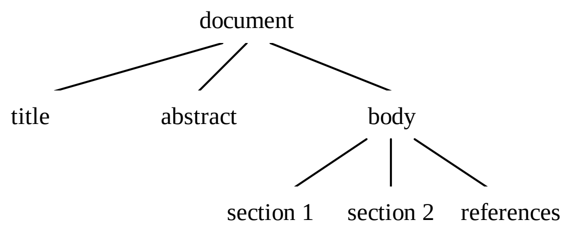
$$\mathrm{P_{unseen}}(w|\theta_{\mathrm{T}}) = \omega \mathrm{P}(w|\theta_{collection})$$

This notation distinguishes the probability estimates for cases where the word has been seen in the text and where the word has not been seen will be in the sample text. We will use this notation later when describing the retrieval algorithm, as it simplifies the description and is similar to the notation used in previous literature [15]. The simple form of linear interpolation where ω is a fixed constant is often referred to as Jelinek-Mercer smoothing.

3. STRUCTURED DOCUMENTS AND LANGUAGE MODELS

The previous section described how language modeling is used in unstructured document retrieval. With structured documents such as XML or HTML, we believe that the information contained in the structure of the document can be used to improve document retrieval. In order to leverage this information, we need to model document structure in the language models.

We model structured documents as trees. The nodes in the tree correspond directly with tags present in the document. A partial tree for a document might look like:



Nodes in the document tree correspond directly to XML tags in the document. For each document node in the tree, we estimate a language model. The language models for leaf nodes with no children can be estimated from the text of the node. The language models for other nodes are estimated by taking a linear interpolation of a language model formed from the text in the node (but not in any of its children) and the language models formed from the children.

We have not specified how the linear interpolation parameters for combining language models in the document tree should be chosen. This could be task specific, and training may be required. The approach we will adopt in this paper is to set the weight on a child node as the accumulated length of the text in the child divided by the accumulated length of the node. By accumulated length we mean the number of words directly in the node plus the accumulated length of the node's children. Setting the parameters in this manner assumes that a word in a one node type is no more important than a word in any other node type; it is the accumulated length of the text in the node that determines how much information is contained in the node.

We also wish to smooth the maximum likelihood models that are estimated directly from the text with a collection language model. In this work, we will combine the maximum likelihood models with the collection model using a linear interpolation with fixed weights. The collection model may be specific to the node type, giving context sensitive smoothing, or the collection model may be one large model estimated from everything in the corpus, giving a larger sample size.

When the λ parameters are set proportional to the text length and a single collection model is used, this results a special case that is very similar to the models used in [5][9]. The tree-based language model estimated using these parameter settings will be identical to a language model estimated by taking a simple linear interpolation of a maximum likelihood estimate from the text in the node and its ancestors and a the collection model.

4. RANKING THE TREE MODELS

In a retrieval environment for structured documents, it is desirable to provide support for both structured queries and unstructured, free-text queries. It is easier to adapt the generative language model to structured documents, so we only consider that model in this paper. It is simpler to support unstructured queries, so we will describe retrieval for them first.

4.1 Unstructured Queries

To rank document components for unstructured queries, we use the generative language modeling approach for IR described in Section 2. For full document retrieval, we need only compute the probability that the document language model generated the query. If we wish to return arbitrary document components, we need to compute the probability that each component generated the query.

Allowing the system to return arbitrary document components may result in the system stuffing the results list with many components from a single document. This behavior is undesirable, so a filter on the results is necessary.

One filter we employ takes a greedy approach to preventing overlap among components in the results list. For each result, it will be thrown out of the results if there is any component higher in the ranking that is an ancestor or descendent of the document component under consideration.

4.2 Structured Queries

Our previous paper on this subject [11] discusses how some structural query operators could be included in the model. We do not currently support any of these operators in our system, so we will not discuss in depth here. However, we will note that the retrieval framework can support most desired structural query operators as relatively easy to implement query nodes.

4.3 Prior Probabilities

Given relevance assessments from past topics, we can estimate prior probabilities of the document component being relevant given its type. Another example prior may depend on the length of the text in the node. A way to incorporate this information is to rank by the probability of the document node given the query. Using Bayes rule, this would allow us incorporate the priors on the nodes. The prior for only the node being ranked would be used, and the system would multiply the probability that the node generated the query by the prior:

$$\begin{aligned} P(N|Q) &= P(Q|\theta_N)P(N)/P(Q) \\ &\propto P(Q|\theta_N)P(N) \end{aligned}$$

This would result in ranking by the probability of the document component node given the query, rather than the other way around.

5. STORAGE AND ALGORITHMS

This section describes how we support structured retrieval in the Lemur toolkit. We first describe the indexes built to support retrieval. Then we describe how the indices are used by the retrieval algorithm. We also present formulas for the computation of the generative probabilities we estimate for retrieval.

5.1 Index Support

There are two main storage structures in Lemur that provide the support necessary for the retrieval algorithm. Lemur stores inverted indexes containing document and node occurrences and document structures information.

5.1.1 Inverted Indexes

The basic idea to storing structured documents in Lemur for retrieval is to use a modified inverted list. Similar to storing term locations for a document entry in an inverted list, we store the nodes and the term frequencies of the term in the nodes in the document entries of the inverted list. The current implementation of the structured document index does not store term locations, but could be adapted to store term locations in the future.

The inverted lists are keyed by term, and each list contains the following:

- document frequency of the term
- a list of document entries, each entry containing
  - document id
  - term frequency (count of term in document)
  - number of nodes the term occurs in
  - a list of node entries, each entry containing
    - node id
    - term frequency (count of term in node)

When read into memory, the inverted lists are stored in an array of integers. The lists are stored on disk using restricted-variable length compression and delta-encoding is applied to document ids and node ids. In the document entry lists, the documents entries are stored in order by ascending document id. The node entry lists are similarly stored in order by increasing node id. Document entries and node entries are only stored in the list when the term frequency is greater than zero. Access to the lists on disks is facilitated with an in-memory lookup table for vocabulary terms.

There is also an analogous set of inverted lists for attribute name/value pairs associated with tags. For example, if the document contained the text

<date calendar="Gregorian">,

the index would have an inverted list keyed by the triple date/calendar/Gregorian. The structure and information stored in the inverted lists for the attribute name/value pairs is identical to those in the inverted lists for terms.

5.1.2 Document Structure

The document structure is stored compressed in memory using restricted variable length compression. A lookup table keyed by document id provides quick access to the block of compressed memory for a document. We choose to store the document structure in memory because it will be requested often during retrieval. For each document, a list of information about the document nodes is stored. For each node, we store:

- parent of the node
- type of node
- length of the node (number of words)

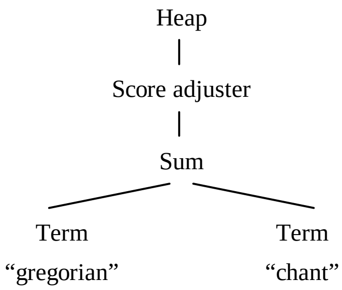
Since this list of information about the document structure is compressed using a variable length encoding, we must decompress the memory to provide efficient access to information about nodes. When the document structure for a document is being decompressed, we also compute:

- accumulated length of the node (length of text directly in the node + accumulated length of children)
- number of children of the node
- a list of the node's children

This decompression and computation of other useful information about the document structure is computed in time linear to the number of nodes in the document being decompressed.

5.2 Retrieval

We construct a query tree to process and rank document components. A typical query tree is illustrated below. The leaf nodes of the query tree are term nodes which read the inverted lists for a term off of disk and create result objects for document components containing the term. The term nodes are also responsible for propagating the term scores up the document tree. The sum node merges the result lists returned by each of the term nodes, combining the score estimates. The score adjuster node adjusts the score estimates to get the generation probabilities and also applies any priors. The heap node maintains a list of the top *n* ranked objects and returns a sorted result list. Efficient retrieval is achieved using a document at a time approach. This requires that the query tree be walked many times during the evaluation of a query, but results a large saving of memory, as only the result objects for a document and the top *n* results objects in the heap must be stored at any point in time.



A more detailed description of each of the query nodes follows. When each query node is called, they are passed a document id to evaluate. In order to know which document should be processed next, the term nodes pass up the next document id in the inverted list. For other query nodes, the minimum next document id among a node's children gets passed up the query tree with the results list. We will describe the query nodes bottom up, as that is how the scores are computed.

We first note that we can rewrite the log of the probability that the document node generated the query as

$$\log\left(\mathrm{P}\left(\mathrm{Q}\middle|\theta_{node}\right)\right) = \sum_{w \in Q, node} qtf(w) \log\left(\frac{\mathrm{P}_{\mathrm{seen}}\left(w\middle|\theta_{node}\right)}{\mathrm{P}_{\mathrm{unseen}}\left(w\middle|\theta_{node}\right)}\right) + \sum_{w \in Q} qtf(w) \log \mathrm{P}_{\mathrm{unseen}}\left(w\middle|\theta_{node}\right)$$

as shown in [15]. This will allow us to easily compute the item in the first sum easily using term nodes, combine these components of the score using a sum node, and then add on the rest using a score adjustment node.

5.2.1 Term Node

The term nodes read in the inverted lists for a term *w* and create results where the score for a result is initialized to

$$qtf(w) \cdot \log\left(\frac{\mathrm{P}_{\mathrm{seen}}\left(w\middle|\theta_{node}\right)}{\mathrm{P}_{\mathrm{unseen}}\left(w\middle|\theta_{node}\right)}\right)$$

The term node assumes that the parent id of a node is smaller than the node's id. It also assumes that the document entries in inverted lists are organized in increasing document id order and the node entries are organized in increasing term id order. The structured document index we built is organized this way. In the following algorithm description, indentation is used to denote the body of a loop.

1 Seek to the next entry in the inverted list where the document id is at least as large as the requested document

2 If the document id of the next entry is the requested document

3   Decompress the document structure information for the document

4   Read in the node entries from the inverted list

5   Create the result objects for the leaf nodes. For each node that contains the term:

6   **Initialize the score for the result to the seen probability part for the node**

$$seen(node) = (1-\omega)\, freq(w, node)\, \lambda(node, node)$$

where

$$\lambda(node, node) = \frac{length(node)}{accumulated\ length(node)}$$

and ω will be used to set the influence of the collection models.

7   Push the node id onto the candidate node heap

8   Store the result object in an array indexed by node id for fast access

9   While the candidate node heap isn't empty:

10   Pop the top node id off of the heap (the largest node id), set it to the current node id

11   Lookup the result from the result array

12   Lookup the node id for the parent of the current node

13   Lookup the parent node's result

14   If the parent node's result object is NULL:

15   Create a new result object for the parent node and put it in the result array, initializing the score to 0

16   Push the parent node's id onto the candidate node heap

17 **Propagate the seen part of the score from the current node to the parent node, setting the parent node's seen part to**

$$seen(parent) + seen(node)\lambda(node, parent)$$

where

$$\lambda(node, parent) = \frac{accumulated\ length(node)}{accumulated\ length(parent)}$$

18 Push the result onto the front of the results list

19 Set the result in the result array for the node to NULL (initializing the result array for the next document)

[*Now each document node that contains the query term (or has a child containing the term) has a result in the results list where the score is the seen probability part for the query term*]

20 For each node in the result list

21 **Compute the unseen part of the generative probability for each node. For linear interpolation with a constant ω and one single node type independent collection model, this is**

$$unseen(w, node) = \omega \mathrm{P}\left(w \middle| \theta_{collection}\right)$$

For linear interpolation with a constant ω and node type specific collection models, this can be computed recursively

$$\begin{aligned} &unseen(w, node) = \\ &\omega \mathrm{P}\left(w \middle| \theta_{collection,type(node)}\right)\lambda(node, node) \\ &+ \sum_{child \in children(node)} unseen(w, child)\lambda(child, node) \end{aligned}$$

22 **Set the score for the result to**

$$qtf(w) \cdot \log\left(\frac{seen(node) + unseen(w, node)}{unseen(w, node)}\right)$$

23 Return the result list and the next document id in the inverted list

The result list now contains results for a single document where the score is

$$qtf(w) \cdot \log\left(\frac{\mathrm{P}_{\mathrm{seen}}\left(w \middle| \theta_{node}\right)}{\mathrm{P}_{\mathrm{unseen}}\left(w \middle| \theta_{node}\right)}\right)$$

and the list is ordered by increasing node id.

5.2.2 Sum Node

The sum node maintains an array of result lists, with one result list for each of the children. It seeks to the next entry in each of the child result lists where the document id is at least as large as the requested document. If necessary, it calls the children nodes to get their next result lists. For the requested document, the sum node merges results from the result lists of the children, setting the score of the new result equal to the sum of the children's results with the same document and node id. This node assumes that results in a result list are ordered by increasing document id, then increasing node id. The results returned by this component have the score

$$\sum_{w \in Q, node} qtf(w) \log\left(\frac{\mathrm{P}_{\mathrm{seen}}\left(w \middle| \theta_{node}\right)}{\mathrm{P}_{\mathrm{unseen}}\left(w \middle| \theta_{node}\right)}\right)$$

and the minimum document id returned by the children is returned.

5.2.3 Score Adjustment Node

The score adjustment node adds

$$\sum_{w \in Q} qtf(w) \log \mathrm{P}_{\mathrm{unseen}}\left(w \middle| \theta_{node}\right)$$

to each of the results, where

$$\mathrm{P}_{\mathrm{unseen}}\left(w \middle| \theta_{node}\right) = unseen(w, node)$$

as defined for the term node. If there is a prior probability for the node, the score adjustment node also adds on the log of the prior. The results in the list now have the score

$$\begin{aligned} &\sum_{w \in Q, node} qtf(w) \log\left(\frac{\mathrm{P}_{\mathrm{seen}}\left(w \middle| \theta_{node}\right)}{\mathrm{P}_{\mathrm{unseen}}\left(w \middle| \theta_{node}\right)}\right) \\ &+ \sum_{w \in Q} qtf(w) \log \mathrm{P}_{\mathrm{unseen}}\left(w \middle| \theta_{node}\right) \\ &+ \log(\mathrm{P}(node)) \end{aligned}$$

$$= \log\left(\mathrm{P}\left(\mathrm{Q} \middle| \theta_{node}\right)\mathrm{P}(node)\right)$$

which is the log of the score by which we wish to rank document components.

5.2.4 Heap Node

The heap node repeatedly calls its child node for result lists until the document collection has been ranked. The next document id it calls for its child to process is the document id returned by the child node in the previous evaluation call. It maintains a heap of the top *n* results. After the document collection has been ranked, it sorts the results by decreasing score and stores them in a result list that is returned.

5.2.5 Other Nodes

There are many other useful nodes that could be useful for retrieval. One example is a node that filters the result lists so that the XML path of the node in the document tree satisfies some requirements. Another example is a node that throws out all but the top *n* components of a document.

6. EXPERIMENT SETUP

The index we created used the Krovetz stemmer and InQuery stopword list. Topics are similarly processed, and all of our queries are constructed from the title, description, and keywords fields. All words in the title, description, and keywords fields of the topic are given equal weight in the query. Table 3 shows the size of components created to support retrieval on the INEX document collection. The total index size including information needed to do context sensitive smoothing is about 70% the size of the original document collection. A better compression ratio could be achieved by compression of the context sensitive smoothing support files. Note that the document term file which is 100 MB is not necessary for the retrieval algorithms described above.

Topic Fields	Context	Prior	Path	inex_eval Strict	Gen
TDK	YES	NO	NO	.0464	.0646
TDK	YES	YES	NO	.0488	.0653
TDK	NO	NO	NO	.0463	.0641
TDK	NO	YES	NO	.0485	.0654

Table 1: Performance of the retrieval system on INEX 2002 CO topics. Context refers to context sensitive smoothing, prior refers to the document component type priors, and path refers to the overlapping path filter.

Official Run Name	Topic Fields	Context	Prior	Path	inex_eval Strict	Gen	inex_eval_ng Strict	Gen	w/o overlap Strict	Gen
LM_context_TDK	TDK	YES	NO	NO	.0717	.0804	.2585	.3199	.2305	.2773
-	TDK	YES	YES	NO						
LM_context_typr_path_TDK	TDK	YES	YES	YES	.0203	.0240				
-	TDK	NO	NO	NO						
-	TDK	NO	YES	NO						
LM_base_typr_path_TDK	TDK	NO	YES	YES	.0204	.0234				

Table 2: Summary of runs and results for INEX 2003 CO topics.

Component	Size (MB)
Inverted file	100
Document term file (allows iteration over terms in a document)	100
Document structure	30
Attributes inverted file	23
Smoothing – single collection model	4
Smoothing – context sensitive models (not compressed)	81
Other files (lookup tables, vocabulary, table of contents, etc.)	12
Total	**350**

Table 3: Lemur structured index component sizes

Table 4 shows approximate running times for index construction and retrieval. The retrieval time for context insensitive smoothing is reasonable at less than 20 seconds per query, but we would like to lower the average query time even more. We feel we can do this with some simple data structure optimizations that will increase memory reuse.

Action	Time (mins)
Indexing	25
Retrieval of 36 INEX 2003 CO topics – context insensitive smoothing	10
Retrieval of 36 INEX 2003 CO topics – context sensitive smoothing	45

Table 4: Indexing and retrieval times using Lemur

The higher retrieval time for the context sensitive retrieval algorithm is due to the recursive computation of the *unseen* component of the score as described Step 21 of Section 5.2.1. Clever redesign of the algorithm may reduce the time some. However, all of the descendent nodes in the document's tree must be visited regardless of whether the descendent nodes contain any of the query terms. This means that the computation of the *unseen* component of the scores is linear in the number of nodes in the document tree, rather than the typically sub-linear case for computation of the *seen* score components. If the λ and ω functions and their parameters are known, it is possible to precompute and store necessary information to reduce the running time to something only slightly larger than the context insensitive version. However, our implementation is meant for research, so we prefer that these parameters remain easily changeable.

7. EXPERIMENT RESULTS

We submitted three official runs as described in Table 2. All of our runs used the title, description, and keyword fields of the topics. Unfortunately, two of our runs performed rather poorly. This is either an error in our path filter or a problem with the component type priors. We would also like to evaluate the additional runs corresponding to the dashes in the table, but we have not been able to do these experiments yet.

The LM_context_TDK run has good performance across all measures. This is our basic language modeling system using context sensitive smoothing. The strong performance of the context sensitive language modeling approach speaks well for the flexibility of language modeling.

Unfortunately, we have not been able to do a through evaluation of variations of the system to figure out which additional components are helpful. We have done some experiments on the INEX 2002 content only topics. The summary of our runs for the 2002 topics is given in Table 1. There is little for us to conclude from the 2002 topics. It is not clear that context sensitive smoothing makes any significant difference. The priors may give a small boost, but the priors were estimated directly from the relevance assessments for the 2002 CO topics. We would like to answer questions of whether context sensitive smoothing is helpful, whether a component type prior helps, and whether component retrieval for this task performs better than standard document retrieval.

8. RELATED WORK

There exists a large and growing body of work in retrieving information from XML documents. Some work is described in our previous paper [11] and much of the more recent work is also described in the INEX 2002 proceedings [13]. With that in mind, we will focus our discussion of related work on language modeling approaches for structured document retrieval.

In [5] a generative language modeling approach for content only queries is described where a document component's language model is estimated by taking a linear interpolation of the maximum likelihood model from the text of the node and its ancestors and a collection model. This corresponds to a special case of our approach. Our model is more flexible in that it allows context sensitive smoothing and different weighting of text in children nodes.

The authors of [9] also present a generative language model for content only queries in structured document retrieval. They estimate the collection model in a different way, using document frequencies instead of collection term frequencies. As with [5], this model can be viewed as a special case of the language modeling approach presented here.

9. CLOSING REMARKS

We presented experiments using a hierarchical language model. The strong performance of language modeling algorithms demonstrates the flexibility and ease of adapting language models to the problem. In our preliminary experiments, context sensitive smoothing did not give much different performance than using a single collection model.

We described data structures and retrieval algorithms to support retrieval of arbitrary XML document components within the Lemur toolkit. We are reasonably pleased with the efficiency of the algorithms for a research system, but we will strive to improve the algorithms and data structures to reduce retrieval times even further.

In our future work, we would like to compare the component retrieval to standard document retrieval. We would also like to investigate query expansion using XML document components. Additionally, we would like to explore different ways of setting the λ weights on the nodes' language models, as we believe that words in some components may convey more useful information than words in other components.

10. ACKNOWLEDGMENTS

This work was sponsored by the Advanced Research and Development Activity in Information Technology (ARDA) under its Statistical Language Modeling for Information Retrieval Research Program. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect those of the sponsor.

11. REFERENCES

[1] Fuhr, N. and K. Großjohann. XIRQL: A query language for information retrieval in XML documents. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 172-180.

[2] Grabs, T. and H.J. Schek. Generating vector spaces on-the-fly for flexible XML retrieval. In *Proceedings of the 25th Annual International ACM SIGIR Workshop on XML Information Retrieval* (2002), ACM.

[3] Hatanao, K., H. Kinutani, M. Yoshikawa, and S. Uemura. Information retrieval system for XML documents. In *Proceedings of Database and Expert Systems Applications* (DEXA 2002), Springer, 758-767.

[4] Hiemstra, D. *Using language models for information retrieval*, Ph.D. Thesis (2001), University of Twente.

[5] Hiemstra, D. A database approach to context-based XML retrieval. In [13], 111-118.

[6] Kazai, G., M. Lalmas, and T. Rölleke. A model for the representation and focused retrieval of structured documents based on fuzzy aggregation. In *The 8th Symposium on String Processing and Information Retrieval* (SPIRE 2001), IEEE, 123-135.

[7] Kazai, G., M. Lalmas, and T. Rölleke. Focussed Structured Document Retrieval. In *Proceedings of the 9th Symposium on String Processing and Information Retrieval* (SPIRE 2002), Springer, 241-247.

[8] Lafferty, J., and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 111-119.

[9] List, J., and A.P. de Vries. CWI at INEX 2002. In [13], 133-140.

[10] Myaeng, S.H., D.H. Jang, M.S. Kim, and Z.C. Zhoo. A flexible model for retrieval of SGML documents. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1998), ACM Press, 138-145.

[11] Ogilvie, P. and J. Callan. Language models and structured document retrieval. In [13], 33-40.

[12] Ponte, J., and W.B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1998), ACM Press, 275-281.

[13] Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX). 2003, DELOS.

[14] Westerweld, T., W. Kraaj, and D. Heimstra. Retrieving web pages using content, links, URLs, and anchors. In *Proceedings of the Tenth Text Retrieval Conference, TREC 2001*, NIST Special publication 500-250 (2002), 663-672.

[15] Zhai, C. and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 334-342.

Cheshire II at INEX '03: Component and Algorithm Fusion for XML Retrieval

Ray R. Larson
School of Information Management and Systems
University of California, Berkeley
Berkeley, California, USA, 94720-4600
ray@sherlock.berkeley.edu

ABSTRACT

This paper describes the retrieval approach that UC Berkeley used in the 2003 INEX evaluation. As in last year's INEX, our primary approach is the combination of a probabilistic methods using a Logistic regression algorithm for estimation of document (article) relevance and/or element relevance, along with Boolean constraints. This year we also used data fusion techniques to combine results from multiple probabilistic retrieval algorithms and multiple search elements for any given query. All of our runs were fully automatic with no manual editing or interactive submission of queries.

Keywords

Information Retrieval, IR Evaluation, XML Retrieval

1. INTRODUCTION

Early in the TREC evaluations a number of participating groups found that fusion of multiple retrieval algorithms provided an improvement over a single search algorithm[13, 2]. With ongoing improvements of the algorithms used in the TREC main (i.e., ad hoc retrieval) task, later analyses[10, 1] found that the greatest effectiveness improvements appeared to occur between relatively ineffective individual methods, and the fusion of ineffective techniques, while often approaching the effectiveness of the best single IR algorithms, seldom exceeded them for individual queries and never exceeded their average performance.

Our approach to XML retrieval in last year's INEX, as reported in our 2002 INEX paper[7], was to use a "Fusion Search" facility in the Cheshire II system that merged the result sets from multiple searches. For the majority of the content-only and content and structure queries separate searches from different indexes and different elements of the collection were merged into a single integrated result set. This facility was developed originally to support combination of results from distributed searches, but has proved to be quite valuable when applied to the differing elements of a single collection as well. At the time of last year's runs for INEX the only operators for combining the results of multiple subqueries within a single query were the Boolean operators AND, OR, and NOT. The Fusion search facility itself used only a simple weighted sum of scores from each input resultset. In Berkeley's 2002 INEX runs, our approach typically involved between 7 and 14 separate queries of the system that were then combined using the fusion search facility to determine the final ranking of the documents or components.

One of the main questions we were investigating in last year's INEX was how to take advantage of more precise search matches (e.g. Boolean title searches) when they are possible for a given query, yet to permit the enhanced recall that probabilistic queries provide. We found in subsequent analysis of the INEX 2002 results, that our implementation of this approach suffered significantly from a number of bugs. As noted in the final INEX 2002 paper, some of the bugs were found the script that converted the results to the INEX submission format, not in retrieval itself, where only the first occurrence of component retrieved for some of the queries was converted to an entry for the submission (this was most signicant in one query where all of the relevant components were in a single article).

We also discovered in analysis of the results from last year, that Fusion Searches were apparently not correctly accumulating scores for each component search in some cases. This turned out to be a particularly costly bug (in terms of the INEX performance measures) caused by a failure to sort some of the intermediate resultsets in a searches before they were merged, leading to an incorrect ranking sequence in the final resultsets, and in some particularly pathological situations resulting in the effective reversal of the correct ranking sequence. This bug was difficult to detect due to the complex interactions in combining multiple resultsets.

For the official INEX 2003 runs, the bugs noted above were corrected. But, unfortunately, another one was discovered rather late in the evaluation process, which may explain (in part) the worse-than-expected results obtained for the official runs (described below in the discussion of MERGE_NORM normalization).

Our principle approach this year was to expand on the basic fusion approach used last year, using both new implementations of algorithms, and new fusion operators. A major addition this year is that we have implemented, and employed, a version of the Okapi BM-25 algorithm (as well as other algorithms not used in the submitted runs). We have not, however, used blind relevance feedback with the Okapi algorithm which may explain (in part) the poor results obtained. The remainder of this paper describes the retrieval algorithms used, new methods for combining results for different elements, and discusses the comparative results for the different official runs. We hope to have versions of the official runs with all (known) bugs repaired to present at the meeting.

2. THE RETRIEVAL ALGORITHMS AND OPERATORS

The original design rationale and features of the Cheshire II search engine have been discussed elsewhere [9, 8] and will only be briefly repeated here with an emphasis on those features that were applied in the INEX evaluation. (This part is virtually identical to the discussion in last year's paper [7], and repeated only for reference). We will then describe our newly implemented algorithms and operators.

2.1 Original Probabilistic and Boolean Operations

The Cheshire II search engine supports various methods for translating a searcher's query into the terms used in indexing the database. These methods include elimination of "noise" words using stopword lists (which can be different for each index and field of the data), particular field-specific query-to-key conversion or "normalization" functions, standard stemming algorithms (a modified version of the Porter stemmer) and support for mapping database and query text words to single forms based on the WordNet dictionary and thesaurus using a adaption of the WordNet "Morphing" algorithm and exception dictionary.

The primary ranked retrieval operation for Cheshire II makes use of a *logistic regression* algorithm to estimate probability of relevance that was originally developed by Berkeley researchers and shown to provide excellent full-text retrieval performance in the TREC evaluation of full-text IR systems[5, 4, 3]. In this algorithm, the estimated probability of relevance given a particular query and a particular record in the database $P(R \mid Q, D)$ is calculated and the documents or components are presented to the user ranked in order of decreasing values of that probability. The actual implementation for calculating $P(R \mid Q, D)$ uses the "log odds" of relevance $\log O(R \mid Q, D)$, where for any events A and B the odds $O(A \mid B)$ is a simple transformation of the probabilities $\frac{P(A|B)}{P(\overline{A}|B)}$. The Logistic Regression model provides estimates for a set of coefficients, c_i, associated with a set of S statistics, X_i, derived from the query and database, such that

$$\log O(R \mid Q, D) \approx c_0 \sum_{i=1}^{S} c_i X_i$$

where c_0 is the intercept term of the regression.

For the set of M *terms* (i.e., words, stems or phrases) that occur in both a particular query and a given document or document component, the equation used in estimating the probability of relevance for the Cheshire II search engine is essentially the same as that used in [4] where the coefficients were estimated using relevance judgements from the TIPSTER test collection:

$X_1 = \frac{1}{M}\sum_{j=1}^{M} logQAF_{t_j}$. This is the log of the absolute frequency of occurrence for term t_j in the query averaged over the M terms in common between the query and the document or document component. The coefficient c_1 used in the current version of the Cheshire II system is 1.269.

$X_2 = \sqrt{QL}$. This is square root of the query length (i.e., the number of terms in the query disregarding stopwords). The c_2 coefficient used is -0.310.

$X_3 = \frac{1}{M}\sum_{j=1}^{M} logDAF_{t_j}$. This is is the log of the absolute frequency of occurrence for term t_j in the document (or component) averaged over the M common terms. The c_3 coefficient used is 0.679.

$X_4 = \sqrt{DL}$. This is square root of the document or component length. In Cheshire II the raw size of the document or component in bytes is used for the document length. The c_4 coefficient used is -0.0674.

$X_5 = \frac{1}{M}\sum_{j=1}^{M} logIDF_{t_j}$. This is is the log of the *inverse document frequency*(IDF) for term t_j in the document averaged over the M common terms. IDF is calculated as the total number of documents or components in the database, divided by the number of documents or components that contain term t_j The c_5 coefficient used is 0.223.

$X_6 = logM$. This is the log of the number of terms that are in both the query and in the document or component. The c_6 coefficient used in Cheshire II is 2.01.

These coefficients and elements of the ranking algorithm have proven to be quite robust and useful across a broad range of document and component types.

The system also supports searches combining probabilistic and (strict) Boolean elements, as well as the FUZZY, RESTRICT and MERGE operators discussed above. Although strict Boolean operators and probabilistic searches are implemented within a single process, using the same inverted file structures, they really function as two parallel *logical* search engines. Each logical search engine produces a set of retrieved documents. When a only one type of search strategy is used then the result is either a probabilistically ranked set or an unranked Boolean result set (these can also be sorted). When both are used in a single query, combined probabilistic and Boolean search results are evaluated using the assumption that the Boolean retrieved set has an estimated $P(R \mid Q_{bool}, D) = 1.0$ for each document in the set, and 0 for the rest of the collection. The final estimate for the probability of relevance used for ranking the results of a search combining strict Boolean and probabilistic strategies is simply:

$$P(R \mid Q, D) = P(R \mid Q_{bool}, D)P(R \mid Q_{prob}, D)$$

where $P(R \mid Q_{prob}, D)$ is the probability estimate from the probabilistic portion of the search, and $P(R \mid Q_{bool}, D)$ the estimate from the Boolean. This has the effect of restricting the results to those items that match the Boolean portion, with ordering based on the probabilistic portion.

2.2 New Fusion Operators

The runs submitted this year make use of a set of new operators for the Cheshire II system. These are the FUZZY, RESTRICT and MERGE operators. Fuzzy operators are versions of the Boolean operators that are less "strict" than the conventional Boolean operators, applied to weighted result lists. In place of Boolean AND, the "!FUZZY_AND" operator takes the mean of the two weights in the result sets for the same record (this differs from the conventional fuzzy AND that take the minimum of the two weight). The "!FUZZY_OR" takes the largest of the two weights for the same record. "!FUZZY_NOT" currently behaves the same way as strict Boolean "NOT". Otherwise these operators are used the same way as the strict Boolean operators.

The "!RESTRICT_TO" and "!RESTRICT_FROM" operators take either a component result and a document result, or two component results (where one component contains the other). As discussed in [7], "components" in the Cheshire II system can be the contents of any tag (or of a set of tags) that are treated as separate documents for the purposes of indexing and retrieval. In the case of component and document results the component list is restricted to components that are in the document result – the matching components only are returned retaining their weight from the original component result. When two nested component results are used with these operators the result is larger components that include one or more of the smaller components. (Note that with component and document results !RESTRICT_TO and !RESTRICT_FROM may be used interchangibly and the type of operation to be performed is determined by the nature of the result sets, but with two component results the nesting of the elements must be taken into account in constructing the query (i.e, Parent_set !RESTRICT_FROM Child_set or Child_set !RESTRICT_TO Parent_set). Naturally Parent and Child can be any sub-query that results in the appropriate kind of component.

The !MERGE_SUM operator combines the two resultsets (like a Boolean OR) but adds the weights (actually the resulting raw ranking adds 1 + a probabilistic result and 1.5 for boolean results with matching document or component ids in both lists, and the original value for items found only in a single result). Note that !MERGE_SUM weights may exceed 1 and are not probabilities.

The !MERGE_MEAN operator combines the two resultsets (like a Boolean OR) but takes the MEAN (or average) of the weights from items in both lists and half of the weight of items in only a single list. This is the (currently) recommended operator for merging probabilistic resultsets.

The !MERGE_NORM operator combines the two resultsets (like !MERGE_MEAN) but it normalizes the weights using MIN_MAX normalization before it takes the MEAN of the weights from items in both lists and half of the weight of items in only a single list. This was where (one) bug occurred in the official runs, because items from a single list were neither normalized or divided in half. This implications of this bug, was that for our official runs using MERGE_NORM, items that occurred in only a *single* result set, among the many partial results merged for each of the queries, were likely to receive *higher* weights in the ranked results than items that occurred in many (or all) of the partial results.

With the new fusion operators described above, the system has available an entire spectrum of combination methods ranging from strict Boolean operations to fuzzy Boolean and normalized mean scores for probabilistic and Boolean results. The motivation for the new operators follows from the basic observation that has driven all research into data fusion methods in IR, that no single retrieval algorithm has been consistently proven to be better than any other algorithm for all types of searches. By providing a set of operators for combining the retrieved sets from different search strategies, we are hoping to capitalize the strengths of particular algorithms while reducing their limitations. In general, assumption behind any implementation of data fusion is that the more evidence the system has about the relationship between a query and a document (including the sort of structural information about the documents found in the INEX queries), the more accurate it will be in predicting the probability that the document will satisfy the user's need. Other researchers have shown that additional information about the location and proximity of Boolean search terms can be used to provide a ranking score for a set of documents[6]. The inference net IR model has shown that the exact match Boolean retrieval status can be used as additional evidence of the probability of relevance in the context of a larger network of probabilistic evidence[14]. In the same way, we treat the set of documents resulting from the exact match Boolean query as a special case of a probabilistically ranked set, with each retrieved document having an equal rank.

2.3 Okapi Implementation

The version of the Okapi BM-25 algorithm implemented in the Cheshire II system is based on the description of the algorithm in Robertson[11], and in TREC notebook proceedings[12].

$$\sum_{T \in Q} w^{(1)} \frac{(k_1+1)tf}{K+tf} \frac{(k_3+1)qtf}{k_3+qtf}$$

Where:

Q is a query containing terms T

K is $k_1((1-b)+b \cdot dl/avdl)$

k_1, b and k_3 are parameters , usually 1.2, 0.75 and 7-1000

tf is the frequency of the term in a specific document

qtf is the frequency of the term in a topic from which Q was derived

dl and $avdl$ are the document length and the average document length measured in some convenient unit

$w^{(1)}$ is the Robertson-Sparck Jones weight:

$$w^{(1)} = \log \frac{(\frac{r+0.5}{R-r+0.5})}{(\frac{n-r+0.5}{N-n-R-r+0.5})}$$

Where, for a given query and a given term:

r is the number of relevant items indexed with a given term

R is the total number of relevant items for the query

n is the number of items indexed with a given term

N is the number of items in the collection

The reason for adopting the Okapi BM-25 algorithm was twofold. First we wanted to provide to users of the Cheshire II system one of the most widely used and best performing algorithms. Secondly, we wanted to be able to do comparative testing and evaluation using this algorithm. The documents weights generated by the Okapi algorithm are on a quite different scale from those generated by the logistic regression algorithm, and they result in subtle differences in the final ranks of documents for the same database and query. Our current implementation uses only the *a priori* version (i.e., without relevance information) of the Robertson-Sparck Jones weights, and thus $w^{(1)}$ is effectively just an IDF weighting. However, because the results from this implementation of Okapi and the Logistic Regression were sufficiently different, they seemed to offer the kind of conditions where data fusion has been shown to be be most effective [10, 1].

3. INEX APPROACH

Our approach in INEX was to use all of the original and new features of the cheshire system in generating the results submitted for our official runs. This section will describe the indexing process and indexes used, and also discuss the scripts used for search processing. The basic database was unchanged from last year's. We did, however, create and use a number of additional indexes and performed a complete reindexing of the INEX document collection. This section will first describe the indexes and component definitions created for INEX 2003.

3.1 Indexing the INEX Database

All indexing in the Cheshire II system is controlled by an SGML Configuration file which describes the database to be created. This configuration file is subsequently used in search processing to control the mapping of search command index names (or Z39.50 numeric attributes representing particular types of bibliographic data) to the physical index files used and also to associated component indexes with particular components and documents.

As noted above, any element or attribute may be indexed. In addition particular values for attributes of elements can be used to control selection of the elements to be added to the index. The configuration file entry for each index definition includes three attributes governing how the child text nodes of the (one or more) element paths specified for the index will be treated. These attributes are:

1. ACCESS: The index data structure used (all of the indexes for INEX used B-TREE indexes).

2. EXTRACT: The type of extraction of the data to be performed, the most common are KEYWORD, or EXACTKEY. EXACTKEY takes the text nodes as a string with order maintained for left-to-right key matching. KEYWORD takes individual tokens from the text node. In addition there is support for extraction of proximity information as well. We made use of proximity processing to manage phrase searches for INEX 2003. Some more specialized extraction methods include DATE and DATETIME extraction, INTEGER, FLOAT and DECIMAL extraction, as well as extraction methods for geographic coordinates.

3. NORMAL: The type of normalization applied to the data extracted from the text nodes. The most commonly used are STEM and NONE. STEM uses an enhanced version of the Porter stemmer, and NONE (in spite of the name) performs case-folding. Specialized normalization routines for different date, datetime and geographic coordinate formats can also be specified.

Each index can have its own specialized stopword list, so that, for example, corporate names have a different set of stopwords from document titles or personal names.

Most of the indexes used in INEX used KEYWORD or KEYWORD_PROXIMITY extraction and STEMming of the keyword tokens. Exceptions to this general rule were date elements (which were extracted using DATE extraction of the year only) and the names of authors which were extracted without stemming or stoplists to retain the full name.

Table 1 lists the document-level (/article) indexes created for INEX and the document elements from which the contents of those indexes were extracted. These indexes (with the addition of the topicshort index) are the same as those used last year. The major difference from last year is that the alltitles, kwd, title, topic, topicshort indexes were set up to support proximity searching.

As noted above the Cheshire system permits parts of the document subtree to be treated as separate documents with their own separate indexes. Tables 4 & 3 describe the XML components created for INEX and the component-level indexes that were created for them.

Table 4 shows the components and the path used to define them. The COMP_SECTION component consists of each identified section (<sec> ... </sec>) in all of the documents, permitting each individual section of a article to be retrieved separately. Similarly, each of the COMP_BIB, COMP_PARAS, and COMP_FIG components, respectively, treat each bibliographic reference (<bb> ... </bb>), paragraph (with all of the alternative paragraph elements shown in Table 4), and figure (<fig> ... </fig>) as individual documents that can be retrieved separately from the entire document.

Table 3 describes the XML component indexes created for the components described in Table 4. These indexes make

Name	Description	Contents
docno	Digital Object ID	//doi
pauthor	Author Names	//fm/au/snm //fm/au/fnm
title	Article Title	//fm/tig/atl

topic	Content Words	//fm/tig/atl
		//abs
		//bdy
		//bibl/bb/atl
		//app

topicshort	Content Words 2	//fm/tig/atl
		//abs
		//kwd
		//st

date	Date of Publication	//hdr2/yr
journal	Journal Title	//hdr1/ti
kwd	Article Keywords	//kwd
abstract	Article Abstract	//abs
author_seq	Author Seq.	//fm/au @sequence
bib_author _fnm	Bib Author Forename	//bb/au/fnm
bib_author _snm	Bib Author Surname	//bb/au/snm
fig	Figure Contents	//fig
ack	Acknowledgements	//ack
alltitles	All Title Elements	//atl, //st
affil	Author Affiliations	//fm/aff
fno	IEEE Article ID	//fno

Table 1: Cheshire Article-Level Indexes for INEX

individual sections (COMP_SECTION) of the INEX documents retrievable by their titles, or by any terms occurring in the section. These are also proximity indexes, so phrase searching is supported within the indexes. Bibliographic references in the articles (COMP_BIB) are made accessible by the author names, titles, and publication date of the individual bibliographic entry, with proximity searching supported for bibliography titles. Individual paragraphs (COMP_PARAS) are searchable by any of the terms in the paragraph, also with proximity searching. Individual figures (COMP_FIG) are indexed by their captions, and vitae (COMP_VITAE) are indexed by keywords within the text, with proximity support.

Almost all of these indexes and components were used during Berkeley's search evaluation runs of the 2003 INEX topics. The official submitted runs and scripts used in INEX are described in the next section.

3.2 INEX '03 Official Runs

Berkeley submitted six retrieval runs for INEX 2003, three CO runs and 3 SCAS runs. We did not submit any VCAS runs. This section describes the individual runs and general approach taken in creating the queries submitted against the INEX database and the scripts used to do the submission. The paragraphs below briefly describe Berkeley's INEX 2003 runs.

Berkeley_CO01 was an automatic CO run using only the topic's title and keyword sections. This run uses automatic query generation with Logistic regression matching combined with boolean phrase matching and MERGE_MEAN partial result combinations. Only article level results are returned in this run (definitely a mistake).

Name	Description	Contents
COMP_SECTION	Sections	//sec
COMP_BIB	Bib Entries	//bib/bibl/bb

COMP_PARAS	Paragraphs	//ilrj|//ip1|//ip2|
		//ip3|//ip4|//ip5|
		//item-none|//p|
		//p1|//p2|//p3|
		//tmath|//tf

COMP_FIG	Figures	//fig
COMP_VITAE	Vitae	//vt

Table 2: Cheshire Components for INEX

Component or Name	Description	Contents
COMP_SECTION		
sec_title	Section Title	//sec/st
sec_words	Section Words	//sec
COMP_BIB		
bib_author	Bib. Author	//au
bib_title	Bib. Title	//atl
bib_date	Bib. Date	//pdt/yr
COMP_PARAS		
para_words	Paragraph Words	*†
COMP_FIG		
fig_caption	Figure Caption	//fgc
COMP_VITAE		
vitae_words	Words from Vitae	//vt

Table 3: Cheshire Component Indexes for INEX †Includes all subelements of paragraph elements.

Berkeley_CO_Okapi was an automatic CO run using only the topic's title and keyword sections. This run uses also uses the same automatic query generation script as Berkeley_CO01 but employs the new implementation of the Okapi BM-25 algorithm for ranked search components, combined with Boolean elements for proximity and term restrictions. Results from multiple components where combined using MERGE_MEAN merging of results. RSV scores were normalized and multiple result sets combined to include Article-level, section-level and paragraph-level results.

Berkeley_CO_MergePrOk was an automatic CO run. This run uses automatic query generation with both Okapi BM-25 and Logistic regression retrieval algorithms combined using a score-normalized merging algorithm (MERGE_NORM). Results from multiple components where combined using MERGE_MEAN and MERGE_NORM merging of results. Separate retrieval of Articles, Sections and paragraphs were combined using score normalized merges of these results. Only Titles and keywords were used in generating the queries, which also included Boolean operations for proximity searching and "negated" terms. This run was intended to demonstrate the effectiveness of fusion of results from two different retrieval algorithms.

Berkeley_SCAS01 was an automatic SCAS run using only the title (XPath specification). This run uses automatic query generation with Logistic regression matching combined with boolean phrase matching and MERGE_MEAN partial result combinations FUZZY_AND and FUZZY_OR operators were used in combining AND and OR elements within an "about" predicate

Berkeley_SCAS_Okapi was a SCAS automatic run using only the topic title. This run uses automatic query generation from XPATH (like run 01) but uses Okapi BM-25 ranking instead of Logistic regression and uses normalized scores in merging results from different aspects of the queries. Results from multiple components where combined using MERGE_NORM merging of results.

Berkeley_SCAS_Okapi2 was another automatic SCAS run. This run also uses automatic query generation and is very similar to Berkeley_SCAS_Okapi. Results from multiple components where also combined using MERGE_NORM merging of results. The major difference is that different indexes were used (that included more of the text – i.e. the topic index vs. the topicshort index used in Berkeley_SCAS_Okapi) for some of the queries. Many of the queries will probably have identical results to the other okapi-based run.

No VCAS runs were submitted (although the SCAS runs should probably have been submitted for both, since path interpretation in matching was not completely strict).

3.2.1 General Script structure and contents

As noted in the overview of Cheshire II features, all of the Cheshire client programs are scriptable using Tcl or Python. For the INEX test runs we created scripts in the Tcl language that, in general, implemented the following sequence of operations:

1. Read and parse topics
2. Extract search elements and generate queries
   (a) Extract query type , title (and it's XPath specification when provided), and keywords.
   (b) Duplicate British spellings in queries to include both British and U.S. spelling (e.g. "colour" becomes "colour color").
   (c) Identify quoted phrases and required (+) or unwanted (-) terms.
   (d) Based on the query type (CO or CAS):
      i. For CO-type queries, construct multiple partial searches including:
         A. Boolean search words and phrases using the topic or topicshort index for all terms from query title and keywords.
         B. Ranked search of words and phrases using the topic or topicshort index for all terms from query title and keywords.
         C. Ranked search of words and phrases using kwd index for all terms from query title.
         D. Probabilistic search of words and phrases using the alltitles index for all terms from query title.
         E. Boolean search of words and phrases using the alltitles index for all terms from query title.
         F. All of the above were combined in a single query using various merge operators.
      ii. For CAS-type queries, the XPath specification was used to choose the indexes (and components) to be searched, and the RESTRICT operators described above were used to validate proper nesting of components.
         A. For each specified "about" clause in the XPath, a merger of phase, keyword, Boolean and ranked retrieval was performed.
3. Submit queries and capture resultsets
   (a) Each query constructed in the previous steps is submitted to the system, and the resultsets with one or more matching documents are stored.
   (b) The requested document element(s) in the XPath specification are extracted from the top-ranked documents and added to the final resultset for the query.
   (c) In the Berkeley_CO_Okapi and Berkeley_CO_MergePrOk runs each of these queries (or rather variant forms of the query) were submitted separately for article-level, paragraph-level and section level results (using the appropriate components). In post-processing these separate sets were combined using an MIN_MAX normalization and re-ordering of scores, resulting list of the top-ranked 100 documents.
4. Convert resultsets to INEX result format. (E.g., extract matching element XPath's, ranks, and document file ids from top-ranked results and output the INEX XML result format for each)

```
((topic @+ Smalltalk !MERGE_NORM (topic @+
{Smalltalk Smalltalk Smalltalk Smalltalk}) )
!FUZZY_OR (topic @+ Lisp !MERGE_NORM (topic @+
{Lisp Lisp Lisp Lisp}) ) !FUZZY_OR (topic @+
Erlang !MERGE_NORM (topic @+ {Erlang Erlang Erlang
Erlang}) ) !FUZZY_OR (topic @+ Java !MERGE_NORM
(topic @+ {Java Java Java Java}) )) !RESTRICT_FROM
((sec_words @+ {"garbage collection" algorithm}
!MERGE_NORM (sec_words $garbage collection$)
!MERGE_NORM (sec_words @+ {$garbage collection$
$garbage collection$ $garbage collection$ $garbage
collection$} ) !MERGE_NORM (sec_words @+ {algorithm
algorithm algorithm algorithm}) ))
TARGETPATH = XML_ELEMENT_sec
```

Figure 1: Berkeley SCAS Query for Topic #68

Figures 1 and 2 show examples of queries generated by the scripts for SCAS and CO topics respectively. The use of "$" is to limit phrases specified in the topic statement. The duplication of terms is to enhance weighting for those terms.

4. EVALUATION

The summary average precision results for the runs described above are shown in Table

The "strict" quantization is intended to be similar to the relevance assessments used in other IR evaluations. (One could argue, however, that a closer approximation to most relevance judgements might be to consider any full document containing a 3 as "relevant", and possibly some of the 2's).

```
(topicshort @+ {"query tightening" "narrow the
search" "incremental query answering" query
tightening,search narrowing,incremental query
answering,knowledge base,concept similarities}
!MERGE_MEAN (topicshort {$query tightening$})
!MERGE_MEAN (topicshort {$narrow the search$})
!MERGE_MEAN (top icshort {$incremental query
answering$})) !MERGE_NORM (alltitles @+ {"query
tightening" "narrow the search" "incremental
query answering"} !MERGE_MEAN (alltitles {$query
tightening$}) !MERGE_MEAN (alltitles {$narrow the
search$}) !MERGE_MEAN (alltitles {$incremental
query answering$})) !MERGE_NORM (kwd @+ {"query
tightening" "narrow the search" "incremental query
answering"} !MERGE_MEAN (kwd {$query tightening$})
!MERGE_MEAN (kwd {$narrow the search$}) !MERGE_MEAN
(kwd {$incremental query answering$})) !MERGE_NORM
(topicshort @ {"query tightening" "narrow the
search" "incremental query answering" query
tightening,search narrowing, incremental query
answering,knowledge base,concept similarities}
!MERGE_MEAN (topicshort {$query tightening$})
!MERGE_MEAN (topicshort {$narrow the search$})
!  MERGE_MEAN (topicshort {$incremental query
answering$})) !MERGE_NORM (alltitles @ {"query
tightening" "narrow the search" "incremental
query answering"} !MERGE_MEAN (alltitles {$query
tightening$}) !MERGE_MEAN (alltitles {$narrow the
search$}) !MERGE_MEAN (alltitles {$incremental
query answering$})) !MERGE_NORM (kwd @ {"query
tightening" "narrow the search" "incremental query
answering"} !MERGE_MEAN (kwd {$query tightening$})
!MERGE_MEAN (kwd {$narrow the search$}) !MERGE_MEAN
(kwd {$incremental query answering$}))
TARGETPATH = XML_ELEMENT_article
```

Figure 2: Berkeley CO Query for Topic #92

Figure 3 and Figure 4 show, respectively the Recall/Precision curves for generalized quantization of each the SCAS and CO results of the officially submitted Berkeley runs. No Berkeley runs appeared in the top ten for all submitted runs. The issue that we were seeking to investigate (whether XML retrieval would benefit from data fusion methods operating across both elements and algoriths, has some cautious confirmation. The MergePrOK run which combined results for both logistic regression and Okapi algorithms showed a marked improvement over the Okapi run alone. However The high-end precision in that run was less than in the Prob run, this may however be due to the bug described previously. In addition, it is likely that if the logistic regress algorithm run (Prob) had included section and paragraph elements, it would probably have had much better overall performance.

5. CONCLUSION

The INEX evaluation has once again proven very interesting, particularly as the first evaluation of the new fusion and resultset merging operators in the Cheshire system. As the above discussion shows, there remains considerable room for improvement of our results. We hope to present some "debugged" runs at the meeting.

Run Name	Short name	Avg Prec (strict)	Avg Prec (gen.)
Berkeley_CO01	Prob	0.0467	0.0175
Berkeley_CO_Okapi	Okapi	0.0318	0.0314
Berkeley_CO_MergePrOk	MergePrOK	0.0546	0.0557
Berkeley_SCAS01	Prob_SCAS	0.1970	0.1545
Berkeley_SCAS_Okapi	Okapi_SCAS	0.0865	0.0682
Berkeley_SCAS_Okapi2	Okapi2_SCAS	0.0869	0.0687

Table 4: Cheshire Components for INEX

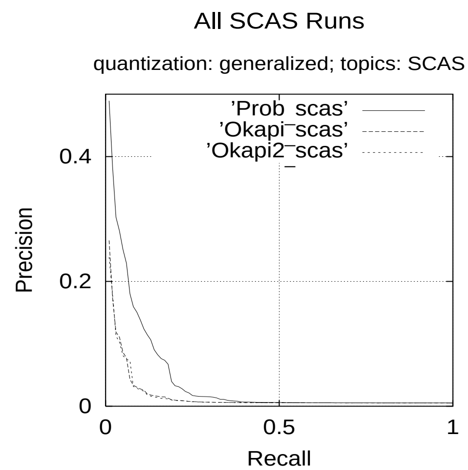


Figure 3: Berkeley SCAS Runs

6. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation and Joint Information Systems Committee(U.K) under *NSF International Digital Libraries Program* award #IIS-9975164.

7. REFERENCES

[1] S. M. Beitzel, E. C. Jensen, A. Chowdhury, O. Frieder, D. Grossman, and N. Goharian. Disproving the fusion hypothesis: An analysis of data fusion via effective information retrieval strategies. In *Proceedings of the 2003 SAC Conference*, pages 1–5, 2003.

[2] N. Belkin, P. B. Kantor, E. A. Fox, and J. A. Shaw. Combining the evidence of multiple query representations for information retrieval. *Information Processing and Management*, 31(3):431–448, 1995.

[3] W. S. Cooper, A. Chen, and F. C. Gey. Experiments in the probabilistic retrieval of full text documents. In D. K. Harman, editor, *Overview of the Third Text Retrieval Conference (TREC-3): (NIST Special Publication 500-225)*, Gaithersburg, MD, 1994. National Institute of Standards and Technology.

[4] W. S. Cooper, F. C. Gey, and A. Chen. Full text retrieval based on a probabilistic equation with coefficients fitted by logistic regression. In D. K. Harman, editor, *The Second Text Retrieval Conference (TREC-2) (NIST Special Publication 500-215)*, pages 57–66, Gaithersburg, MD, 1994. National Institute of Standards and Technology.

[5] W. S. Cooper, F. C. Gey, and D. P. Dabney. Probabilistic retrieval based on staged logistic regression. In *15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Copenhagen,*

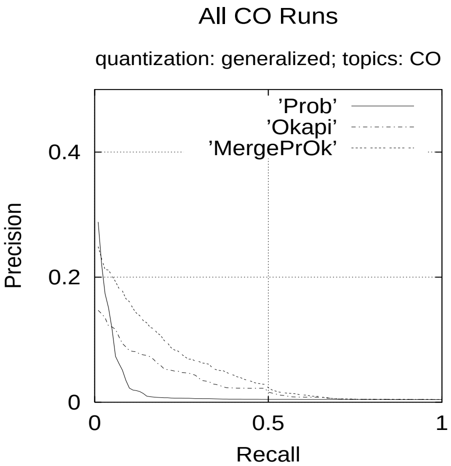


Figure 4: Berkeley CO Runs

Denmark, June 21-24, pages 198–210, New York, 1992. ACM.

[6] M. A. Hearst. Improving full-text precision on short queries using simple constraints. In *Proceedings of SDAIR '96, Las Vegas, NV, April 1996*, pages 59–68, Las Vegas, 1996. University of Nevada, Las Vegas.

[7] R. R. Larson. Cheshire II at INEX: Using a hybrid logistic regression and boolean model for XML retrieval. In *Proceedings of the First Annual Workshop of the Initiative for the Evaluation of XML retrieval (INEX)*, pages 18–25. DELOS workshop series, 2003.

[8] R. R. Larson and J. McDonough. Cheshire II at TREC 6: Interactive probabilistic retrieval. In D. Harman and E. Voorhees, editors, *TREC 6 Proceedings (Notebook)*, pages 405–415, Gaithersburg, MD, 1997. National Institute of Standards and Technology.

[9] R. R. Larson, J. McDonough, P. O'Leary, L. Kuntz, and R. Moon. Cheshire II: Designing a next-generation online catalog. *Journal of the American Society for Information Science*, 47(7):555–567, July 1996.

[10] J. H. Lee. Analyses of multiple evidence combination. In *SIGIR '97: Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 27-31, 1997, Philadelphia*, pages 267–276. ACM, 1997.

[11] S. E. Robertson and S. Walker. On relevance weights with little relevance information. In *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 16–24. ACM Press, 1997.

[12] S. E. Robertson, S. Walker, and M. M. Hancock-Beauliee. OKAPI at TREC-7: ad hoc, filtering, vlc and interactive track. In *Text Retrieval Conference (TREC-7), Nov. 9-1 1998 (Notebook)*, pages 152–164, 1998.

[13] J. A. Shaw and E. A. Fox. Combination of multiple searches. In *Proceedings of the 2nd Text REtrieval Conference (TREC-2), National Institute of Standards and Technology Special Publication 500-215*, pages 243–252, 1994.

[14] H. Turtle and W. B. Croft. Inference networks for document retrieval. In J.-L. Vidick, editor, *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, pages 1–24, New York, 1990. Association for Computing Machinery, ACM.

Identifying and Ranking Relevant Document Elements

Andrew Trotman and Richard A. O'Keefe
Department of Computer Science
University of Otago
Dunedin, New Zealand

andrew@cs.otago.ac.nz, ok@otago.ac.nz

1. ABSTRACT

A method of indexing and searching structured documents for element retrieval is discussed. Documents are indexed using a modified inverted file retrieval system. Modified postings include pointers into a collection-wide document structure tree (the corpus tree) describing the structure of every document in the collection.

Retrieval topics are converted into Boolean queries. Queries are used to identify relevant documents. Documents are then ranked using Okapi BM25 and finally relevant elements are identified using coverage. Search results are presented sorted first by document then coverage.

The design is presented in the context of the second annual INEX workshop.

2. INTRODUCTION

Otago first entered INEX [2] during its second year. There were three objectives: understand the participation process, gain access to this and last year's judgments, and create a baseline for comparing future experiments.

Participation involved design of six topics, generation and submission of search results, and online judging of three topics. Of these, generating the results was the most problematic as it required software changes.

The chosen retrieval engine was designed from the onset for retrieval of whole academic documents in XML [1]. A predecessor can be seen on BioMedNet and ChemWeb [4]. This engine, like that used in the IEEE digital library, returns relevance ranked lists of whole documents – the natural (citable) unit of information in an academic environment. From experience, information vendors are not interested in converting their documents from propriety DTDs into a common DTD or any other format – so software was needed to handle documents in heterogeneous formats.

Boolean searching, field restricting and relevance ranking were already supported, so modifications focused on identifying and ranking document elements. The modified retrieval engine can be thought of as working in three parts. Candidate documents are identified using a Boolean query. Candidates are then ranked using Okapi BM25 [6]. Finally, relevant non-overlapping elements are identified and presented as the result. Although it is easier to understand in three parts, in fact the most relevant elements of the most relevant documents are computed in a single pass of the indexes.

3. INDEXING

Much of the index design has already been described elsewhere [7]. Inverted file retrieval is used. There is one dictionary file and each dictionary term points to a single inverted list of postings.

An unstructured inverted list is usually represented $\{<d_1, f_1>, <d_2, f_2>, \ldots, <d_n, f_n>\}$ where d_n is a document ordinal number and f_n is the frequency of the given term in the given document. For structured retrieval, each $<d_n, f_n>$ pair is replaced by the triple $<d_n, p_n, f_n>$, where p_n is a position in the document. When phrase or proximity searching is required, this triple is replaced with the triple $<d_n, p_n, w_n>$ where w_n is the ordinal number of the term in the collection (starting from 0 at the start of the collection, incrementing by 1 for each term, not incrementing for tags, and not reset at the beginning of each record). On disk the postings are stored compressed.

The p_n value in each posting is a position in the corpus tree. The tagging structure for any one document represents a tree walk. Start at the root of the tree. When an open tag is encountered, the branch labelled with the tag name is followed downwards. When a close tag is encountered, the walk backtracks one branch. For a well-formed XML document, the walk will start and end at the root. This tree-walking property also holds for a collection of well-formed documents. The tree they collectively describe is called the corpus tree and can be built during single pass indexing. As each node is encountered for the first time, a branch is added to the tree and labelled with a unique ordinal identifier, p_n. Terms can lie either at the nodes or the leaves of this tree.

The corpus tree includes every single path in every single document, but is unlikely to match the structure of any one document. In Figure 1, three well-formed documents are given, as is the corpus tree for those documents. For clarity, the branches of the tree are labelled with which document they describe although this information is not computed and not stored.

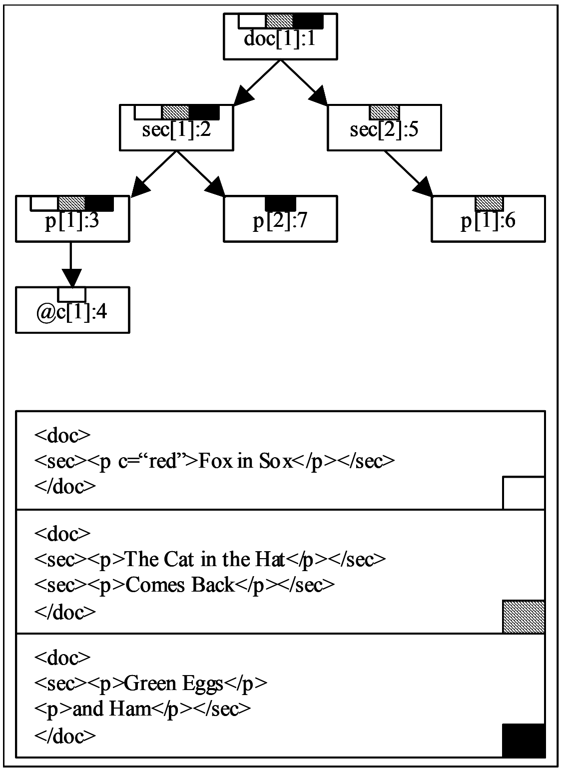


Figure 1: Three documents and the corpus tree including every path through every document, but not matching the structure of any one document. For the purpose of this figure each document is marked white, gray, or black and each node with which documents include that path. Each node is numbered with the instance of the tag (e.g. p[2]) and the node id, p_n (after the colon).

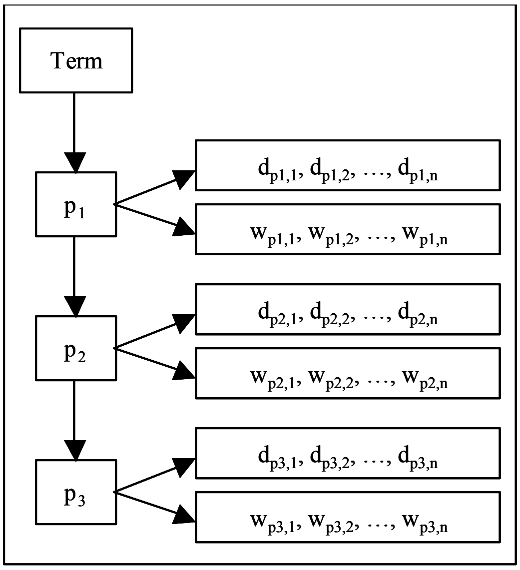


Figure 2: The in-memory postings structure allows quick access to only those postings relevant to the required document elements.

The inverted lists are built and processed using the structure represented in Figure 2. Postings for each term are ordered by increasing p_n. Each p_n points to the list of document ids (the *d*-sublist) and word ids (the *w*-sublist) found at that point in the tree. Each list is held in increasing order and compressed.

To search the collection for a given term, each *d*-sublist is examined in turn. By doing so, documents may not be examined in turn. This does not matter so long as all documents that would be examined are examined. Further, whole documents may not be examined in turn – this, too, does not matter as many ranking functions can be computed piecewise[1]. To field-restrict a term, a restricted set of sublists is examined. The *w*-sublists are used for proximity searching.

Storing and processing the postings in this way has computational advantages. For a field-restricted search, postings not pertaining to the restriction can be skipped. As postings are stored compressed, they need not even be decompressed. Word postings are used only for proximity searching. On disk the *w*-sublists are collected together and stored after all *d*-sublists. They are not even loaded from disk if not needed.

4. SEARCHING

As the retrieval engine starts up, the corpus tree is loaded and an additional structure is created from it, the field list. For each instance of each tag, the list of nodes at or below that node is collected. For each tag, the same is collected. These lists are then merged and sorted.

Table 1: The field list for the corpus tree given in Figure 1.

Field	Restriction
@c	{4}
@c[1]	{4}
doc	{1, 2, 3, 4, 5, 6, 7}
doc[1]	{1, 2, 3, 4, 5, 6, 7}
p	{3, 4, 6, 7}
p[1]	{3, 4, 6}
p[2]	{7}
sec	{2, 3, 4, 5, 6, 7}
sec[1]	{2, 3, 4, 7}
sec[2]	{5, 6}

The field list for the Figure 1 corpus tree is given in Table 1. From this, a search restricted to 'sec' requires postings at or below all 'sec' nodes of the

[1] BM25 cannot, so the lists are merged then processed.

corpus tree, or where $p_n=\{2, 3, 4, 5, 6, 7\}$. To search in ‘p[1]’, the postings are needed where $p_n=\{3, 4, 6\}$. For a search restricted to ‘p[1] in sec’, these two lists are ANDed together (giving $p_n=\{3, 4, 6\}$), and the members of this list are checked against the corpus tree to ensure they satisfy ‘p[1] in sec’ and not ‘sec in p[1]’.

Equivalence tag restrictions are also computed from the field list. The restrictions for each equivalent tag are ORed giving the equivalent restriction. If, for example, ‘p[2]’ and ‘@c’ were equivalent in Table 1, the restriction would be $p_n=\{4, 7\}$.

Several extensions were added to support element and attribute retrieval.

Attributes are now distinguished from tags by prefixing attributes with an @ symbol. This symbol was chosen because it makes for easy parsing of INEX queries, which use the same symbol.

The attribute value is considered to be content lying not only within the attribute, but also the tag. For example, “<tag att=“number”> term </tag>”, is equivalent to “<tag> <@att> number </@att> term </tag>”. In this way, a search for “number in tag” will succeed.

Tags can now be identified not only by their name and path, but also by the tag instance. Where before it was only possible to restrict to paragraph for example, it is now possible to restrict to the second paragraph.

Trotman [7] suggests the corpus tree will be small for real data. In this extended model this no longer holds true. In the TREC [3] Wall Street Journal collection there are only 20 nodes, for INEX there are 198,041 nodes after ‘noise’ nodes are removed (4,789 with attributes and instances also removed).

Table 2: Tags ignored during indexing.

ariel	en	item-text	ss
art	entry	label	stanza
b	enum	large	sub
bi	f	li	super
bq	it	line	tbody
bu	item	math	tf
bui	item-bold	proof	tfoot
cen	item-both	rm	tgroup
colspec	item-bullet	rom	thead
couplet	item-diamond	row	theorem
dd	item-letpara	scp	tmath
ddhd	item-mdash	sgmlf	tt
dt	item-numpara	sgmlmath	u
dthd	item-roman	spanspec	ub

Many tags are used to mark elements too small to be relevant. An example of such a tag is ‘ref’, used to mark references in the text. This tag cannot be relevant to any topic as the contents are simply reference numbers. Some tags were used for visual appearance such as ‘b’ used to mark text in bold. Others were used as typesetting hints such as ‘art’ used to specify the size of an image. If any of these tags, or those in Table 2 were encountered during indexing, tagging was ignored (until the matching close tag), but the content still indexed.

5. QUERY FORMATION

The title of the topic is extracted and converted into a Boolean query. This query is used to determine which documents to retrieve. Ranking is computed from the postings for the search terms.

For content and structure (CAS) topics, the target element is computed and stored for later use. The complete path for each about-function is computed by concatenating the about-path to the context-element restricting it. All equivalent paths are then computed by permuting this path with the equivalence tags. This fully specified path now replaces the original about-path and the context-element is removed.

At this point, the topic has been transformed from INEX topic syntax into a query whereby each about-function is Boolean separated and explicitly field restricted.

Create mandatory by ANDing each mandatory term (+)
Create optional by ORing each optional term
Create exclusion by ORing each exclusion term (-)

If all three sub-expressions are non-null, combine:
    mandatory AND (* OR optional) NOT exclusion

If two sub-expressions are non-null, combine using one of:
    mandatory AND (* OR optional)
    optional NOT exclusion
    mandatory NOT exclusion

If only one sub-expression is non-null, use one of:
    mandatory
    optional
    * NOT exclusion

Where ‘*’ finds all documents

Figure 3: Algorithm to convert an about phrase into a Boolean expression.

Examining the about-string, optional, mandatory (+), and exclusion (-) terms are allowed. These terms are converted into a Boolean expression. Optional terms are collected and converted into a sub-expression by ORing (“a b c” → “a OR b OR c”). Likewise, exclusion terms are also ORed. Mandatory terms are collected and ANDed (“+d +e

+f" → "d AND e AND f"). These three sub-expressions are then combined to form a complete about-query. The whole algorithm is presented in figure 3.

Separate about functions are already Boolean separated so these operators are preserved. Finally, all context-elements must be satisfied so these are ANDed together.

For content only (CO) topics, a Boolean expression is computed exactly as for one about-string using the algorithm presented in Figure 3.

6. RANKING

The retrieval engine is a Boolean ranking hybrid. Result sets are computed in two parts; a bit-string of documents satisfying the query, and a set of accumulators holding document weights.

The Boolean expression constructed above is converted into a parse tree then evaluated. At each leaf, the posting are loaded and converted into a bit-string, one bit per document. The bit-strings are then combined at the nodes of the parse tree using the operator there. At the root of the tree, the bit-string has set bits for all documents satisfying the query and unset for those that do not.

The accumulator values are the sum of Okapi BM25 scores computed at each leaf of the parse tree. Scores are summed regardless of the operators in the parse tree.

For AND and OR nodes scores are summed because the influence at these nodes is the sum of influences of the children.

For NOT nodes, they are also summed. If a document is excluded from the result set, the accumulator value is irrelevant. If a document is not, it is either re-included through other terms (e.g. mammal OR (dog NOT cat)), or there is a double negative in the query (e.g. cat NOT (dog NOT cat)). In both cases, the document has successfully satisfied a query leaf so receives a positive weight.

The Boolean ranking hybrid engine was extended to include ranking elements. Although whole documents are valid as results for CO topics, CAS topics specify a target element. This targeting establishes the retrieval unit. If the target element is 'sec', this tag must be returned. It essentially directs the retrieval engine to search and rank each given tag instance separately.

Wilkinson [8] suggests that ranking whole documents then extracting elements from these is a poor ranking strategy. The opposite may hold for this collection. A relevant element lies in the greater context of a relevant document. A relevant document will lie in a relevant journal, which, in turn, lies in a relevant collection. To this end, every paragraph of every section of every document is contextually placed so extracting elements from relevant documents may be a good approach.

6.1 Coverage

The coverage of any one posting is computed as those nodes in the corpus tree at or above the posting. Each posting is already annotated with a pointer into the tree, p_n. To compute the coverage, the tree is traversed upwards from p_n to the root. Coverage is computed for each document with respect to each search term.

For each document in the result set, the weighted coverage is computed as the covered branches of the document tree and how many search terms cover that branch. This is computed during a single pass of the indexes by storing the weighted coverage as part of each accumulator.

In any given document, the document root must have the highest weighted coverage, but this can be equal to that of other nodes. For CO topics, all branches of the document tree with coverage less than the root are pruned. The remaining leaves are presented as the result set for that document. In this way, the most information dense elements in the document are considered most relevant and no part of any document is returned more than once (overlapping is eliminated).

If a target element is specified in a CAS topic, all non-target branches are pruned. From the remains, those branches with the highest weighted coverage are presented as the result set for that document.

So, recall is determined by evaluation of the Boolean expression, documents are then ranked using Okapi BM25, and elements are selected by weighted coverage. As all the metrics needed for ranking are available at search time, the search and rank process is computed in a single pass of the postings.

7. RESULTS

Evaluation results are presented in Table 3.

Table 3: INEX performance measures

Strict	Precision	Rank
CO	0.0249	42nd
SCAS	0.1799	24th
CO-ng-o	0.2210	Unknown
CO-ng-s	0.2210	10th
Generalized	Precision	Rank
CO	0.0241	31st
SCAS	0.1214	28th
CO-ng-o	0.2889	9th
CO-ng-s	0.2889	4th

The retrieval engine performed badly using the INEX_EVAL measure. This is most likely because this measure treats each tag in a hierarchy as relevant but coverage eliminates overlapping tags – the measure is inappropriate for this retrieval technique.

Good results were shown when performance is measured using INEX_EVAL_NG. NG measures the ratio of relevant to irrelevant information returned. Coverage finds those parts of the document that contain most of the search terms. The correlation between information density and coverage is reflected in the result.

The results show the best performance when generalized quantization is used. This suggests the ordering of the results is not optimal for strict quantization – or the most relevant documents are not ranked before less relevant documents. This may be a consequence of sorting into document order before coverage order.

8. OTAGO AT INEX

The participation process involved the design and contribution of six topics. Of these, four were selected for inclusion in the final topic set. Otago was assigned three of these to assess. The assessment took three people one week each; this was one week per topic.

The retrieval engine described herein was used for designing the contributed topics. This was somewhat problematic as the topic parser was written at the same time the topics were being written, each with few examples.

From the final CAS topic set, 19 required corrections, corrections finally running to 12 rounds! This suggests the topic syntax is unnecessarily complex. See our further contribution [5] for a discussion on a possible language to use for future workshops.

The assessors were overburdened by the multitude of obviously irrelevant documents to assess. Examining some of these documents suggests many retrieval engines were aiming at high recall by retrieving any document containing any of the title terms. In particular, the word 'java' appeared in one topic; this was a somewhat popular research area over the years included in the IEEE collection. The assessment task could be reduced by carefully designing topics (and retrieval engines) to avoid this problem.

9. CONCLUSIONS

Element ranking was added to a Boolean ranking hybrid retrieval engine. Relevant documents were identified using Boolean searching. Documents were ranked using Okapi BM25. Finally coverage was used to rank elements within documents.

The results suggest coverage is a good method of identifying relevant and non-overlapping elements. Performance was best for generalized quantization, so ordering is not ideal. This may be a consequence of presenting results in document order.

10. ACKNOWLEDGEMENTS

In addition to the authors, Yerin Yoo contributed to the assessment task. Without her contribution we would not have been able to complete the task.

This work was supported by University of Otago Research Grant (UORG) funding.

11. REFERENCES

[1] Bray, T., Paoli, J., & Sperberg-McQueen, C. (1988). Extensible markup language (XML) 1.0, W3C recommendation. Available: http://www.w3.org/TR/REC-xml.

[2] Fuhr, N., Gövert, N., Kazai, G., & Lalmas, M. (2002). INEX: Initiative for the evaluation of XML retrieval. In *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*.

[3] Harman, D. (1993). Overview of the first TREC conference. In *Proceedings of the 16th ACM SIGIR Conference on Information Retrieval*, (pp. 36-47).

[4] Hitchcock, S., Quek, F., Carr, L., Hall, W., Witbrock, A., & Tarr, I. (1988). Towards universal linking for electronic journals. *Serials Review,* 24(1), 21-33.

[5] O'Keefe, R. A., & Trotman, A. (2003). The simplest query language that could possibly work. In *Proceedings of the 2nd Workshop of the initiative for the evaluation of XML retrieval (INEX)*. (to appear)

[6] Robertson, S. E., Walker, S., Beaulieu, M. M., Gatford, M., & Payne, A. (1995). Okapi at TREC-4. In *Proceedings of the 4th Text REtrieval Conference (TREC-4)*, (pp. 73-96).

[7] Trotman, A. (2003). Searching structured documents. *Information Processing & Management,* (to appear) doi:10.1016/S0306-4573(03)00041-4, available on ScienceDirect since 6 June 2003.

[8] Wilkinson, R. (1994). Effective retrieval of structured documents. In *Proceedings of the 17th ACM SIGIR Conference on Information Retrieval*, (pp. 311-317).

The Simplest Query Language That Could Possibly Work

Richard A. O'Keefe
Department of Computer Science
University of Otago
Dunedin, New Zealand
ok@cs.otago.ac.nz

Andrew Trotman
Department of Computer Science
University of Otago
Dunedin, New Zealand
andrew@cs.otago.ac.nz

ABSTRACT
The INEX'03 query language proved to be much too complicated for the INEX participants to use well, let alone anyone else. We need something simpler, but not too simple. Something which is basically a hybrid between Boolean IR queries and a stripped down CSS will do the job.

1. INEX NEEDS A QUERY LANGUAGE.
In the INEX conferences, we are trying to develop a data collection and a set of queries with known answers that can provide a solid basis for research and experimentation with XML information retrieval.

In order to communicate between researchers in the same year, we need a common query language. For INEX'02 there was such a language. In INEX'03 there was another. In order to communicate between the researchers who produce the queries in one year and the researchers who use them in later years, we need a stable, well-defined language.

The designer(s) of the INEX'03 query language had every reason to feel pleased. After the INEX'02 query language proved to need revision, surely this was the simplest thing that could possibly work: take an extremely well established XML structural query language (XPath) and add to it a minimal set of features for Information Retrieval.

It seems to be agreed that XPath is not a language for the casual user. But this paper is not concerned with user query languages. The query language we need is a query language for use by researchers who are expert in information retrieval and XML. What counts is whether the query language is suitable *for us*, not users.

Unfortunately, the production of this year's queries proved conclusively that the INEX'03 query language is far too complicated *for us*:

- It proved too hard to use. Of the 30 CAS queries that were selected, 19 (nearly $\frac{2}{3}$), were either syntactically illegal or otherwise wrong. It took no fewer than 12 rounds of correction before we had a completed collection of queries.
- Like many W3C productions, XPath is quirky, to put it kindly. It is very powerful in some respects, but there are queries that are very hard to express. For example, *//body//ip1//name* | *//body//ip2//name* is legal, but *//body//(ip1|ip2)//name* is not.
- It proved to be hard to implement. Presumably everyone who submitted a query for consideration had already checked it with some XML IR engine; how else could they have known that the query had about the right number of relevant answers? Yet a large number of queries were syntactically or semantically wrong. That should have been noticed. At least one implementor switched the semantics of the / and // operators.
- It proved to be hard to implement for another reason. XPath is quite powerful, in ways that are not likely to be useful for information retrieval, and yet if XPath was not implemented in full, were we really implementing the INEX'03 query language? This year, it turned out that most of the power of XPath was not needed. It wasn't the simplest thing that could possibly have worked. For example, we[23] found that there were 198,041 nodes in the index after ignoring "noise" tags. Yet if ordinal position was also ignored, there were only 10,522 distinct paths. Not one of this year's selected CAS queries used the ordinal position ($[n]$) feature of XPath.
- XPath has a clear definition of the "string value" of a node; the definition is precise, but given the actual XML markup in the document collection we are working with, it's not the definition we want. For example, if there is one mention of Joe Bloggs in the collection, as $\langle au\rangle\langle fnm\rangle Joe\langle /fnm\rangle\langle snm\rangle Bloggs\langle /snm\rangle\langle /au\rangle$, then the string value is "JoeBloggs" and a search for the word "Bloggs" is guaranteed to miss it.

  Worse, markup that is supposed to enclose numbers very commonly includes punctuation as well; the rules of XPath say that trying to convert such a string value to numeric form is an error. Yet we want to query it.

2. THE INEX'03 QUERY LANGUAGE WAS TOO HARD TO USE.
Every group had to submit 3 CAS and 3 CO queries. These submissions were supposed to have been tested, and known to have a reasonable number (not too high, not too low) of relevant answers. In fact, some answers were provided with each submission. So each submitted query should have been a legal INEX'03 query.

From this pool, 30 CAS and 36 CO queries were selected. Of the 30 CAS queries, 19 had either syntax errors or serious semantic errors. The most common semantic error was using the "child" operator / when the "descendant" operator // was intended.

This is a shocking error rate.

It wasn't just hard to get the queries right in the first place; it was hard to fix them. It took 12 rounds of corrections before we had a workable set of queries, starting from what were presumably the best queries in the first place.

3. WHAT SHOULD WE LOOK FOR IN A QUERY LANGUAGE?

3.1 We want something WE can use.

This paper is not about query interfaces or query languages for end users. This paper is solely concerned with query languages for *researchers* producing or using INEX data. Complexity is not necessarily a problem *for us*, as long as it is useful complexity. Requiring an intimate knowledge of XML or XML related technologies is not necessarily a problem *for us*. Requiring lots of punctuation in just the right places is not necessarily a problem *for us*.

While complexity need not be a problem, we need to take a step back and start with something much simpler than XPath, because it is an empirically established fact that it was too complicated *for us*. It is not likely that the query language we propose in this paper will serve for all time; what does matter is that it should be possible to automatically translate it into whatever richer language may be devised in the future. Simplicity now means easier conversion in the future. So one guiding rule is that nothing should be included in the query language unless it was actually used in this year's or last year's queries.

We do not want to limit INEX participation to experimenters following an "orthodox line" in query languages. Keeping the query language simple keeps the conference open to approaches with as yet unimagined index structures and retrieval techniques. XPath and XPath-like languages penalise such approaches.

3.2 Databases and information retrieval are different

It is useful to distinguish between *database* query languages and *information retrieval* query languages. They have some similarities, but the differences are fundamental, and mean that an XML database query language is unlikely to be a good foundation for an XML information retrieval query language.

The CODASYL database language, "network" databases, the relational algebra, the relational calculus, SQL, the Object Query Language (OQL) in the ODMG Object Database Standard[4], and various spatial and temporal extensions of relational databases, even the Smalltalk dialect used in Gemstone, all have these fundamental characteristics in common:

- To a large extent, as [9] puts it, this "data is primarily intended for computer, not human, consumption."
- A "database" is made up of elementary values (numbers, strings, dates, and so on) aggregated using a predefined set of container types with precise data structure semantics and labelled with user defined labels (column names, relation names, and so on).
- The user-defined labels have user-defined semantics which the database is aware of only to the extent that constraints are stated.
- Even when there are user-defined structures (classes in ODMG, Gemstone, and SQL3, for example), these may be seen as instances of one of a fixed set of meta-structures. For example, the ODMG standard provides an Object Interchange Format by means of which any object database may be dumped as a text stream; instances of classes all have a fixed format here and it is clear that "class" is a single meta-structure.
- There is a structured query language with a (more-or-less) formal definition which relates any legal query to a precise semantics, by appealing to the data structure semantics of the container types and meta-structures and to any stated constraints.
- A query processor is expected to obey the semantics of any query it accepts *precisely*; it may exploit known properties of the query language to transform a query into one with better performance, typically by using indexes.
- If a query has more than one answer, *all* of the answers are relevant. Someone who doesn't want all of the answers is expected to write a more specific query.

Database query languages are just like programming languages. (Very bad programming languages, some of them, notably SQL.) The person formulating the query is expected to understand the relevant user-defined labels and constraints and to "program" a query which expresses his or her needs. A database query engine is required to obey the query literally, just as a C compiler is required to translate C faithfully, even rubbish. If you ask an ODMG database the OQL query *select p from Persons p where p.address.city = "Dunedin"* and the answer includes a *p* for which *p.address.city = "Mosgiel"*, you will be seriously unhappy, even though Mosgiel is only 10 to 15 minutes' drive from Dunedin.

Since SGML was designed, the SGML slogan has been "a document is a database". For many years there have been SGML document database engines, notably SIM[16]. As XML is a special case of SGML, it is natural to view an XML document as a database.

- The elementary values are strings. The aggregates are labelled attributed tree structures. The data structure semantics is provided by GROVEs, or the DOM. Element type names and attribute names are the user defined labels.

- Constraints are stated by means of DTDs or XML Schemas. XML Schemas in particular express the notion "a database is a document".

What you get, on that view, is a database query language for tree-structured databases.

Information retrieval is very different. Instead of saying "the programmer knows precisely what s/he wants and how that's represented, I must do exactly what s/he says", information retrieval engines say "the user wants to find out about something and has given me a hint about what it is, I must be helpful". If you ask an information retrieval system *"agricultural research Dunedin"* and it comes back with a web page about *"Invermay Agricultural Centre, Mosgiel"*, you are not angry with it for disobeying you but impressed with how clever it was to find something so helpful.

The fact that information retrieval systems regard the user's query as a clue about what the user wants instead of a precise specification has enormous consequences for the design of information retrieval languages. So does the fact that the text they search is itself *not* in a precisely defined language.

When you construct a DTD or Schema for a family of XML documents, you describe how the XML parts fit together. But if you have free text in some of the elements, it remains just as informal as free text on its own.

At one end, we have data without a known precise semantics. At the other end, we have queries that are regarded as clues rather than commands. As Shlomo Geva[13] pointed out in the INEX mailing list, even the Boolean operators are not taken all that seriously by some retrieval engines. If two relational or object database engines holding the same information give different answers to a single query, at least one of them is broken. If two information retrieval engines holding the same document collection give different answers to a query, one of them might be better, but each of them might find something useful that the other doesn't. It certainly doesn't mean that either of them is wrong. All of this makes it hard to design elaborate information retrieval query languages. What earthly use is elaborate precise syntax when you don't have, can't have, and wouldn't want, precise semantics?

Of course we can embed a database query language in an IR query language (find precisely this set of documents and use that as a clue combined with the other clues in the query to find what I really want instead), and we can embed an IR query language in a database query language (give me precisely the answers satisfying a bunch of tests one of which is this clue about what I have in mind). Confusion seems unavoidable; at least we should be clear about which parts are precise and which parts are fuzzy.

3.3 It's all about indexes.

The great strength of Information Retrieval systems is their indexes.

An information retrieval language for XML should exploit this. It should avoid "structural" queries that are hard to handle with plausible index structures.

This does not mean that we should always be limited to queries that can be expressed in terms of currently known index structures. On the contrary, if someone comes across a reasonable query that is not expressible in the INEX'04 query language, that's a *good* thing, because it suggests a research topic: what kind of index could support this kind of query?

3.4 "Descendant" is more useful than "child".

An extremely common mistake in the INEX'03 queries was using the "child" axis (/) when the "descendant" axis (//) was intended.

The designers of CSS recognised that "descendant" queries were more common when they used the invisible operator to mean "descendant", making "descendant" easier to say than "child".

Consider *//article/bdy/sec/ip1*. That may be what you want, but you might have wanted *//article/bdy/sec/bq/ip1* elements as well, had you known about them. The query *//article//bdy//sec/ip1* is more likely to be what you really mean.

It turns out that *none* of the INEX'03 queries needs "child" at all; in each case "descendant" will do.

4. POPULAR XML QUERY LANGUAGES.

The world is awash in query languages for semistructured data, ranging from the complicated (CSS) to the mindbogglingly complicated (XQuery).

4.1 HyTime

HyTime[15, 14, 21] introduced many important things to SGML. One of them was a query language, HyQ[19].

However, the current standard says "HyTime recommends the use of the Standard Document Query Language (SDQL), defined in the DSSSL standard, ISO/IEC 10179:1996 Document Style Semantics and Specification Language, for the queryloc and nmquery element forms. The SDQL language includes equivalents of all the HyTime location address forms."

Early drafts of XPath looked like a stripped down HyQ.

HyQ is all about precise location of points and ranges both in trees and in multimedia coördinate systems. It is quite complicated. But it is worthy of note as one of the two ancestors of most XML query languages. (The other is SQL.)

4.2 DSSSL

DSSSL[17] is the SGML version of XSL and XSLT[6]. It contains a Scheme-based query and transformation language. It must be said that DSSSL is incomparably easier to read than XSLT. The Standard Document Query language is basically some datatypes for collections of nodes and some functions that manipulate them. It's a programming language, not an IR query language.

4.3 CSS

A CSS[3] $\langle$*selector*$\rangle$ is a collection of $\langle$*path*$\rangle$s or-ed together. In each $\langle$*path*$\rangle$, the focus is on the rightmost element; it is

Table 1: CSS grammar

⟨*selector*⟩	::=	⟨*path*⟩ {⟨*or*⟩ ⟨*path*⟩}*
⟨*or*⟩	::=	‘`,`’
⟨*path*⟩	::=	{⟨*siblings*⟩ ⟨*down*⟩}* ⟨*siblings*⟩
⟨*down*⟩	::=	‘>’ | *empty*
⟨*siblings*⟩	::=	{⟨*element*⟩ ⟨*followed-by*⟩}* ⟨*element*⟩
⟨*followed-by*⟩	::=	‘`+`’
⟨*element*⟩	::=	(⟨*name*⟩ | ⟨*any*⟩ | ⟨*filter*⟩) ⟨*filter*⟩*
⟨*any*⟩	::=	‘`*`’
⟨*filter*⟩	::=	⟨*exists*⟩|⟨*equals*⟩|⟨*word*⟩|⟨*prefix*⟩|⟨*first*⟩|⟨*lang*⟩
⟨*exists*⟩	::=	‘`[`’ ⟨*name*⟩ ‘`]`’
⟨*equals*⟩	::=	‘`[`’ ⟨*name*⟩ ‘`=`’ ⟨*value*⟩ ‘`]`’
⟨*word*⟩	::=	‘`[`’ ⟨*name*⟩ ‘∼=’ ⟨*value*⟩ ‘`]`’
⟨*prefix*⟩	::=	‘`[`’ ⟨*name*⟩ ‘|=’ ⟨*value*⟩ ‘`]`’
⟨*first*⟩	::=	‘`:first-child`’
⟨*lang*⟩	::=	‘`:lang(`’⟨*value*⟩‘`)`’

that element which the following style will be applied to. Working from right to left, an element must be a sibling (‘`+`’), a child (‘>’), or a descendant (invisible operator) of the element to its left.

An ⟨*element*⟩ test may check for an element ⟨*name*⟩ or not (⟨*any*⟩ or omitted). It may check whether an element is the ‘`:first-child`’ of its parent. This means that XPath’s /*[3 and p] is expressible as *:first-child+*+p. But XPath’s /p[3] is not quite expressible; p:first-child+p+p does not allow other elements between the *p* elements.

A ⟨*filter*⟩ may check whether an attribute is present, whether it is present and has normalised value exactly equal to a given text, whether it is present and contains a given white space delimited word, or whether it is present, looks like an xml:lang value, and has a given lang code as prefix. The grammar is given in Table 1.

There is no negation anywhere in CSS. You cannot test whether an attribute is present and *not* equal to a string. Paths cannot be negated. Within its limits, CSS seems quite usable.

4.4 XPath

This year’s query language was based on XPath 1.0. XPath 1.0 has several uses in W3C standards. One of them is XPointer. XPointer provides a means of pointing *precisely* to a location or range in a document. That is, XPointer, and the underlying XPath, are *database* query languages for XML.

We can get an idea of the complexity of various extensions and relatives of XPath by looking at the sizes of the defining reports; to master any of them requires reading at least this much material. Since the reports are provided in HTML, the page count depends on how you display it. Therefore we normalise the number of screens by the number of screens for XPath 1.0 in Table 2.

The “all up” entries include the Data Model and Functions and Operators documents, which are essential parts of XPath 2.0, XSLT 2.0, and XQuery 1.0. To get page counts for the browser and paper size we used, multiply by left column by about 28.

Table 2: Length of Specification (Normalised)

1.0	XPath 1.0[7]
0.7	XML-QL[10]
1.5	XQL[22]
3.2	XSLT 1.0[6]
4.2	XSLT 1.0 + XPath 1.0 (XSLT includes XPath)
2.4	XQuery 1.0 and XPath 2.0 Data Model[11]
5.8	XQuery 1.0 and XPath 2.0 Functions&Operators[20]
3.1	XPath 2.0[1]
11.3	XPath 2.0 all up
9.0	XQuery 1.0[2]
17.3	XQuery 1.0 all up
10.1	XSLT 2.0[18]
18.3	XSLT 2.0 all up

If XPath 1.0 was too complex for us to master, can any of the other W3C query languages be easier? XML-QL looks as though it might be, but it is not a W3C recommendation, and [9] explicitly says that “. . . we take a database view, as opposed to document view, of XML. We consider an XML document to be a database . . . ”.

In fact all of these languages take a database view, making them unsuitable as foundations for an information retrieval query language. Space does not permit thorough discussion of YATL[8], XQL[22], Quilt[5] (Quilt and XPath 1.0 are closely related), YATL[8], or others.

4.5 XIRQL

XIRQL[12] was designed as an “information retrieval” query language, not a “database” query language. However, it extends XQL, so parts of it resemble XPath, including the distinction between “child” and “descendant” which we failed to master. In the INEX collection, it was not clear to most of us what the root actually was, so the ability to refer to the root is not useful to us either.

The abstract of [12] tells is that XIRQL integrates “weighting and ranking, relevance-oriented search, datatypes with vague predicates, and semantic relativism ... by using ideas from logic-basic probabilistic IR models.” This means that important and attractive as XIRQL is, it is too closely tied to one particular approach to be suitable for INEX.

We propose a much simpler and less capable language, which can be seen as a very small sublanguage of XIRQL, and also of other query languages.

5. THE STRING-VALUE PROBLEM.

Practically everything in XPath 1.0 that involves strings is defined in terms of the "string-value" of a node. The rules are spelled out in section 5 of the XPath 1.0 specification. Roughly speaking,

1. The string-value of a text item (parsed character data or CDATA) is the obvious text value.
2. The string-value of an element or of the entire document is the concatenation of the string-values of its text descendants in document order.
3. The string-value of an attribute is its normalised value as spelled out in the XML 1.0 specification. (An XML processor that does not validate cannot be used as the basis for an XPath implementation.)

So ⟨*au*⟩⟨*fnm*⟩*Joe*⟨*/fnm*⟩⟨*snm*⟩*Bloggs*⟨*/snm*⟩⟨*/au*⟩ has string-value "*JoeBloggs*".

If you go looking for "*Bloggs*" in ⟨au⟩, XPath 1.0 guarantees you won't find it.

Of course, we don't have to follow XPath's definition of string-value. But if we don't do that, there isn't much point in following XPath's complex and limiting syntax either.

This definition of string value goes back to HyTime; every XML-related standard we've checked uses essentially the same definition. CSS and XSLT provide means for transforming a document by adding material at the beginning or end of an element's contents; the string value can be quite different in the transformed document. XPath was too hard; bringing XSLT into it would clearly be inadvisable.

There are three plausible ways around this problem.

- Add an extra space at the end of each text item. This gives the answer "`Joe␣Bloggs␣`", which will work. In rare cases like "⟨*u*⟩*A*⟨*/u*⟩*ccelerator*" this may break words up, but it will almost always help.
- For items which should be treated as having word breaks, add an attribute in the DTD:

  ```
  <!ATTLIST snm INEXword #FIXED "break">
  ```

  Ensure that there is at least one white space character at the boundaries of every element with INEXword="break".
- Allow the indexing software to make the decision just as it does for stemming. Attributes like `INEXword` offer guidance, not rigid command.

The first approach is simpler. If we were seeking the precision of database queries, the second approach would be better. Examples like T⟨scp⟩itle⟨/scp⟩ W⟨scp⟩ords⟨/scp⟩ may make it essential even for us (although the INEXscan attribute should solve this problem). But whichever approach we take, we are divorcing ourselves from XPath.

5.1 Numbers

An XML document contains only strings. Many of this year's queries involved numeric comparisons. That requires converting strings to numbers. XPath specifies precisely how that is done. (The rules are somewhat different in XPath 2.0, but do not affect the present point.)

The problem is that the INEX'03 document collection is a realistic collection of sloppily marked up text. There are elements such as ⟨*yr*⟩ which are supposed to contain numbers, but also contain punctuation marks and other junk. Trying to convert such a string to a number is an error in XPath. If we want to know whether $yr > 1999$, we do not want our query to be derailed by ⟨*yr*⟩`2000,␣`⟨*/yr*⟩, as it *must* be in XPath.

Not only do we need rules for converting text to numbers that are different from the rules in XPath, we need to interpret comparisons fuzzily. If you ask a database for a record with $date > 1999$ and it reports a record with $date = 1999$, that's an error. If you ask an information retrieval system for documents with $yr > 1999$ and it returns one with $yr = 1999$, that's not an error, it's just somewhat less relevant than one that matches the clue precisely.

6. ARCHITECTURAL FORMS

HyTime was really several interesting standards package together unintelligibly. One of the key features presented was the idea of "architectural forms" and of architectural form processing.

Basically, the idea is that a document may be marked up (and validated) according to one DTD, yet processed according to another (traditionally but confusingly called a) meta-DTD. Attributes in the source DTD say how to map the elements and attributes physically present to the ones that ought to be present according to the target DTD. A processing instruction with a special form is used to tell an architectural-form-aware processor which attributes to use for this purpose.

This may sound like XSLT, or, if you are into arcana, like linkage declarations in SGML. In fact it is something much simpler. Elements and attributes may be dropped, renamed, or copied as they are.

Why would you parse in one DTD and process according to another? You might have a formatter that can handle many structures, and a specialised DTD that is only intended to use some of the features. You might have a meta-DTD written using English words for markup, and Swedish users who would like to use Swedish words, so they validate against a DTD which uses Swedish words, but which uses architectural form processing to map to the English version. You might wish to make fine distinctions; for example you might want to use ⟨species⟩ and ⟨foreign⟩ tags in your markup, but they might both be simply mapped to ⟨italic⟩.

With the INEX collection, we have a collection of documents marked up for printing. Some of the distinctions made in the DTD are not important for information retrieval purposes. The INEX'03 rules took this into account. For example, ⟨ip1⟩, ⟨ip2⟩, ⟨ip3⟩, ⟨ip4⟩ were all to be treated by the query engine as equivalent to ⟨p⟩.

That's the wrong time to do it. It had the unpleasant consequence that you asked for p[n] the element you got could be p[m] with $m \neq n$.

It is not the queries which determine which tags are equivalent, but the DTD designer and document collector. The replacement of tags by equivalents should be done before the documents are indexed, so that the index and the query agree about what elements are which. That is just what architectural form processing can do for you.

We may not want to index some elements, either because they do not contain text or because the text is never useful. (We yearned mightily for some way to get rid of ⟨ref⟩ elements during evaluation. They should never have been returned in the first place.)

Some elements may be presentation markup which it is useful to ignore (see Table 2 in [23]). This is especially useful because these are the tags which spoil the simple "add a space after each element" rule for modified string-value. For example, given *⟨st⟩V⟨scp⟩OICE⟨/scp⟩ XML⟨/st⟩* we would like this to be treated as *⟨st⟩VOICE XML⟨/st⟩*. We want to ignore the tags of these elements, but not their contents.

In the spirit of architectural form processing, we can address these issues by adding attribute declarations in the DTD. XML allows us to add attribute declarations without changing the original ones, so xmlarticle.dtd could become

```
<!ENTITY old-dtd PUBLIC "..." "oldarticle.dtd">
%old-dtd;
<!ATTLIST ...>
...
<!ATTLIST ...>
```

with the original xmlarticle.dtd renamed to oldarticle.dtd. It is important that this can be done without touching the original DTD or the original XML files in any way.

The three attributes we want to add are

- INEXscan

  nothing do not index this tag or its descendants

  content do not index this tag; index its content

  element index this tag; do not index its content

  all index this tag and its content

  The evaluation tool should heed this attribute; it would materially reduce the labour of judging.

- INEXname

  if present, the name that is to be used in the index, and in queries, instead of the original element type.

- INEXatts

  a list of pairs of names: *attr* - means "do not index *@attr*, *attr alt* means "index *@attr* under the name *@alt* instead". If an attribute is not in the list, it is indexed as itself.

For example, we might have

```
<!ATTLIST ip1 INEXname NMTOKEN #FIXED "p">
<!ATTLIST ip2 INEXname NMTOKEN #FIXED "p">
<!ATTLIST ip3 INEXname NMTOKEN #FIXED "p">
<!ATTLIST ip4 INEXname NMTOKEN #FIXED "p">
<!ATTLIST scp INEXscan NMTOKEN #FIXED "content">
<!ATTLIST ref INEXscan NMTOKEN #FIXED "nothing">
```

The mapping can be handled by a trivial post-parser.

7. THE SIMPLEST THING THAT COULD POSSIBLY WORK.

The following query language was constructed to be just powerful enough to handle the queries people actually wrote. It clearly separates paths and text queries, allowing Boolean combinations of text queries but not of paths.

$$
\begin{array}{lll}
\langle \textit{topic} \rangle & ::= & \langle \textit{about} \rangle \\
 & | & \langle \textit{filtered-path} \rangle\ [\langle \textit{star} \rangle]\ \langle \textit{about} \rangle \\
 & | & \langle \textit{filtered-path} \rangle\ [\langle \textit{about} \rangle] \\
 & & \langle \textit{filtered-path} \rangle\ [\langle \textit{star} \rangle]\ \langle \textit{about} \rangle
\end{array}
$$

An ⟨*about*⟩ is basically a Boolean query plus context for the terms. A ⟨*filtered-path*⟩ describes a path in an XML document; the attributes of elements may be checked. There is no way of marking the "child" relation anywhere, or of specifying ordinal position.

If P and Q match ⟨*filtered-path*⟩ and A and B match ⟨*about*⟩, then A means "answer any elements that are about A"; PA means "answer any instances of P that are about A"; $PAQB$ means "for instances of P that are about A return instances of Q under that P which are about B"; and a missing A imposes no constraint.

⟨*star*⟩ ::= '`/`' '`*`'

A ⟨*star*⟩ may precede the final ⟨*about*⟩. This is to handle the queries which used `//*` in XPath. It means that once an instance of the preceding P or Q has been found, *any* descendant of that instance which fits the last ⟨*about*⟩ may be reported. Such descendants are of course subject to ranking in the same way as any others, elements which are too "dilute" should not be a problem.

⟨*filtered-path*⟩ ::= ⟨*filtered-elem*⟩ {'`/`' ⟨*filtered-elem*⟩}*
⟨*filtered-elem*⟩ ::= XML-name ⟨*filter*⟩*

An XML-name is any XML identifier, possibly including colons. The time to deal with namespaces will be when we have to. The '/' operator means "descendant", not "child". This is what most people expected '/' to mean in the INEX'03 query language.

⟨*filter*⟩ ::= '[' ⟨*attr-path*⟩ ⟨*range-list*⟩ ']'
⟨*range-list*⟩ ::= ⟨*range*⟩ {',' ⟨*range*⟩}*
⟨*range*⟩ ::= number ['..' [number]]
| '..' number
⟨*range*⟩ ::= [number] '..' [number]
⟨*attr-path*⟩ ::= ⟨*attr*⟩|⟨*simple-path*⟩
| ⟨*simple-path*⟩ ⟨*attr*⟩
⟨*attr*⟩ ::= '@' XML-name
⟨*simple-path*⟩ ::= XML-name {'/' XML-name}*

A filter compares text with a range of numbers. An ⟨*attr-path*⟩ is followed to find some text; the text may be the (modified) string value of an attribute or the (modified) string value of an element. Spaces and punctuation are trimmed from that modified string value; if the result can be converted to a number, the filter is satisfied to the degree that the number is in one of the ranges.

In a range $x..y$, x is the lower bound and y is the upper bound. It is an error if $x > y$. Missing x means $-\infty$; missing y means $+\infty$.

This query language does not use conventional notation like $<$ or $=$. There are two reasons for that. One is that these queries are supposed to be easy to express in XML, and XML makes it hard to use $<$. The second is that $<$ and $=$ are associated with precise meanings. But this is an information retrieval query language; a value which is not precisely in range may still be somewhat relevant. Since we don't intend the standard meaning of the mathematical signs, we shouldn't use them; it is important not to lie to the user.

⟨*about*⟩ ::= '(' ⟨*or-query*⟩ ')'
⟨*or-query*⟩ ::= ⟨*and-query*⟩ {'|' ⟨*and-query*⟩}*
⟨*and-query*⟩ ::= ⟨*not-query*⟩ {'&' ⟨*not-query*⟩}*
⟨*not-query*⟩ ::= ⟨*text-query*⟩ | '∼' ⟨*text-query*⟩

An IR engine may interpret these Boolean operators the way it would normally interpret any Boolean operators. The conventional precedence of the Boolean operators is followed. They need not be "precise", and although it is tempting to define algebraic identities for this query language, it would be inappropriate. The ampersand is also awkward to express in XML; some other spelling such as ';' could be allowed.

⟨*text-query*⟩ ::= ⟨*basic-query*⟩
| ⟨*basic-query*⟩ ':' ⟨*simple-path-list*⟩
⟨*basic-query*⟩ ::= {⟨*restriction*⟩ ⟨*term*⟩}+ | ⟨*about*⟩
⟨*term*⟩ ::= word | '"' word+ '"' | '' word+ ''
⟨*restriction*⟩ ::= empty | '+' | '-'
⟨*simple-path-list*⟩ ::= ⟨*simple-path*⟩{','⟨*simple-path-list*⟩}*

A text query may ask whether a basic query matches the current element, or whether it matches some descendant element. The commas in a simple path list mean "or" just as they do in CSS.

A word is an XML-name that doesn't include any dots, colons, or underscores, or is a pair of such names with an apostrophe in between, or is a number. A sequence of words between matching quotation marks is a phrase. The '+' and '-' restrictions have the same meaning as in the INEX'03 query language.

That's all there is to it. A parser for this language has been built using Lex and Yacc.

Several features that were considered but deliberately excluded:

- Filtering on anything other than a numeric range. In simple cases, this can be handled by the *PAQB* pattern. Complex cases haven't arisen. When they do, it will be important to be clear about whether we want precise matches, so that XHTML documents making extensive use of the "class" attribute could be handled, or information retrieval matches, in which case we could simply have [⟨*attr-path*⟩ ⟨*about*⟩].
- Any kind of language sensitivity. This is what the CSS '| =' predicate is for, and its ':lang" predicate. When the INEX collection includes mixed-language documents, we could perhaps use [:lang *word*].
- Any kind of position checks. It is easy enough to add syntax for this, just copy XPath. What's hard is to interpret it. In XPath, *bdy//p[2]* means "among the descendants of a ⟨bdy⟩ element take the second ⟨p⟩ whether it has the same parent as *p[1]* or not". To avoid this, we might take the INEX definition of *p[2]*, "the 2nd ⟨p⟩ child of some descendant of ⟨bdy⟩". Adapting [:first-child] from CSS would make more sense.
- Allowing any number of ⟨*path*⟩⟨*about*⟩ pairs. There's no difficulty in adding this, it just isn't needed.
- Allowing an axis other than "descendant". From a DTD, it is possible to compute a binary relation "can have child", the transitive closure of which is "can have proper descendant". This can be used to check the plausibility of queries. CSS also allows "child" and "sibling", which are similarly checkable. The complex mixing of axes in XPath makes it hard to check; we don't want to go there.

8. SOME SAMPLE INEX'03 QUERIES

Query 61 //article[about(.,'clustering +distributed') and about(.//sec,'java')]

⇒ article(clustering +distributed & java:sec)

Query 64 //article[about(./, 'hollerith')] //sec[about(./, 'DEHOMAG')]

⇒ article(hollerith) sec(DEHOMAG)

Query 66 /article[./fm//yr < '2000']//sec[about(.,'"search engines"')]

⇒ article[fm/yr ..1999] sec("search engines")

Query 68 //article[about(., '+Smalltalk') or about(., '+Lisp') or about(.,'+Erlang') or about(., '+Java')]// bdy//sec[about(., '+"garbage collection" +algorithm')]

⇒ article(+Smalltalk|+Lisp|+Erlang|+Java) bdy/sec(+"garbage collection" +algorithm)

Query 71 //article[about(.,'formal methods verify correctness aviation systems')]/bdy//*[about(.,'case study application model checking theorem proving')]

⇒ article(formal methods verify correctness aviation systems) bdy/*(case study application model checking theorem proving)

Query 76 //article[(./fm//yr='2000' OR ./fm//yr='1999') AND about(., '"intelligent transportation system"')]//sec[about(., 'automation +vehicle')]

⇒ article[fm/yr 1999..2000]("intelligent transportation system") sec(automation +vehicle)

Query 91 Internet traffic

⇒ (Internet traffic)

9. REFERENCES

[1] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) Version 2.0. W3C Working Draft, The World Wide Web Consortium, August 2003.

[2] S. Boag, D. Chamberlin, M. Fernández, D. Florescu, and J. Robie. XQuery 1.0: An XML Query Language. W3C Working Draft, The World Wide Web Consortium, November 2003.

[3] B. Bos, H. W. Lie, C. Lilley, and I. Jacobs. Cascading Style Sheets, level 2; CSS2 Specification. W3C Recommendation, The World Wide Web Consortium, May 1998.

[4] R. Cattell, D. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, January 2000.

[5] D. Chamberlin, J. Robie, and D. Florescu. Quilt: an XML query language for heterogeneous data sources. In *The World Wide Web and Databases: Third International Workshop WebDB 2000, Dallas, TX, USA, May 2000. Selected Papers*, number 1997 in Lecture Notes in Computer Science. Springer-Verlag, January 2001.

[6] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, The World Wide Web Consortium, November 1999.

[7] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, The World Wide Web Consortium, November 1999.

[8] S. Cluet, S. Jacqmin, and J. Simeon. The New YATL: Design and Specifications. Technical report, INRIA, 1999.

[9] A. Deutsch, M. Fernández, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. Technical report, AT&T Labs, 1998.

[10] A. Deutsch, M. Fernández, D. Florescu, A. Levy, and D. Suciu. A query Language for XML. W3C submission, The World Wide Web Consortium, August 1998.

[11] M. Fernándex, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 Data Model. W3C Working Draft, The World Wide Web Consortium, November 2003.

[12] N. Fuhr and K. Grosjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Research and Development in Information Retrieval*, pages 172–180, 2001.

[13] S. Geva. Re: Semantics of CAS Topics for the SCAS Task. E-mail to the INEX'03 participants, July 2003.

[14] C. F. Goldfarb. Hytime: A Standard for Structured Hypermedia Interchange. *IEEE Computer*, 24(8):81–84, August 1991.

[15] C. F. Goldfarb, S. R. Newcomb, W. E. Kimber, and P. J. Newcomb. *Information processing—Hypermedia/Time-based Structuring Language (HyTime)*. Number 10744:1997 in ISO/IEC. International Organization for Standardization (ISO), second edition, 1997.

[16] T. R. M. D. S. Group. The Structured Information Manager. Web Site, 2003. viewed in November 2003.

[17] ISO. *Document Style Semantics and Specification Language (DSSSL)*. Number 10179:1996 in ISO/IEC. International Organization for Standardization (ISO), 1996.

[18] M. Kay. XSL Transformations (XSLT) Version 2.0. W3C Working Draft, The World Wide Web Consortium, November 2003.

[19] W. E. Kimber. HyTime and Sgml: Understanding the HyTime HyQ Query Language. internal report, IBM, August 1993. findable on the Web.

[20] A. Malhotra, J. Melton, and N. Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Working Draft, The World Wide Web Consortium, November 2003.

[21] S. R. Newcomb, N. A. Kipp, and V. T. Newcomb. The HyTime Hypermedia/Time-Based Document Structuring Language. *Communications of the ACM*, November 1991.

[22] J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). W3C submission, The World Wide Web Consortium, 1998.

[23] A. Trotman and R. A. O'Keefe. Identifying and Ranking Relevant Document Elements. In *INEX '03*, 2003.

Helsinki's EXTIRP@INEX

[Draft]

Antoine Doucet* Lili Aunimo Miro Lehtonen Renaud Petit
Department of Computer Science
P. O. Box 26 (Teollisuuskatu 23)
FIN–00014 University of Helsinki
Finland
[antoine.doucet|lili.aunimo|miro.lehtonen|renaud.petit]@cs.Helsinki.FI

ABSTRACT
In this paper, we present EXTIRP[1], a novel XML retrieval system. It aims at retrieving exact coverage elements by decomposing the collection into a set of minimal retrieval units. With respect to a query, a similarity measure is computed for each of those units using an aggregation of relevance scores for a word features vector space model and for a novel maximal frequent sequences model. The similarity measure are finally propagated bottom-up, from each minimal unit to its ancestors. The system also includes query expansion, for which we may repeat the process several times.

1. INTRODUCTION
In this paper, we focus on the problem of finding an answer with an optimal coverage of the topic, given a non-structural query (CO topics in INEX). That is, we want to find a trade-off between replying to a query with a 15 pages article and replying a fragment that is not sufficient when deprived of its context.

In related work, it was tried to index every single element of the document collection, but modeling and computing an RSV (retrieval status value) for each element causes a clear problem w.r.t. efficiency. Our approach consists in defining minimal retrieval units, such that none of their subelements is big enough to be of interest by itself. An RSV is computed for each minimal unit, using word term features in the vector space model and maximal frequent phrases. The RSV values of the minimal units are finally propagated upwards to all their ancestors. One or more quey expansion steps can be iterated to form more relevant topic descriptions.

The technique for defining minimal retrieval units is presented in section 2. The two models we used are then presented (section 3), followed by the corresponding techniques to evaluate similarities within these models (section 4). In the following section, our query expansion technique is presented. The last section describing the system is number 6, where we present the method used to propagate RSVs upwards. We finally describe our runs in section 7 and conlude.

*This author is supported by the Academy of Finland (project 50959; DoReMi - Document Management, Information Retrieval and Text Mining)
[1]EXacT coverage IR based on static Passage clusters

2. PREPARATORY PROCEDURES
Finding the most relevant text documents for each given topic is the basic problem to be solved in traditional IR. But, as the document collection is in XML format, we can identify two additional challenges that must be overcome before any traditional IR methods can be applied. First, the document collection consists of 125 XML documents which alone are too big to be retrieved on their own. Therefore, the collection is divided into smaller XML units which we shall call *XML fragments*. Second, the XML fragments contain all the XML markup that is present in the original XML format. Our goal is to convert the XML fragments into a text-only format where all XML markup has been removed without losing any of the information that is implicitly or explicitly coded in the XML structure of the original documents.

2.1 Division of the collection
The division of the collection was performed at different levels of granularity. We will later refer to these subdivisions as Minimal Retrieval Units.

2.2 Structural conversion
The simplest solution is to strip all XML tags off, after which we only have textual content left. However, this method leads to the loss of crucial information.

3. MODELING DOCUMENTS AND QUERIES
In our approach, we represent the Minimal Retrieval Units (MRU) by word features within the vector space model, and by Maximal Frequent Sequences (MFS), accounting for the sequential aspect of text. For each of those two representations, a Retrieval Status Value (RSV) is computed. Those values are later combined to form a single RSV per MRU, that will be later propagated to parent nodes as described in section 6.

3.1 Preprocessing
The first step of the process is to cleanse the data. A first way to do this consists in skipping a set of words that are considered least informative, the *stopwords*. We also discarded all words of small size (less than three characters).

We also reduced each word to its stem using the Porter algorithm[8]. For example, the words "models", "modelling" and "modeled" are all stemmed to "model". This technique for reducing words to their root permits to further reduce the number of word terms.

This feature selection phase brings more computational comfort for the next steps since it greatly reduces the size of the document collection representation in the vector space model (the *dimension* of the vector space).

3.2 Modeling document fragments

The set of the remaining word stems W is used to represent the MRUs of the document collection within the *vector space model*. Each minimal retrieval fragment is represented by a $\|W\|$-dimensional vector filled in with a weight standing for the importance of each word w.r.t. that fragment. To calculate this weight, we used a normalized *tfidf* variation following the "tfc" term-weighting components as detailed by Salton et al. [11], that is:

$$tfidf_w = \frac{tf_w \cdot \log \frac{N}{n_w}}{\sqrt{\sum_{w_i \in W} \left(tf_{w_i} \cdot \log \frac{N}{n_{w_i}} \right)^2}}$$

Where tf_w is the term frequency of the word w. N is the total number of MRUs in the document collection and n the number of MRUs in which w occurs.

3.3 Extracting Maximal Frequent Sequences

3.3.1 *Definition and technique*

Maximal Frequent Sequences (MFS) are sequences of words that are frequent in the document collection and, moreover, that are not contained in any other longer frequent sequence. Given a frequency threshold σ, a sequence is considered to be frequent if it appears in at least σ documents.

Ahonen-Myka presents an algorithm combining bottom-up and greedy methods in [1], that permits to extract maximal sequences without considering all their frequent subsequences. This is a necessity, since maximal frequent sequences in documents may be rather long.

Nevertheless, when we tried to extract the maximal frequent sequences from the collection of minimal retrieval units extracted in 2, their number and the total number of word features in the collection did pose a clear computational problem and did not actually permit to obtain any result.

To bypass this complexity problem, we decomposed the collection of MRUs into several disjoint subcollections, small enough so that we could efficiently extract the set of maximal frequent sequences of each subcollection. Joining all the sets of MFS', we obtained an *approximate* of the maximal frequent sequence set for the full collection.

We conjecture that more consistent subcollections permit to obtain a better approximation. This is due to the fact that maximal frequent sequences are formed from similar text fragments. Followingly, we formed the subcollection by clustering similar documents together using the common k-means algorithm (see for example [15, 4]).

3.3.2 *Main Strengths of the Maximal Frequent Sequences*

The method efficiently extracts all the maximal frequent word sequences from the collection. From the definitions above, a sequence is said to be maximal if and only if no other frequent sequence contains that sequence.

Furthermore, a *gap* between words is allowed: in a sentence, the words do not have to appear continuously. A parameter g tells how many other words two words in a sequence can have between them. The parameter g usually gets values between 1 and 3.

For instance, if g = 2, a phrase "president Bush" will be found in both of the following text fragments:

```
...President of the United States Bush...
...President George W. Bush...
```

Note: Articles and prepositions were pruned away during the preprocessing.

This allowance of gaps between words of a sequence is probably the strongest specificity of the method, compared to the other existing methods for extracting text descriptors. This greatly increases the quality of the phrase, since processing takes the variety of natural language into account.

The other powerful specificity of the technique is the ability to extract maximal frequent sequences of any length. This permits to obtain a very compact description of documents. For example, by restricting the length of phrases to 8, the presence, in the document collection, of a frequent 25 words long phrase would result in thousands of phrases representing the same knowledge as this one maximal sequence.

3.4 Modeling queries

To build our queries, we only considered words found in the <title> and <keywords> elements. For consistency, we applied them the same preprocessing technique as to document fragments.

Vector space model. A novelty in INEX 2003 was the possibility to precede keywords with infix operators ("-" preceding a keyword meant that word was not desired, whereas "+" preceding a keyword indicated that word was especially important). We attached different weights to keywords preceded by such operators:

- no infix operator: the normal case, weight 1
- +: especially important, weight 1.5
- -: especially not desired, negative weight -1

In practice, things were not that simple, since the same word could occur within two phrases with contradictory operators (e.g., "language" in topic 111 occurs in -"programming language" and in +"human language"). In such rare cases, we ignored the word (weight: 0).

Keyphrases. All the phrases occurring in the <title> and <keywords> elements are stored in the (possibly empty) set of keyphrases representing the topic. For example, topic 117 (see figure 1) will be represented by the 4 phrases: "Patricia Tries", "text search", "string search algorithm", "string pattern matching".

4. EVALUATING DOCUMENTS

Once document fragments and queries are represented within our two models, a way to estimate the relevance of a fragment w.r.t. a query remains to be found. As mentioned earlier, we compute separate RSV values for the word features vector space model and the MFS model. In a second step, we aggregate these two RSVs into one single relevance score for each document fragment w.r.t. a query.

4.1 Word features RSV

The vector space model offers a very convenient framework for computing similarities between documents and queries. Indeed, there exists a number of techniques to compare two vectors. Eucclidean distance, Jaccard and cosine similarity being the most frequently used in IR. We have used cosine similarity because of its computational efficiency. By normalizing the vectors, which we did in the indexing phase, $cosine(\overrightarrow{d_1}, \overrightarrow{d_2})$ simplifies to vector product $(d_1 \cdot d_2)$.

4.2 MFS RSV

The first step is to create an MFS index for the document collection. Once a set of maximal frequent sequences has been extracted and each document is attached the corresponding phrases, as detailed in subsection 3.3, it remains to define the procedure to match a phrase describing a document and a keyphrase (from a query).

Note, that from here onwards, *keyphrase* denotes a phrase found in a query, and *maximal sequence* denotes a phrase extracted from a document fragment.

Our approach consists in decomposing keyphrases of the query into pairs. Each of these pairs is bound to a score representing its *quantity of relevance*. Informally speaking, the quantity of relevance of a word pair tells how much it makes a document relevant to include an occurrence of this pair. This value depends on the specificity of the pair (expressed in terms of inverted document frequency) and modifiers, among which an *adjacency coefficient*, reducing the quantity of relevance given to a pair formed by two words that are not adjacent.

4.2.1 Definitions:

Let D be a collection of N document fragments and $A_1 \dots A_m$ a keyphrase of size m. Let A_i and A_j be 2 words of $A_1 \dots A_m$ occurring in this order, and n be the number of MRUs of the collection in which A_iA_j was found. We define the quantity of relevance of the pair A_iA_j as:

$$Q_{rel}(A_iA_j) = idf(A_iA_j, D) \cdot adj(A_iA_j)$$

Where $idf(A_iA_j, D)$ represents the specificity of A_iA_j in the collection D:

$$idf(A_iA_j, D) = \log\left(\frac{N}{n}\right)$$

and $adj(A_iA_j)$ is a score modifier to penalize word pairs formed by non adjacent word sequences of $A_1 \dots A_m$:

$$adj(A_iA_j) = \begin{cases} 1, & \text{if } A_i \text{ and } A_j \text{ are adjacent} \\ 0 \leq \alpha_1 \leq 1, & \text{if d}(A_i,A_j) = 1 \\ 0 \leq \alpha_2 \leq \alpha_1 & \text{if d}(A_i,A_j) = 2 \\ \dots & \\ 0 \leq \alpha_{n-2} \leq \alpha_{n-3}, & \text{if d}(A_i,A_j) = n-2 \end{cases}$$

Followingly, the larger the distance between the two words, the lowest quantity of relevance is attributed to the corresponding pair. In our runs, we will actually ignore distances higher than 1 (i.e., $(k > 1) \Rightarrow (\alpha_k = 0)$).

Note the adjacency value of A_iA_j in $A_1 \dots A_m$ is also called the *modifying coefficient of* A_iA_j.

4.2.2 Example:

For example, ignoring distances above 1, a keyphrase ABCD is decomposed into 5 tuples (pair, modifying coefficient):

(AB, 1), (BC, 1), (CD, 1), (AC, α_1), (BD, α_1)

Let us compare this keyphrase to the documents d_1, d_2, d_3, d_4 and d_5, described respectively by the frequent sequences AB, AC, AFB, ABC and ACB. The corresponding quantities of relevance brought by the keyphrase ABCD are shown in table 1.

Assuming equal idf values, we observe that the quantities of relevance form a coherent order. The longest matches rank first, and matches of equal size are untied by adjacency. Moreover, non adjacent matches (AC and ABC) are not ignored as in many other phrases representations [7].

4.3 Aggregated RSV

In practice, some queries do not contain any keyphrase, and some documents do not contain any MFS. However, there can of course be correct answers to these queries, and those documents must be relevant answers to some queries. Also, all document fragments containing the same matching phrases get the same MFS RSV. Therefore, it is necessary to find a way to separate them. The word-based cosine similarity measure is very appropriate for that.

Another natural response would have been to re-decompose the pairs into single words and form fragment vectors accordingly. However, this would not be satisfying, because least frequent words are all missed by the algorithm for MFS extraction. An even more important category of missed words is that of frequent words that do not frequently cooccur with other words. The loss would be considerable.

This is the reason to compute another RSV using a basic word-features vector space model. To combine both RSVs to one single score, we must first make them comparable by mapping them to a common interval. To do so, we used *Max Norm*, as presented in [12], which permits to bring all positives scores within the range [0,1]:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="117" query_type="CO" ct_no="98">
<title>Patricia Tries </title>
<description>Find documents/elements that describe Patricia tries and
their use.</description>
<narrative>To be relevant, a document/element must deal with the use
of Patricia Tries for text search. Description of the standard
algorithm, optimisied implementation and use in Information retrieval
applications are all relevant. </narrative>
<keywords>Patricia tries, tries, text search, string search algorithm,
string pattern matching</keywords>
</inex_topic>
```

Figure 1: Topic 117

Document	MFS	Corresponding pairs	Matches	Quantity of relevance
d_1	AB	AB	AB	idf(AB)
d_2	ACD	AC CD AD	AC CD	idf(CD) + α_1.idf(AC)
d_3	AFB	AF FB AB	AB	idf(AB)
d_4	ABC	AB BC AC	AB BC AC	idf(AB) + idf(BC) + α_1.idf(AC)
d_5	ACB	AC CB AB	AC AB	idf(AB) + α_1.idf(AC)

Table 1: Quantity of relevance stemming from various indexing phrases w.r.t. a keyphrase query ABCD

$$New\ Score = \frac{Old\ Score}{Max\ Score}$$

Following this normalization, we aggregate both RSVs using a linear interpolation factor λ representing the relative weight of scores obtained with each technique (similarly as in [6]).

$$Aggregated\ Score = \lambda \cdot RSV_{Word\_Features} + (1-\lambda) \cdot RSV_{MFS}$$

The evidence of experiments with the INEX 2002 collection showed good results when weighting the single word RSV with the number of distinct words in the query, and the MFS RSV with the number of distinct words found in keyphrases of the query. In other words:

$$\lambda = \frac{\#(\text{distinct words})}{\#(\text{distinct words}) + \#(\text{distinct words in keyphrases})}$$

5. QUERY EXPANSION

Query expansion (QE)was used in two of the three runs that we submitted to INEX 2003. Both of these runs performed better than the one with no expansion at all. In runs one and two, the only difference was the use of QE, so it serves as a good benchmark for its significance for the performance EXTIRP. In the run without QE, the strict precision was 0.0061, and the generalized one 0.0105, when empty results were ignored. The corresponding figures when QE is added, are strict precision: 0.0323 and generalized precision: 0.0222.

In the rest of this chapter we will first describe some background concepts of QE in general. In section 5.2, we will describe our QE method, and in 5.3, we will describe further work in developing the method.

5.1 Background

It is generally agreed that modern variants of query expansion improve the results of a query engine [2]. However, there are many different ways in which QE can be performed. Some methods are based on relevance feedback, which can be blind, or which can involve feedback from the user. In both cases, the QE approach is local because it is based on the retrieved set of documents. A global QE approach uses the the information derived from the whole document collection. Modern global QE methods usually use an automatically constructed thesaurus [9, 3]. Other methods are based on manually crafted thesauri, such as WordNet, but experimental studies have shown that if the expansion terms from such theasuri are selected automatically, QE can even degrade the performance of the system [14].

5.2 The Process

Our QE process can be considered a form of blind relevance feedback that has been inspired by the standard Rocchio way [10] of calculating the modified query vectors. However, it is different from the traditional relevance feedback framework in that it takes into account only positive terms and no negative terms and in that it does not take into account all of the terms in the fragments, but only the best ones, which limits in practice the number of expansion terms per QE iteration between zero and ten. However, experimental studies have shown that even a few, carefully selected QE terms can considerably improve the performance of a system [13].

Here is the outline of the process:

1. Run EXTIRP. The output from EXTIRP is a set of ranked lists of document fragments. There is one list per topic and the fragments are ranked according to their RSV values with regard to the topic.
2. Take the ten topmost items of each list.
3. Calculate the similarity threshold value.
4. For each topic do:
   (a) Take those fragments whose RSV value is greater than the similarity threshold value. Make a list of words occurring in these fragments followed by their frequency count, and sort by frequency.
   (b) Take the ten topmost words and expand the topic with them.
   (c) Multiply the weights of the old terms by two and give the new terms the weight 1.
5. Run EXTIRP with the expanded topics.

We will now describe each of the steps in the process in more detail. In steps 1 and 5 EXTIRP is run with the same parameters and the RSV value is calculated according to these. This means that the only things that change from the first iteration to the second are the keywords in the topic and the threshold value for similarity.

In step 3, the similarity threshold for a given topic is determined in the following way: Read the topmost RSV value of the matches for each topic and maintain a list of the six smallest values. The threshold value is the highest one among the six smallest values. This way of determinig the similarity threshold value implies that there are always at least six topics that are not expanded. The topics vary a lot and it is thus necessary to treat them differently from each other. The number six was determined by training the QE method on the topics and assesments of the year 2002. The similarity threshold value has to be set always when the QE method is applied because running EXTIRP with different parameters results in radically different RSV values.

In step 4 (a), a list of words occurring in the fragments is created. This list is of course pruned from stopwords and the words are stemmed. As a stopword list a standard list for English language is used as well as a collection-specific list. The collection-specific list was created simply by gathering the most frequent terms in the collection. In stemming, the Porter stemmer [8] is used [2].

In step 4 (c), the weights of the old terms are multiplied by two and the new terms are given the weight 1. The possible weights of the old terms are: -1, 1 and 1.5. This means that the term weights in the expanded topic vectors can have the following values: -2, 1, 2 and 3. The topic vectors are normalized so that their length is one when they are processed by EXTIRP.

[2]The program was obtained from http://www.tartarus.org/ martin/PorterStemmer/

1. *Initialisation*:
   - $\forall n \in N$, score(n)=0
   - $\forall m \in M$, score(m)=RSV(m)
2. *Iterate*:
   - $\forall m \in M$: $\forall n_m \in N$ such that n_m is an ancestor of m, $score(n_m) = score(n_m) + score(m)$
3. *Final step*:
   - $\forall n \in N$, score(n)=$\frac{score(n)}{size(n)^{Upf}}$

Figure 2: Greedy upward propagation algorithm

5.3 Suggestions for improvement and furher work

The above QE method can be developed further in many ways. We plan to treat different topics in more individual ways, run the method through more iterations and perform QE on negative query terms as well. For example, EXTIRP can be run separately for each topic instead of running it for all topics at the same time, and the improvement of the RSV values can be calculated separately for each topic. In this way, the number of iterations performed per topic will vary. The number of iterations of the QE method does not have to be limited to one. Instead, we can perform QE until a stable level of performance is reached. Expansion of negative query terms can be performed in a similar way as expansion of positive query terms. In negative expansion, we will run EXTIRP with the negative terms and expand the topics with those terms that are most common in the matches of this negative query.

6. UPWARD PROPAGATION OF MINIMAL RETRIEVAL UNITS

The result of the previous steps is an RSV value for each MRU of the document collection, as defined in section 2. We propose in this section a technique for assigning an RSV to each ancestor of at least one MRU.

The principle is to propagate upwards the relevance value of each MRU node, weighting each relevance value upon the size of the corresponding element. We define the size of a node as the sum of the sizes of all its leaf descendants. In turn, the size of a leaf node is the number of characters it contains (its #PCDATA value).

Let A be an XML document, N the set of element nodes of A, M the set of MRUs of A. We compute the score of each node $n \in A$ as shown in figure 2.

Where UPF (Upward Propagation Factor) is a parameter that modulates the importance of the size of the elements. High UPF values give more penality to high sizes, and cause smaller candidates to be privileged. On the other hand, if UPF=0, for any given article, the best score will be given to the full article.

Because we assume that users go through answers in increasing rank order, we decided to avoid to propose them a document fragment they had already seen. Therefore, as a postprocessing, we decided to prune every node having an ancestor with a higher rank.

Following this, for example if UPF=0, the set of answers will only contain full articles.

7. OUR RUNS

- **UHel-Run1.**
  - MRU granularity: paragraph
  - Number of clusters: 200
  - MFS frequency threshold: $\sigma = 7$
  - Upward propagation: 2
  - No Query Expansion
- **UHel-Run2.**
  - MRU granularity: subsections
  - Number of clusters: 100
  - MFS frequency threshold: $\sigma = 7$
  - Upward propagation: 2
  - With Query Expansion
- **UHel-Run3.**
  - MRU granularity: subsections
  - Number of clusters: 100
  - MFS frequency threshold: $\sigma = 7$
  - Upward propagation: 5
  - With Query Expansion

8. CONCLUSIONS

We came up with a new technique for exploiting the logical structure of XML document so as to give more focused answers to information retrieval queries. We implemented this system and submitted runs to INEX 2003. At the moment, we have not properly analyzed our results, but we observed with satisfaction that our runs 2 and 3 are ranking 5th and 4th w.r.t. to strict quantisation in the inex_eval_ng evaluation technique [5].

However, there is a number of improvements to be achieved. First, we plan to reuse the clusterings formed prior to the extraction of maximal frequent sequences, aiming at query optimization. The idea is that by comparing queries to centroids of MRU clusters, we will be able to efficiently skip large quantities of MRUs, without having to compute similarity measures for each minimal unit individually.

Another concern is the fact that the current upward propagation formula is exponential in nature, meaning a small variation in the UPF factor can cause a switch from a set of answers with a large majority of minimal units to a set of answers with a large majority of full articles. Part of our future work is to explore the various ways to smooth this effect.

9. REFERENCES

[1] H. Ahonen-Myka. Finding All Frequent Maximal Sequences in Text. In *Proceedings of the 16th International Conference on Machine Learning ICML-99 Workshop on Machine Learning in Text Data Analysis, Ljubljana, Slovenia*, pages 11–17. J. Stefan Institute, eds. D. Mladenic and M. Grobelnik, 1999.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, ACM Press, New York, 1999.

[3] C. J. Crouch and B. Yang. Experiments in automatic statistical thesaurus construction. In *Proceedings of the 15th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 77–88, 1992.

[4] A. Doucet and H. Ahonen-Myka. Naive clustering of a large xml document collection. In *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, pages 81–87, 2002.

[5] N. Gövert, G. Kazai, N. Fuhr, and M. Lalmas. Evaluating the effectiveness of content-oriented XML retrieval. Technical report, University of Dortmund, Computer Science 6, 2003.

[6] M. Marx, J. Kamps, and M. de Rijke. The university of amsterdam at inex, 2002.

[7] M. Mitra, C. Buckley, A. Singhal, and C. Cardie. An analysis of statistical and syntactic phrases. In *Proceedings of RIAO97, Computer-Assisted Information Searching on the Internet*, pages 200–214, 1987.

[8] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[9] Y. Qiu and H. Frei. Concept based query expansion. In *Proceedings of the 16th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–169, 1993.

[10] J. J. Rocchio. *Relevance feedback in information retrieval. In Salton, G., editor, The SMART Retrieval System - Experiments in Automatic Document Processing.* Prentice Hall Inc., 1971.

[11] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management: an International Journal*, 24(5):513–523, 1988.

[12] C. C. Vogt and G. W. Cottrell. Predicting the performance of linearly combined IR systems. pages 190–196.

[13] E. Voorhees. Relevance feedback revisited. In *Proceedings of the 15th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1–10, 1992.

[14] E. Voorhees. Query expansion using lexical-semantic relations. In *Proceedings of the 17th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 61–69, 1994.

[15] P. Willett. Recent trends in hierarchic document clustering: a critical review. *In Information Processing and Management*, 24(5):577–597, 1988.

Using value-added document representations in INEX

Birger Larsen, Haakon Lund, Jacob K. Andresen and Peter Ingwersen
Department of Information Studies
Royal School of Library and Information Science
Birketinget 6, DK-2300 Copenhagen S, Denmark
{blar,hl,jaa,pi}@db.dk

ABSTRACT
We describe our efforts to generate different functional and cognitive representations of the INEX corpus, and the results of simple runs with these representations.

1 INTRODUCTION
Highly structured XML documents offer unique opportunities for extracting many different representations of documents for information retrieval (IR) purposes. In this paper we describe our efforts to work with combinations of different representations generated from the corpus of the INEX collection as well as from external sources. The purpose of the experiments was to initiate tests of the theory of polyrepresentation [3] with different cognitive and functional representations of the document corpus.

The paper is structured as follows: The theory of polyrepresentation is briefly discussed as a theoretical framework for the experiments in section 2. Section 3 describes the experimental setup, and section 4 analyses the results. Section 5 gives tentative conclusions.

2 POLYREPRESENTATION
The theory of polyrepresentation [3] provides a theoretical background for working with different representations from several sources. In summary, the theory hypotheses that overlaps between different cognitive and functional representations of both users' information needs as well as documents can be exploited for reducing the uncertainties inherent in Information Retrieval (IR), and thereby improve the performance of IR systems. Two or more different cognitive representations pointing at the same documents is regarded as multi-evidence of those documents being relevant, and suggests to apply a principle of 'intentional redundancy' [2] with the purpose of reducing the uncertainties by placing emphasis on overlaps between representations. Better results are expected when cognitively unlike representations are used, e.g., the document title (made by the author) vs. intellectually assigned descriptors from indexers.

Although the theory of polyrepresentation is holistic in nature and amalgamates user-oriented approaches with both Boolean and best match principles it is, however, inherently *Boolean* in much of its reasoning. This is apparent in the pronounced focus on cognitive retrieval overlaps, i.e., *sets* of documents retrieved based on different cognitive representations, see, e.g., the appendix example in [3]. A little discussed, but inherent point is that the structure ensures the *quality* of the sets that are matched. But this structure does not necessarily have to be of a Boolean nature – other kinds of structure may be implemented. Such may include the probabilistic query operators in the InQuery IR system for instance as utilised by [4] to achieve various degrees of structure in queries.

Inspired by the work of Madsen and Pedersen [12] Larsen [9] proposes the idea of a polyrepresentation continuum (See Figure 1 below) as a model for discussing how structured a given implementation of polyrepresentation is.

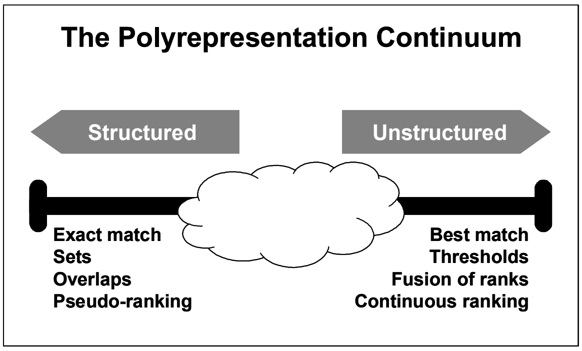


Figure 1. The polyrepresentation continuum [From 9, p. 36]

At the *structured* pole of the continuum the implementations are based on exact match principles, leading to sets of retrieved documents for each representation from which overlaps can be formed and a pseudo-ranking be constructed. At the *unstructured* pole of the continuum the implementations are based on best match principles leading to a rank of the documents that are retrieved as input for polyrepresentation. Rather than straight generation of overlaps between sets, the implementations at the unstructured pole of the polyrepresentation continuum will consist of fusing ranks to produce a final ranked output, perhaps aided by thresholds to provide the necessary quality by restricting the ranks to be fused to the top ranked documents only.

Few empirical investigations have been carried out of the theory of polyrepresentation so far. Larsen [8] reports a small online Boolean experiment at the structured end of the continuum. The MSc thesis of Madsen and Pedersen [12] combines a highly structured Boolean approach with probabilistic query operators in a best match system, and is as such placed closer to the middle of the continuum.

3 METHODS

The main focus of the runs submitted to INEX2003 was on obtaining functionally and cognitively different representations of the documents. Only simple fusion strategies for combining the representations were used because of lacking time to experiment with more advanced ones (See section 3.2). The runs submitted to INEX2003 were therefore close to the unstructured pole of the polyrepresentation continuum. The investigation of more advanced strategies for how to combine these in a suitable structured manner according to the theory of polyrepresentation is the subject of future work. Note that the purpose of the experiments reported in the present paper was to retrieve *whole* documents, and not document components as in most approaches in INEX.

Functionally different representations are defined as representations originating from the same cognitive agent, e.g., the article title or figure captions made by the author [3]. In relation to IR, representations are regarded as cognitively different if they originate from other cognitive agents than the author, e.g., descriptors from a thesaurus assigned intellectually to the documents, or later citations or links to the document by other authors. The corpus of the INEX test collections offers excellent opportunities for the generation of functionally different representations originating from the author because of the elaborate XML structure of the documents. In addition, a range of cognitively different representations of the documents are available because the journals in the corpus are indexed in the INSPEC database. A further opportunity offered by the INEX corpus is to exploit the references in the bibliographies to generate citation-based representations.

The InQuery IR system was used for all runs because it offers the possibility to store different representations of the documents in fields and to combine these using both Boolean and softer query operators.

3.1 Indexes and fields[1]

Two indexes were constructed, each containing three fields: one with author generated representations, one with intellectually assigned descriptors from a domain thesaurus, and one with a citation index generated from the corpus (See Figure 2 and Figure 3).

The first field consists of different types of *titles* from the documents: the article title, the section headings at all levels, and the cited titles from the bibliographies. These are either generated or selected by the author. The inclusion of section headings is inspired by the Subject Access Project (SAP) [1; 18] where section headings, figure and table captions were extracted as representations in addition to the article titles. The use of cited titles has been proposed by Kwok [6; 7], and tested by Salton and Zhang [16]. The latter experiment did not show any general gains from including cited titles. However, only those articles that were also source documents in the test collections used were included in the experiment, i.e., only a limited selection of cited titles was used in the experiments. The INEX corpus has *all* cited titles and may thus provide better results with the cited titles. The path used for extracting the cited titles was //bb/atl. This includes the titles of cited journal articles and conference papers, but not the titles of cited books or reports. More than 7,000 documents contained such cited titles with an average of 9.9 cited titles per document.

Titles (FLD001)
(Article title, section titles, and cited titles)
	//fm/tig/atl
	//st
	//bb/atl

Descriptors (FLD002)	Intellectually assigned descriptors
Citation index (FLD003) (Boomerang effect)	Best possible tuning with INEX2002 test collection

Figure 2. Index A (without expansion on descriptors)

The second field consists of intellectually assigned descriptors from the INSPEC thesaurus. These were available for 7,711 of the 12,107 documents in the

[1] After submission we discovered a number of errors in the indexing process. Attempts have been made to correct these, and the methods and results reported here are for the corrected runs.

INEX corpus. Because only relatively few descriptors are assigned to each document by the INSPEC indexers this representation contained relatively few index keys. In an effort to enlarge this representation we expanded the descriptors by adding all the synonyms (the used for (UF) relation) as well as the narrower terms (NT) from the INSPEC thesaurus. Index A contained the un-expanded descriptors (Figure 2), and Index B contained the expanded descriptors (Figure 3).

Titles (FLD001)
(Article title, section titles, and cited titles)
	/fm/tig/atl
	//st
	//bb/atl

Descriptors (FLD002) (expanded document representation)	Intellectually assigned descriptors, expanded from the INSPEC thesaurus (NT, UF)
Citation index (FLD003) (Boomerang effect)	Best possible tuning with INEX2002 test collection

Figure 3. Index B (with descriptors expanded from the thesaurus)

The third field in both indexes contained data for constructing a citation index, i.e., data to identify the references in each document. When indexed in the database documents can be retrieved that refer to (cite) a particular *seed document*. Such search strategies have shown promising results [13-15], but have rarely been exploited in IR research[2]. This is probably partly due to a lack of citation data in the test collections developed in the last decade, and partly due to the lack of seed documents to represent the information need. A particular approach to identify such seed documents automatically was used to construct queries for the citation index (See section 3.2). The index was constructed based on the cited titles discussed above in combination with the cited year. Because there were numerous typos etc. in the cited titles an implementation of the edit distance algorithm was used to identify variants to the same cited document[3]. 7,111 documents contained references with both cited titles and cited years. In these documents there were 70,634 unique citations after merging of variants, and these were mentioned a total of 192,881 times in the documents. The citations were represented by id-numbers to ease processing.

3.2 Queries

Only CO topics were used because the only whole documents were retrieved with the tested approach.

The same queries were used for both the title field and the field containing descriptors (FLD001 and FLD002). These were constructed manually from the title elements of the CO topics translating the INEX operators into InQuery's probabilistic query operators (See Figure 4).

In order to be able to match the content of the citation index with the topics, the latter had to be translated into citations. This was done with a best match version of the so-called boomerang effect proposed in [8; 10]. In short, the boomerang effect extracted the citations from sets of documents retrieved by natural language queries from a range of functional and cognitive representations. These citations were used as seed documents in a citation search that can retrieve later documents that cite the seed documents. The occurrence of the citations between representations and their frequency was used to weight and select which citations to use as seed documents as well as to weight the seed documents in the query (See [10] for details). The boomerang effect used was the best possible tuning based on the INEX2002 test collection: citations were extracted from 8 documents resulting in 252 seed documents in average per query.

InQuery's #sum operator was used to combine the fields (See Figure 4). Only a simple strategy was used to fuse the fields because the main focus was on obtaining functionally and cognitively different representations of the documents. Therefore the runs can be characterised as being at the unstructured end of the polyrepresentation continuum. The same queries were used for index A and index B.

```
#sum (

#field (FLD001 #and(#1(natural language processing)
(#1(human language))) #not(#1(programming
language)) #not(#1(modeling language)))

#field (FLD002 #and(#1(natural language processing)
(#1(human language))) #not(#1(programming
language)) #not(#1(modeling language)))

#field (FLD003 #WSUM(1 3797.98 CIT_ID46361
2404.53 CIT_ID28456 1898.99 CIT_ID43757 1898.99
CIT_ID43816 1898.99 CIT_ID57141 ... )) )
```

Figure 4. Sample query (CO topic 111). Note that the citation query in FLD003 has been shortened.

[2] Increasingly, web search engines exploit link data. However, there are indications that although similar in conception links and citations may be quite different in practice, see e.g., [17]. CiteSeer is an exception because it uses citations extracted from scientific papers [11].

[3] We greatly acknowledge the Department of Information Studies, University of Tampere, Finland for making the source code for this implementation available to us.

3.3 Runs

The two main runs were the runs on index A and index B to study the effect of the expanded descriptors. We also did runs on the individual fields to assess their contribution to the overall result. Six runs are reported here: IndexA_run, IndexB_run, Titles_run, Descriptor_run, Descriptor_expanded_run, and Citation_index_run.

4 RESULTS

Table 1 shows the results for the strict quantification function in inex_eval. Overall the results display a low performance compared to the best runs in INEX2003: The highest AvgP value was 0.0419 for the Titles_run. The top 10 in INEX2003 was in the 0.1140-0.0677 range.

Run	AvgP (strict)
IndexA_run	0.0385
IndexB_run	0.0300
Titles_run	0.0419
Citation_index_run	0.0327
Descriptor_run	0.0099
Descriptor_expanded_run	0.0009

Table 1. Overall results. Strict quantification function.

Figure 5 and Figure 6 show P-R curves for the runs. It is immediately obvious from Figure 6 and the AvgP value for the run (Table 1) that there has been a processing error in the expansion of the descriptors. This means that the performance of the IndexB_run has suffered: Compared to IndexA the performance drops from 0.0385 to 0.0300 for IndexB.

Figure 6 shows the performance each individual field. The un-expanded descriptors in themselves perform quite poorly (AvgP = 0.0099), and the idea of expanding this representation is supported. The Titles_run have the best performance of all 6 runs (AvgP = 0.0419), followed by the Citation_index_run (AvgP = 0.0327).

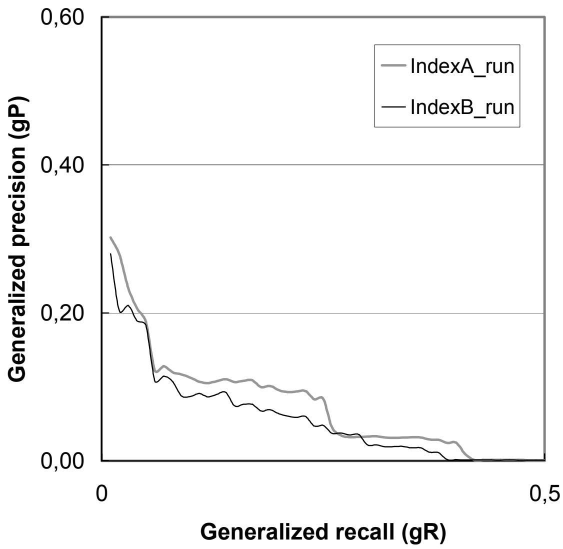


Figure 5. P-R curves for IndexA and IndexB run using the strict quantification function in inex_eval.

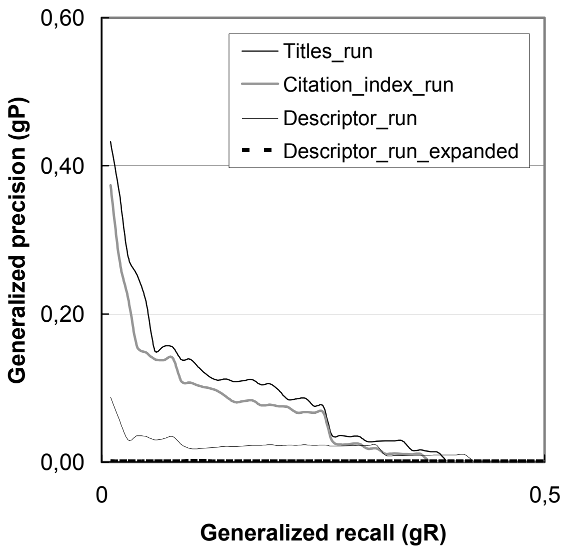


Figure 6. P-R curves for the individual fields using the strict quantification function in inex_eval.

5 CONCLUSIONS

The overall aim of our runs submitted to INEX2003 was to work on obtaining functionally and cognitively different representations of the documents. Two of these were successful: The titles representation consisting of the article title, headings and cited titles, and the citation index, which performed fairly well. The intellectually assigned descriptors did not perform well, and it was attempted to expand these in the document representation by using the INSPEC thesaurus. Due to technical errors this failed succeed, and we do not know the effect of the expansion.

Future work includes correction of the expansion error and a subsequent investigation of its behaviour and performance. Other expansion techniques on the query side can also be implemented, e.g., similar to the ones tested in [5].

The approach tested in the runs was close to the un-structured pole of the polyrepresentation continuum. Future work also includes investigations of more advanced structured query strategies to improve the quality of the initial set used, and move the tests closer to the structured pole of the continuum.

6 ACKNOWLEDGMENTS

The InQuery IR system was provided by the Center for Intelligent Information Retrieval, University of Massachusetts, Amherst, MA, USA. The

7 REFERENCES

1. Atherton-Cochrane, P. (1978): *Books are for use : final report of the subject access project to the Council on Library Resources*. Syracuse, N. Y.: School of Information Studies, Syracuse University. 172 p.
2. Ingwersen, P. (1994): Polyrepresentation of information needs and semantic entities : elements of a cognitive theory for information retrieval interaction. In: Croft, W. B. and van Rijsbergen, C. J. eds. *SIGIR '94 : Proceedings of the seventeenth annual international ACM-SIGIR conference on research and development in information retrieval, organised by Dublin City University, 3-6 July 1994, Dublin, Ireland.* London: Springer-Verlag, p. 101-110.
3. Ingwersen, P. (1996): Cognitive perspectives of information retrieval interaction : elements of a cognitive IR theory. *Journal of Documentation*, 52(1), 3-50.
4. Kekäläinen, J. and Järvelin, K. (1998): The impact of query structure and query expansion on retrieval performance. In: Croft, W. B., Moffat, A., van Rijsbergen, C. J. and Zobel, J. eds. *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (ACM SIGIR '98), Melbourne, Australia, August 24-28, 1998*. New York: ACM Press, p. 130-137.
5. Kristensen, J. (1993): Expanding end-user's query statements for free text searching with a search-aid thesaurus. *Information Processing & Management*, 29(6), 733-744.
6. Kwok, K. L. (1975): The use of titles and cited titles as document representations for automatic classification. *Information Processing & Management*, 11(8-12), 201-206.
7. Kwok, K. L. (1984): A document-document similarity measure based on cited titles and probability theory, and its application to relevance feedback retrieval. In: van Rijsbergen, C. J. ed. *Research and development in information retrieval : proceedings of the third joint BCS and ACM symposium, King's College, Cambridge, 2-6 July 1984*. Cambridge: Cambridge University Press, p. 221-231.
8. Larsen, B. (2002): Exploiting citation overlaps for information retrieval: generating a boomerang effect from the network of scientific papers. *Scientometrics*, 54(2), 155-178.
9. Larsen, B. (2004): *References and citations in automatic indexing and retrieval systems : experiments with the boomerang effect*. Copenhagen: Royal School of Library and Information Science. XIII, 291 p. (PhD thesis - under review)
10. Larsen, B. and Ingwersen, P. (2002): The boomerang effect : retrieving scientific documents via the network of references and citations. In: Beaulieu, M., Baeza-Yates, R., Myaeng, S. H. and Järvelin, K. eds. *Proceedings of SIGIR 2002 : the twenty-fifth annual international ACM SIGIR conference on research and development in information retrieval, August 11-15, 2002, Tampere, Finland*. New York: ACM Press, p. 397-398. (Poster paper) [http://www.db.dk/blar (Preprint), visited 3-8-2003]
11. Lawrence, S., Giles, C. L. and Bollacker, K. D. (1999): Autonomous Citation Matching. In: Etzioni, O., Müller, J. P. and Bradshaw, J. M. eds. *AGENTS '99. Proceedings of the Third*

Annual Conference on Autonomous Agents, May 1-5, 1999, Seattle, WA, USA. New York: ACM Press, p. 392-393. (Poster paper) [http://citeseer.nj.nec.com/lawrence99autonomous.html (preprint), visited 16-8-2003]

12. Madsen, M. and Pedersen, H. (2003): *Polyrepræsentation som IR metode : afprøvning af polyrepræsentationsteorien i et best match IR system [Polyrepresentation as IR method : test of the theory of polyrepresentation in a best match IR system]*. [Copenhagen]: Danmarks Biblioteksskole. 106 p.+ XLIII p. (In Danish - unpublished MLIS thesis)

13. McCain, K. W. (1989): Descriptor and citation retrieval in the medical behavioral sciences literature: Retrieval overlaps and novelty distribution. *Journal of the American Society for Information Science*, 40(2), 110-114.

14. Pao, M. L. (1993): Term and citation retrieval - a field-study. *Information Processing & Management*, 29(1), 95-112.

15. Salton, G. (1971): Automatic indexing using bibliographic citations. *Journal of Documentation*, 27(2), 98-110.

16. Salton, G. and Zhang, Y. (1986): Enhancement of text representations using related document titles. *Information Processing & Management*, 22(5), 385-394.

17. Thellwall, M. (2003): What is this link doing here : begining a fine-grained process of identifying reasons for academic hyperlink creation. *Information Research*, 8(2), paper no. 151. [http://informationr.net/ir/8-3/paper151.html, visited 8-11-2003]

18. Wormell, I. (1985): *Subject Access Project : SAP : improved subject retrieval for monographic publications*. Lund: Lund University. 141 p.

Adapting the Extended Vector Model for XML Retrieval

Carolyn J. Crouch
Department of Computer Science
University of Minnesota Duluth
Duluth, MN 55812
(218) 726-7607
ccrouch@d.umn.edu

Sameer Apte
Department of Computer Science
University of Minnesota Duluth
Duluth, MN 55812
(218) 726-7607
apte0002@d.umn.edu

Harsh Bapat
Persistent Systems Pvt. Ltd.
Pune, India 411016
+91 (20) 567 8900
harsh_bapat@persistent.co.in

ABSTRACT
In this paper, we describe our approach to XML retrieval, which is based on the extended vector space model initially proposed by Fox [5]. The current implementation of our system and results to date are reported. The basic functions are performed using the Smart experimental retrieval system. Early results confirm the viability of the extended vector space model in this environment.

1. INTRODUCTION
When we began our work with INEX last year, our goal was to confirm the utility of Salton's vector space model [10] in its extended form for XML retrieval. Long familiarity with Smart [9] and it capabilities led us to believe that it could be used for this purpose. Our approach was described in the Proceedings of last year's workshop [3]. Much initial effort was spent on the translation of documents and topics from XML to internal Smart format and the subsequent translation of results back into INEX format. When we reported our results in [3], our system was still in a very rudimentary stage and far from error-free.

During the past year, we built upon and extended this work. We now have an operational system. For the sake of clarity, a brief overview follows.

1.1 Background
Everyone involved in information retrieval is familiar with the vector space model, wherein documents and queries are represented as weighted term vectors. The weight assigned to a term is indicative of the contribution of that term to the meaning of the document. Very commonly, *tf-idf* weights [11] or some variation thereof [12] are used. The similarity between vectors (e.g., document and query) is represented by the mathematical similarity of their corresponding term vectors.

In 1983, Fox [5] proposed an extension of the vector space model—the so-called extended vector space model—to allow for the incorporation of objective identifiers with content identifiers in the representation of a document. An extended vector can include different classes of information about a document, such as author name, bibliographic citations, etc., along with content terms. In this model, a document vector consists of a set of subvectors, where each subvector represents a different class of information (i.e., concept class or c-type). Our current representation of an XML document/query consists of 18 c-types (i.e., *article, ti, atl, pub_yr, sec, st, fgc, article_au_fnm, article_au_snm, abs, kwd, ack, tig, bibl_au_fnm, bibl_au_snm, bibl_ti, bibl_atl, p*) as defined in INEX guidelines. Similarity between extended vectors is calculated as a linear combination of the similarities of corresponding subvectors.

Use of the extended vector model for document retrieval normally raises at least two problems: the construction of the extended search request [4, 6] and the selection of the coefficients for combining subvector similarities. For XML retrieval, of course, the query is posed in a form that is easily translated into an extended vector. The second problem—the weighting of the subvectors themselves—remains open to investigation. Another issue of some interest here is the weighting of terms within the subvectors—objective vs. subjective. (We have produced some useful results in relation to the term weighting issue; our work on the weighting of subvectors is promising but not well developed. In any case, subvector weighting is unlikely to have a measurable effect within the large INEX window.)

The extended vector capability of Smart appeared to us well suited for XML with respect to the retrieval of documents. But there is no facility for retrieving at the element level (or at various levels of granularity), which is a requirement of INEX tasks. We are interested in determining the feasibility of incorporating the functionality (i.e., flexibility and granularity) required for XML retrieval within the extended vector environment. (Our first experiment in this vein, based on the methods of Grabs and Schek [7, 8], was not successful. However, more work is necessary before conclusions can be drawn.)

1.2 System Description
Our system handles the processing of XML text as follows:

(1) The documents are parsed using a simple XML parser available on the web. Each of our 18 c-types is now identifiable in terms of its XML path.

(2) The documents and queries are translated into Smart format and indexed by Smart as extended vectors. (For the results reported in this paper, we used only the *article*-based indexing.)

(3) Retrieval takes place by running the queries against the indexed collection. The result is a list of *articles* ordered by decreasing similarity to the query. (A variety of term weighting schemes are available through Smart.)

(4) For each query, results are sorted by correlation and the top 100 elements are converted to INEX format and reported.

The retrieval itself is straight-forward. The only variation is the splitting of certain CAS queries into separate portions which are then run in parallel to ensure that the elements retrieved meet the specified criteria. See [3] for an example of this type.

2. EXPERIMENTS

In the following sections, we describe the experiments performed with respect to the processing of the CO and CAS topics, respectively. In all cases, we use only the topic title and keywords as search words in query construction.

2.1 Using CO Topics

Our first task is to formulate the CO topic in extended vector form. Of the 18 c-types composing the extended vector, 8 contain subjective identifiers (i.e., *abs*, *kwd*, *bdy*, *atl*, *edintro*, *ack*, *bibl_atl*, and *bibl_ti*). The extended vector topic is formed by associating the search words of the topic with each of these 8 c-types. The remaining c-types contain objective identifiers and are not used in formulating CO queries. Our more interesting experiments are discussed briefly below; see [1] for more information. The subvectors were equally weighted in all these cases.

2.1.1 2002 Topics

All the results based on 2002 topics were produced through the original inex_eval.

Tuned *Lnu-ltu* Term Weighting: In this experiment, we tuned the collection as indicated by Singhal, *et. al*., in [13]. Results under generalized quantization were 0.065 whereas strict quantization produced 0.095.

Augmented *tf-idf* (*atc*) Term Weighting: 2002 topics under generalized quantization produced an average precision of 0.033.

Retrieval at the Element Level: In this experiment, we used indexings of the collection at the paragraph and section levels in addition to the article level. Untuned (estimated) *Lnu-ltu* weights were used in each case. For each query, the rank-ordered lists were sorted and the top 100 elements reported. Average precision was 0.042 under generalized quantization.

Flexible Retrieval: In this experiment, we used the method of Grabs and Schek [7, 8]. Identifying the paragraph as the basic indexing node, we used the paragraph indexing of the collection and reported the best elements by calculating statistics for other elements on the fly. Average precision was 0.018 under generalized quantization.

2.1.2 2003 Topics

Our 2003 CO submission was based on parameters that produced the best results for 2002 CO topics, i.e., *Lnu-ltu* term weighting with equal subvector weights. The recall-precision graphs for 2003 CO topics under the revised inex_eval are given below in Figures 1 and 2. The results under inex_eval_ng (overlap ignored) are shown in Figures 3 and 4.

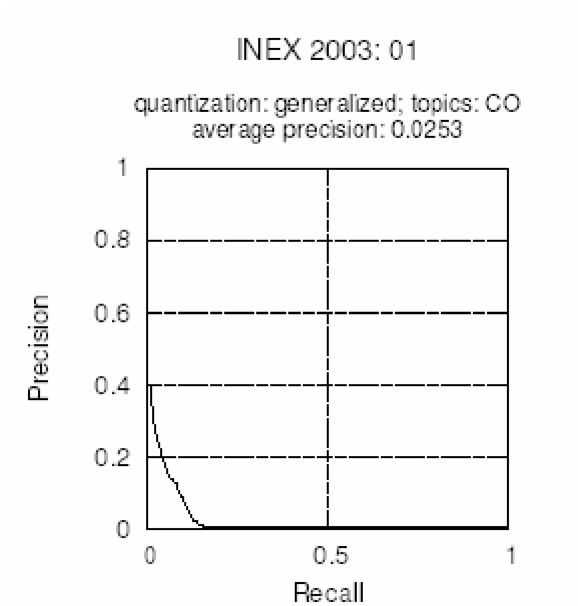


Figure 1. Recall-precision for CO, Generalized

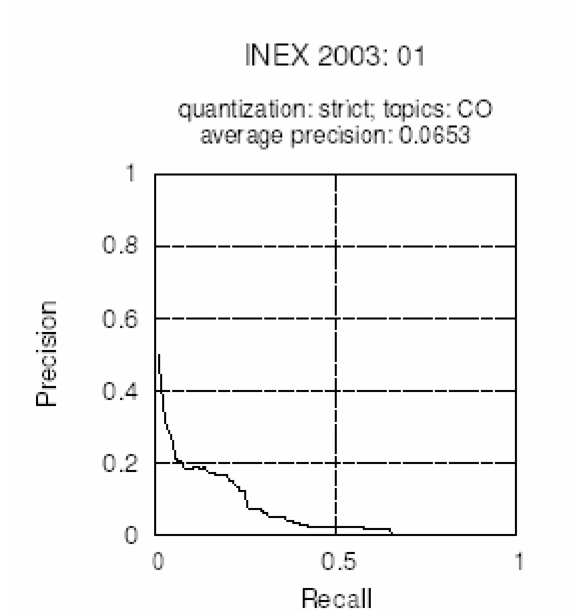


Figure 2. Recall-precision for CO, Strict

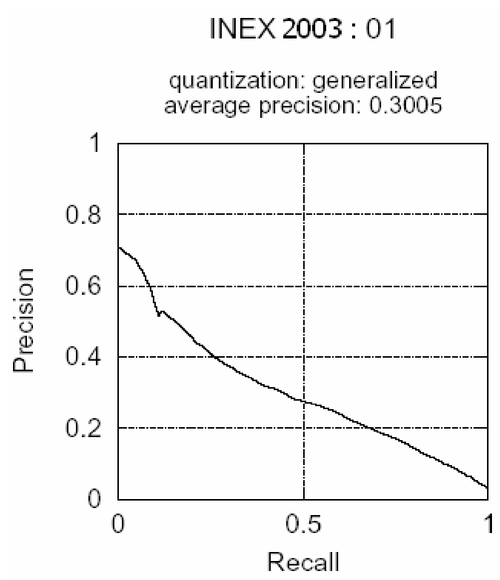


Figure 3: Recall-precision for CO, Gen under ng

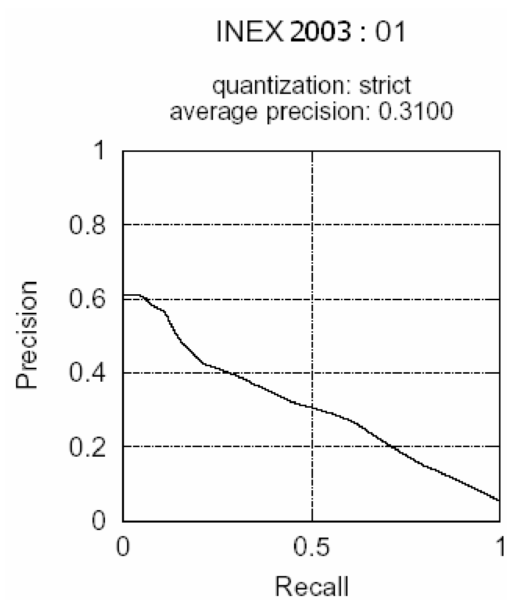


Figure 4: Recall-precision for CO, Strict under ng

2.2 Using CAS Topics

Although we were able to formalize the formulation of extended vector CO topics very easily, CAS topics are more of a challenge. At present, we are still working on this task. Thus the 2003 topic formulations were performed manually. Equal subvector weighting was applied in each case. Experiments performed during the past year using the INEX 2002 queries are described briefly below; see [2] for more information. Evaluation was performed through the original inex_eval.

2.2.1 2002 Topics

Untuned *Lnu_ltu* Term Weighting: All subvectors are weighted in this fashion. Not unexpectedly, average precision was low—0.179 under generalized and 0.222 under strict quantization.

Lnu_ltu (for subjective subvectors) and *nnn* (for objective subvectors) Term Weighting: Here we used simple term frequency weights (*nnn*) for the objective subvectors combined with *Lnu_ltu* weights for the subjective subvectors. Average precision was 0.187 under generalized and 0.235 under strict quantization.

Augmented tf-idf (*atc*) Term Weighting: All subvectors were weighted with *atc* weights. Average precision was 0.194 and 0.238 under generalized and strict quantization, respectively.

Augmented tf-idf (*atc*—for subjective subvectors) and *nnn* (for objective subvectors) Term Weighting: These weights returned an average precision of 0.192 under generalized and 0.243 under strict quantization.

2.2.2 2003 Topics

Our 2003 submission used *atc* term weighting for all subvectors with equal subvector weights. Due to the exigencies of the academic schedule, we were able to submit only under VCAS. We will report these results when they become available through INEX.

2.3 Results

During the past year, we produced a working system. Our results for CO topics are on the whole very good, ranking first or second in 3 of the 4 evaluations (e.g., in Figures 3 and 4) under inex-eval-ng. Yet although we were able to produce decent results for the 2002 CO topics under the original inex_eval, our results for the 2003 CO topics under the revised inex_eval fall far from the top. We are still assessing the causes. The assessment of our CAS submission awaits evaluation from INEX. An overview of results to date may be seen in Table 1.

3. CONCLUSIONS

Our system is still in an early stage of development. The issue of term weighting has now become clearer; the weighting of the subvectors themselves is still an open question. The next challenge is to develop a method of returning results at the element level, i.e., to retrieve at the desired level of granularity. Our plans include further investigation of the methods of others along with the development of an approach that may be better suited to our own environment.

Table 1. Comparison of Best Case Avg Precision for CO Topics

	UMD		INEX	
	gen	strict	gen	strict
'02 Topics	0.0650	0.0950	0.0700	0.0880
'03 Topics: inex_eval	0.0253	0.0653	0.0968	0.1140
'03 Topics: inex_eval_ng*	0.2971	0.2961	0.3051	0.2961
'03 Topics: inex_eval_ng**	0.2789	0.2873	0.3408	0.2966

* overlap ignored; ** overlap considered

4. REFERENCES

[1] Apte, S. Adapting the extended vector space model for content-oriented XML retrieval. Master's Thesis, Dept. of Computer Science, University of Minnesota Duluth (2003).

[2] Bapat, H. Adapting the extended vector space model for structured XML retrieval. Master's Thesis, Dept. of Computer Science, University of Minnesota Duluth (2003).

[3] Crouch, C., Apte, S., and Bapat, H. Using the extended vector model for XML retrieval. In Proc of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), (Schloss Dagstuhl, 2002), 99-104.

[4] Crouch, C., Crouch, D. and Nareddy, K. The automatic generation of extended queries. In Proc. of the 13th Annual International ACM SIGIR Conference, (Brussels, 1990), 369-383.

[5] Fox, E. A. Extending the Boolean and vector space models of information retrieval with p-norm queries and multiple concept types. Ph.D. Dissertation, Department of Computer Science, Cornell University (1983).

[6] Fox, E., Nunn, G. and Lee, W. Coefficients for combining concept classes in a collection. In Proc. of the 11th Annual International ACM SIGIR Conference, (Grenoble, 1988), 291-307.

[7] Grabs, T. and Schek, H. Generating vector spaces on-the-fly for flexible XML retrieval. In Proc of the ACM SIGIR Workshop on XML and Information Retrieval, (Tampere, Finland, 2002), 4-13.

[8] Grabs, T. and Schek, H. ETH Zurich at INEX: Flexible information retrieval from XML with PowerDB-XML. INEX 2002 Workshop Proceedings, (Dortland, 2002), 35-40.

[9] Salton, G. Automatic information organization and retrieval. Addison-Wesley, Reading PA (1968).

[10] Salton, G., Wong, A., and Yang, C. S. A vector space model for automatic indexing. Comm. ACM 18, 11 (1975), 613-620.

[11] Salton, G. and Buckley, C. Term weighting approaches in automatic text retrieval. In IP&M 24, 5 (1988), 513-523.

[12] Singhal, A. AT&T at TREC-6. In The Sixth Text REtrieval Conf (TREC-6), NIST SP 500-240 (1998), 215-225.

[13] Singhal, A., Buckley, C., and Mitra, M. Pivoted document length normalization. In Proc. Of the 19th Annual International ACM SIGIR Conference, (Zurich,1996), 21-19.

Cooperative XML (CoXML) Query Answering at INEX 03

Shaorong Liu and Wesley W. Chu
UCLA Computer Science Department, Los Angeles, CA 90095
{sliu, wwc}@cs.ucla.edu

ABSTRACT

The Extensible Markup Language (XML) is becoming the most popular format for information representation and data exchange. Much research has been investigated on providing flexible query facilities while aiming at efficient techniques to extract data from XML documents. However, most of them are focused on only the exact matching of query conditions. In this paper, we describe a cooperative XML query answering system, CoXML, which cooperates with the users by extending query relaxation techniques and provides approximate matching of query conditions. We also present our participation effort in the Initiative for the Evaluation of XML Retrieval (INEX) with CoXML.

1. INTRODUCTION

With the growing popularity of the Extensible Markup Language (XML) [12], more and more information is stored and exchanged in the XML format [1]. XML is essentially a textual representation of hierarchical (tree-like) data where a meaningful piece of data is bounded by matching starting and ending tags, such as <name> and </name>.

To cope with the tree-like structures in the XML model, several XML-specific query languages have recently been proposed (e.g. Xpath [15], Quilt [3], XML-QL [13] and XQuery [16] etc.). All these XML query languages aim at only the exact matching of query conditions. Answers are found when those XML documents match the given query conditions *exactly*. However, this may not always be the case in the XML model. To remedy this condition, we are developing a query relaxation framework for searching answers that match the given query conditions *approximately*. *Query relaxation* enables systems to relax the user query to a less restricted form to derive approximate answers. Such a technique has been successfully used in the relational databases (e.g. CoBase [5]) and has proven to be a valuable technique for deriving approximate answers.

In the XML domain, the need for query relaxation increases since the flexible nature of the XML model allows varied structure or values, and the non-rigid XML tag syntax enables users to embed a wealth of meta-information in XML documents. Query relaxation is more important for the XML model [14] than for the relational model because:

1. The schema in the XML model [14] is substantially larger and more complex than the schema in the relational model. Therefore, it is often unrealistic for users to understand the full schema and compose very complex queries. Thus, it is critical to be able to relax a user's query when the original query yields null or insufficient answers.

2. As the number of data sources available on the web increases, it is becoming increasingly common to build systems that gather data from the heterogeneous data sources. The structures of these data sources are different although using the same ontology for similar contents. Therefore, the capability to query against differently-structured data sources is becoming increasingly important [8, 9]. Query relaxation allows a query to relax its structure and matches data sources with relaxed structures.

Query relaxation in the XML model introduces new challenges than the relational database. Query relaxation in the relational model is basically focused on the value aspect. For example, for a relational query "*find a person with a salary range 50K – 55K*", if there is no answer or not enough answers available, it can be relaxed to a query "*find person with a salary range 45K - 60K.*" In the XML model, in addition to the value relaxation, a new type of relaxation called *structure relaxation* is introduced. Structure relaxation relaxes the nodes and/or edges of a query tree.

Further, we shall develop a methodology to provide automatic structure relaxations and to evaluate the effectiveness of XML structure relaxations.

A knowledge-based relaxation index structure called XML Type Abstraction Hierarchy (X-TAH) is introduced to provide scalable XML query relaxations. X-TAH is a hierarchical tree-like knowledge structure that builds multi-level knowledge representation about the XML data tree. X-TAH can be used to guide the XML query relaxation process.

The paper is organized as follows: Section 2 provides some background information which cover XML data model, query model and XML query relaxation types. Section 3 describes the system architecture that we are using for this year's INEX retrieval task. Query execution and query relaxation processes are presented in Section 4.

The experimental performance is discussed in Section 5. Finally we summarize our participation effort in INEX 03 and discuss future works in Section 6.

2. XML Data Model and Query Relaxation

We first briefly describe the XML data and query model and then introduce query relaxation types in the XML model.

2.1 Data Model and Query Model

An XML document can typically be represented as an ordered, labeled tree where nodes correspond to elements and attributes, and edges represent element inclusion relationships. Each node has a label which is the tag name of its corresponding element or attribute. Elements' text content or attributes' values become the values of their corresponding nodes. Similarly, a query against an XML document can be represented as a tree with two types of edges: a parent-child edge denoted as "/", or an ancestor-descendant edge denoted as "//".

Note that in the paper, we treat an attribute as a sub-element of an element and a reference IDREF as a special type of value.

2.2 Query Relaxation Types

In the XML model, there are two types of query relaxations, value relaxations and structure relaxations:

2.2.1 Value Relaxation

In the XML context, value relaxation involves expanding the value scope of certain nodes to allow the matching of additional answers. A value can be relaxed to a range of numeric values or a set of non-numeric values. Figure 1 illustrates an example of numeric value relaxation and an example of non-numeric value relaxation. The query in Figure 1b is a relaxed query for that in Figure 1a by a numerical value relaxation, and the query in Figure 1d is a relaxed query for that in Figure 1c by a non-numeric value relaxation.

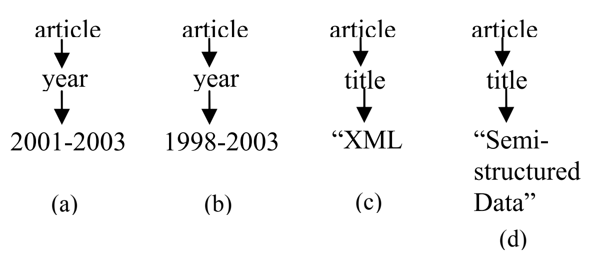


Figure 1: An example of value relaxation

2.2.2 Structure Relaxation

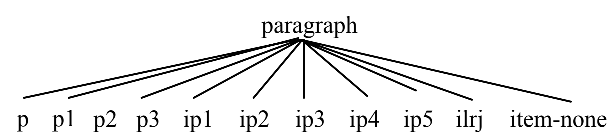
In XML context, structural relaxation is the process of relaxing the nodes and/or edges of a query tree. After the relaxation, a new query tree may have a different structure than the original query tree. There are three types of structural relaxations.

1) Edge Relaxation

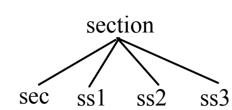
In an edge relaxation, a parent-child edge ('/') in a query tree can be relaxed to an ancestor-descendant edge ('//'). The semantics of edge relaxation is that while the original query finds answers with only a parent-child relationship, the new query will be able to find answers with an ancestor-descendent relationship which is a superset of a parent-child relationship. For example, query topic 69 */article/bdy/sec[about(.//st, "Information Retrieval")]* can be relaxed to */article/bdy//sec[about(.//st, "Informaiton Retrieval")]* by relaxing the structural relationship between node *bdy* and *sec* from "/" to "//".

2) Node Re-label

In this relaxation type, certain nodes can be re-labeled to similar or equivalent tag names according to the domain knowledge. For example, in INEX 03, domain experts have identified sets of equivalent tags as shown in Figure 2. With this domain knowledge, the query */article/bdy//sec [about(., "XML")]* can be relaxed to */article/bdy//section [about(., "XML")]* by generalizing node *sec*'s label to *section*.



(a) Equivalent names for paragraph-like tags



(b) Equivalent names for section-like tags

Figure 2: Domain knowledge for equivalent tags in INEX 03

3) Node Deletion

In this relaxation type, certain nodes can be deleted while preserving the "superset" property. When a node v is a leaf node, it can simply be removed. When v is an internal node, the children of node v will be connected to the parent of v with ancestor-descendant edges ("//"). For example, a query */article/bdy/sec[about(., "Information Integration")]* can be relaxed to */article//sec [about(.,"Information Integration")]* by deleting internal node *bdy* so that a section in an article's appendix talking about "Information Integration" can also be returned as an approximate answer.

3. The CoXML Framework

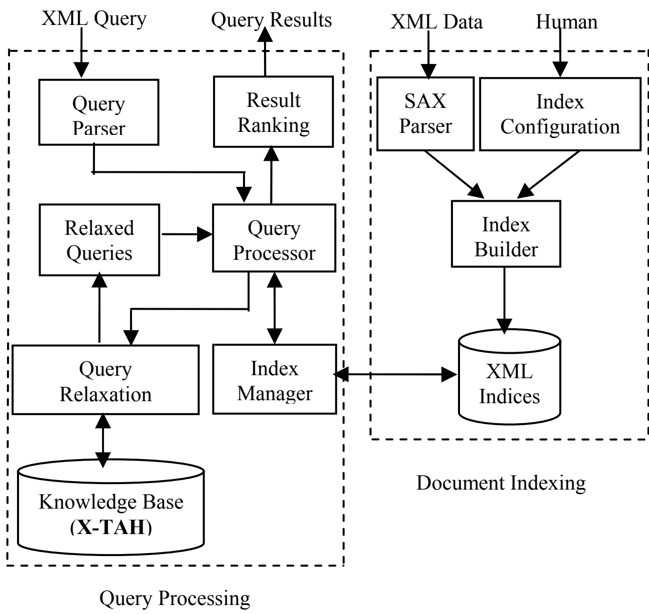


Figure 3: The CoXML System Architecture

Figure 3 shows the cooperative XML query answering system (CoXML) which performs two types of functions: document indexing and query processing as discussed in the following:

Document Indexing

While a *SAX parser* parses XML documents, the *Index Builder* builds indices on these data based on the index configurations provided by the *Index Configurations* module (Section 3.1). The *Index Builder* module builds several types of indices (refer to section 3.2) for query processing.

Query Processing

An XML query is first parsed by the *Query Parser* to check its correctness. If the query is invalid, it will be returned to the user with the error information. Otherwise, the *Query Processor* will consult the *Index Manager* to load the corresponding indices for processing the query. If there are enough XML answers returned, the *Result Ranking* module will rank the results based on their relevancy to the query and return the ranked results to the user. If there is no answer or the available answers are not enough, the X-TAH that resides in the *Knowledge Base* will guide the *Query Relaxation Manager* to relax the query. Then the relaxed queries will be resubmitted to the *Query Processor* for answering. This process will be repeated until there are enough answers available or the query is no longer relaxable.

3.1 Index Configurations

XML documents in the INEX document collections are document-centric. There are two types of tags in these documents: 1) semantic tags, and 2) presentation tags. Semantic tags describe the semantics of the elements. For example, in the XML document fragment example in Figure 4, <article>, <bdy> and <sec> etc. are semantic tags for they encode the semantics of the elements, while <scp> is a presentation tag because it only senses the purpose of displaying: informing the browser to display the characters bounded by <scp> and </scp> in lower cases.

```
1. <article>
2.   <bdy>….
3.     <sec>
4.         <st>K<scp>NOWLEDGE</scp> B<scp>ASED</scp>
5.               S<scp>EMANTIC</scp> T<scp>EMPORAL</scp>
6.                 I<scp>MAGE</scp> M<scp>ODEL</scp>
7.       </st> …
8.     </sec> ….
10.   </bdy> ….
11. </article>
```

Figure 4 : An XML document fragment

Presentation tags sometimes are undesirable in query processing. For example, suppose a user wants to find an article that has a section with title containing a keyword "knowledge", which can be expressed in XQuery as *//article [contains(//sec/st, "knowledge")]*. Intuitively, the XML document fragment in Figure 4 is an answer because the title of the article's section (Line 4-7 in Figure 4) is "Knowledge Based…". However, if we do not ignore the markup <scp> and </scp> (Line 4), it will not be returned as an answer since the presentation tag <scp> separates "K" from "NOWLEDGE".

To support keyword and phrase matching in document-centric XML documents, it is necessary to ignore such presentation tags [2]. The set of ignorable tags during indexing is listed in the *Index Configurations* module (Figure 3). For XML documents in the INEX document collections, the list of ignorable tags for index configurations is shown in Table 1.

Category	Ignorable Tags
List-items	item-bold, item-both, item-bullet, item-diamond, item-letpara, item-mdash, item-numpara, item-roman, item-text
Lists	li, l1, l2, l3, l4, l5, l6, l7, l8, l9, la, lb, lc, ld, le, list, numeric-list, numeric-rbrace, bullet-list
Text font, style, size, emphasis etc	ss, tt, b, ub, it, rm, scp, u, sub, super, large, ariel, bi, bu, bui, cen, rom, h, h1, h1a, h2, h2a, h3, h4

Table 1: Index configurations used in INEX

3.2 Indexing XML documents

Each node in an XML data tree is represented by a triple (ID, size, level), where *ID* uniquely identifies the node in the XML document collections, *size* indicates the size of the sub-tree rooted at this node and *level* describes the node's height in the data tree. The advantage of this encoding scheme is that the hierarchical relationship (either parent-child or ancestor-descendant relationships) between any pair of nodes can be checked in constant time.

Values of nodes are processed in the following three steps:

1) A stop words list is used to delete words with weak discriminative powers (such as articles, pronouns, conjunctions and auxiliary words). This step significantly reduces the index size.

2) The Lovins stemmer [7] is used to derive word stems. For example, the stem for "clustering", "clusters" and "clustered" is "cluster". Word stemming reduces the index size and also supports keyword matching.

3) Each stem is represented as a pair of (ID, pos), where *ID* is the unique identifier of a node that contains this stem and *pos* is its relative position in the node's value. We assign a node's ID to its corresponding value to avoid the expensive join operations between nodes and their values and keep each stem's relative position in a node's value to support phrase matching.

To support efficient and scalable query processing, the *Index Builder* builds several types of indices, as listed below:

- Tag Name Index (tag name → name identifier)

  Each tag name *s* is mapped to a unique name identifier (NID) to minimize index size and computation overhead by eliminating string comparisons.

- Node Index (name identifier → (ID, size, level))

  Each name identifier is mapped to a set of nodes (in the form of (ID, size, level)) whose labels are the same as the one represented by the name identifier.

- Inverted Stem Index (stem s → (ID, pos))

  Each stem *s* is mapped to a set of pairs (ID, pos), where *ID* is the unique identifier of the node that contains stem *s* and *pos* is its relative position in the node's value.

- Text Size Index (ID → text size)

  For each node that has a value, its ID is mapped to the number of words it contains. The text size index is useful for result ranking (refer to section 4.4).

The indices for the XML document fragment in Figure 5 are shown in Table 2, which consist of four indices: a tag name index (Table 2.a); a node index (Table 2.b); an inverted stem index (Table 2.c) and a text size index (Table 2.d).

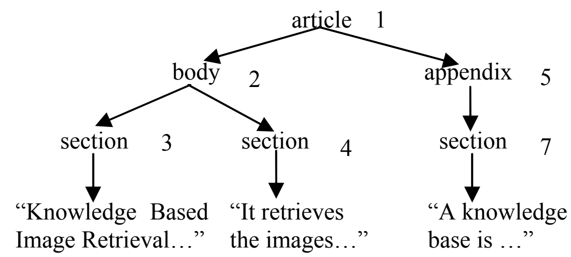


Figure 5 : An XML document fragment

Tag Name	NID
article	0
appendix	1
body	2
section	3

(a) A tag name index

NID	Nodes (ID, size, level)
0	(1, 5, 1)
1	(5, 1, 2)
2	(2, 2, 2)
3	(3, 0, 3) (4, 0, 3) (7, 0, 3)

(b) A node index

Stem	(ID, pos) pairs
bas	(3, 1) (7, 2)
imag	(3,2) (4, 3)
knowledg	(3, 0) (7, 1)
retrief	(3, 3) (4, 1)

(c) An inverted stem index

ID	Text Size
1	1000
2	600
…	…
7	100

(d) A text size index

Table 2: Indices for the XML fragment in Figure 5, a) maps a tag name to a unique name identifier; b) maps a name identifier to a set of nodes in the format of (ID, size, level); c) maps a stem to a set of (ID, pos) pairs d) maps a node ID to its text size

3.3 Knowledge Base

Knowledge Base is an important part in the system architecture, which facilitates XML query relaxation and consists of the following two parts:

1) Domain Ontology

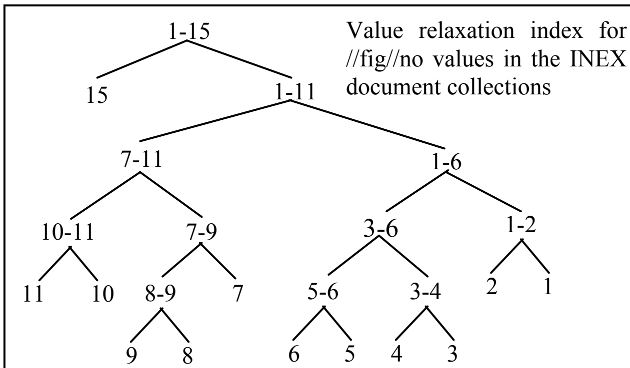
Domain ontology provides the semantic relationships among the tag names in an XML dataset, such as groups of equivalent or similar tag names which can guide the node re-label. For example, Figure 2 lists two sets of equivalent or similar tag names for INEX 03, one for section-like nodes (Figure 2a) and another for paragraph-like nodes (Figure 2b).

2) Knowledge-based XML Relaxation Index (X-TAH)

will iterate until either there are enough answers or the query is no longer relaxable.

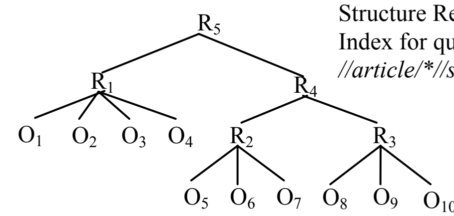
Query relaxation enlarges the search scope of query conditions which can be accomplished by viewing a query object at a higher conceptual level. To support query relaxation in the XML model, we are generating two types of relaxation index structures, XML Type Abstract Hierarchy - X-TAH: value relaxation index and structure relaxation index for guiding value and structure relaxations.

An X-TAH is a tree-like multi-level knowledge representation of the structure and value characteristics of an XML data tree. X-TAH can be automatically generated from a set of objects based on their inter-object distance [8]. Objects in an XML value relaxation index are values of XML elements and attributes, while objects in an XML structure relaxation index are structure fragments of XML data trees. X-TAH has two types of nodes: internal nodes and leaf nodes. This differentiates it from a traditional cluster which has no internal nodes. An internal node in an X-TAH is a representative that summarizes the characteristics of all the objects in that cluster, while a leaf node is an object that is either a value (in the XML value relaxation index) or a structure fragment of an XML data tree (in the XML structure relaxation index). For example, Figure 6 is an X-TAH for the values of *//fig//no* in the INEX document collections. Figure 7 is an X-TAH for structure relaxation for *//article/*//section*.



Value relaxation index for //fig//no values in the INEX document collections

Figure 6: An Example of value relaxation index (X-TAH)



Structure Relaxation Index for query pattern *//article/*//section*

O_1: //article/bdy/sec O_2: //article/bdy/sec/ss1

O_3: //article/bdy/sec/ss1/ss2 O_4: //article/bdy/sec/ss1/ss2/ss3

O_5: //article/bm/sec O_6: //article/bm/sec/ss1

O_7: //article/bm/sec/ss1/ss2 O_8: //article/bm/app/sec

O_9: //article/bm/app/sec/ss1 O_{10} : //article/bm/app/sec/ss1/ss2

R_1 : //article/bdy//section R_2 & R_4: //article/bm//section

R3 : //article/bm/app//section R5 : //article//section

Figure 7: An Example of structure relaxation index (X-TAH)

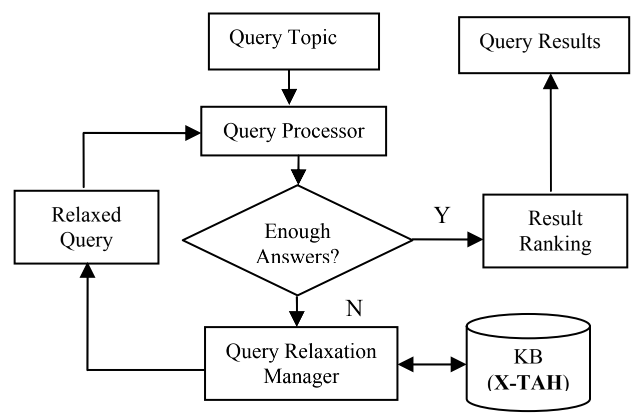


Figure 8: The control flow of CoXML query processing

4. Query Processing and Relaxation

The control flow for processing the INEX query topics is illustrated in Figure 8. First, each topic is translated into a tree representation that the *Query Processor* can follow and process. Next, the query is executed to produce a set of results. If there are enough answers produced, the *Result Ranking* ranks each result based on its relevancy to the query. Otherwise, the *Query Relaxation Manager* relaxes the query based on an X-TAH (*Knowledge Base*). The relaxed queries are then submitted to the *Query Processor* for deriving approximate answers. This process

4.1 Transformation of INEX Query Topics

The topic transformation can be accomplished by the following three steps:

1) Translating each INEX query topic expressed in XPath [15] into a tree representation. This is a straightforward step as most XPath expressions use tree structures.

2) Categorizing each term and phrase in the <title></title> part of a query into one of the three categories as defined below:

- PREFER (P)

  Any term or phrase prefixed by "+" belongs to this category.

- REJECT (R)

Any term or phrase prefixed with "-" or appearing after "!=" operator belongs to this category.

- NORMAL (N)

Any term or phrase not in the PREFER or REJECT category is classified in the NORMAL category.

3) Expanding a query's value predicates in the <title></title> part with terms and phrases in the <keyword></keyword> part that do not appear in the <title></title> part. Such terms and phrases are in the KEYWORD (K) category.

For example, the tree representation for the INEX 03 query topic 89 (Figure 9) with classified terms and phrases and expanded keyword value predicates is shown in Figure 10.

```
<inex_topic topic_id="89" query_type="CAS" ct_no="123">
<title>
//article[about(./bdy,'clustering "vector quantization" +fuzzy +k-
means +c-means -SOFM -SOM')]//bm//bb[about(.,'"vector
quantization" +fuzzy clustering +k-means +c-means') AND
about(./pdt,'1999') AND ./au/snm != 'kohonen']
 </title>
<description>
Find articles about vector quantization or clustering and return
bibliography details of cited publications about clustering and
vector quantization methods, from recent years, not authored by
Kohonen.
</description>
<narrative>
Bibliography elements of publications, preferably from around
2000 (1996 to 2002 is fine, descending relevance thereafter).
Preferred documents have reference to k-means or c-means
clustering. Not interested in publications where the author is
Kohonen, or in his work on self organizing feature maps (SOM
SOFM). The citing article and the cited publication should be about
clustering or vector quantization methods.
</narrative>
<keywords>
cluster analysis,adaptive clustering,Generalized Lloyd, LBG, GLA
</keywords>
</inex_topic>
```

Figure 9: INEX 03 Query Topic 89

4.2 Query Processing

After the topic translation, a query tree is sent to the *Query Processor* for execution. Several query processing strategies have been proposed for XML tree pattern queries [e.g.11, 10]. The basic idea of these query processing strategies is to decompose an XML tree pattern query into a set of basic structural relationships (i.e. parent-child relationship and ancestor-descendant relationship) between pairs of nodes. Query answers can be derived by first matching each of these basic structural relationships and then combing these basic matches. Matching each structural relationship is usually based on XML indices and structural join algorithms [10, 4 etc.]. We leverage on these query processing strategies for deriving the exact matched query answers with additional care for processing value constraints in a query tree.

As illustrated in section 4.1, each term and phrase in the <title></title> and <keyword></keyword> part of a query topic is classified into one of the four categories. The semantics for terms and phrases in the PREFER, NORMAL and KEYWORD categories are quite clear. However, the semantics for terms and phrases in the REJECT category is context sensitive. If a value predicate in a query contains only REJECT category terms and phrases, it is interpreted as "strictly MUST NOT". Otherwise it means "fuzzy MUST NOT". For example, for the query tree in Figure 10, the semantics for "R: SOFT, SOM" under node *bdy* is different from that for "R: Kohonen" under node *snm*. The semantics for the first one is that if an article's body (*bdy*) contains either term "SOFT" or "SOM", it is still an answer but with lower relevancy. However, the semantics for the second one is that if an author's surname (*snm)* contains the term "Kohonen", it will not be returned as an answer.

4.3 Query Relaxation

If there is no answer or not enough available answers, the *Query Processor* will call the *Query Relaxation Manager* to relax the query in the following three steps:

1) A set of relaxable conditions as well as their respective relaxation order are generated. For example, for INEX 03 query topic 85, *//article[.fm//yr >= 1998 and .//fig//no >9]//sec[about(.//p, 'VR, "virtual reality", "virtual environment", cyberspace "augmented reality"')]*, the set of relaxable conditions and their relaxation order may be assigned as: relaxing the value of figure numbers (*//article//figure/no > 9*) first and then relaxing the value of the article's year (*//article/fm/yr >= 1998*).

2) For each relaxable condition, a relaxation index (X-TAH) will be selected to guide the relaxation process. The *Query Relaxation Manager* will first examine the internal representatives to find the one that contains the exact or closest match against the relaxable condition and relax the query condition accordingly. There are two types of operations in an X-TAH: i) Generalization - moving up the hierarchy to enlarge the search scope; and ii) Specification – moving down the hierarchy to narrow the search scope. The query relaxation process may incur a sequence of Generalization and Specification operations.

3) The relaxed queries will be sent to the *Query Processor* to derive approximate answers. This relaxation process will continue until there are enough answers or the query is no longer relaxable.

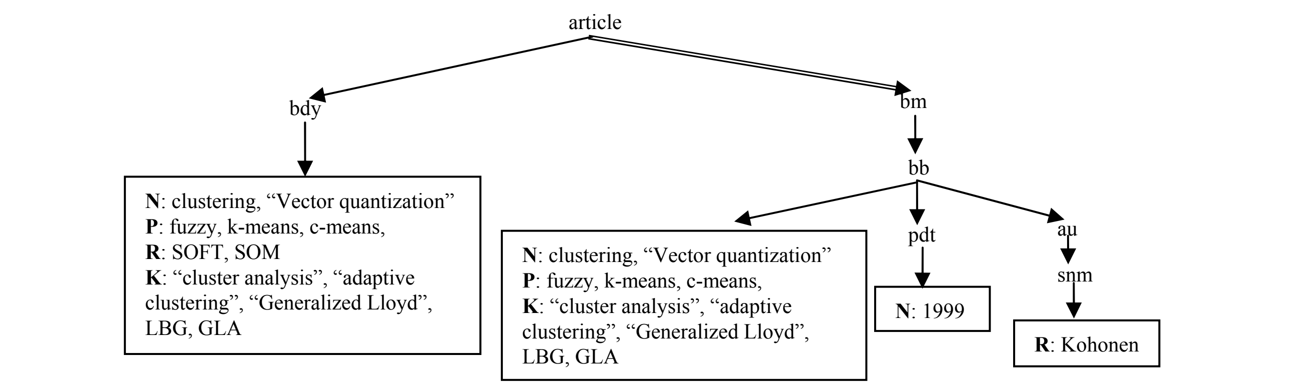


Figure10: The tree representation of query topic 89 in INEX 03

For example, in the query topic 85, to relax the query condition, *//article//figure//no > 9*, the *Query Relaxation Manager* will select the value relaxation index in Figure 6 to guide the relaxation process. The system first locates the closest matched internal representative, which is 8-9, and then relaxes the query condition to *//article/figure//no > 8* to derive approximate answers.

Similarly, to relax the structure constraint *//article/bdy/sec* in the query topic 69 (i.e. */article/bdy/sec[about(.//st, ‘“information retrieval”’)]*), the *Query Relaxation Manager* will first locate the closest matched internal representative, which is *//article/bdy//sec*, and will relax the query topic to *//article/bdy//sec[about (.//st, ‘“information retrieval”’)]*.

4.4 Result Ranking

The query results are ranked by the *Result Ranking* module before returning them to the user. Query results are ranked according to the following priorities: first query results from the original query and then approximate answers from the relaxed queries. The approximate answers are further ranked according to the relaxation order. For example, for the query topic 85, there are two relaxation conditions: 1) *//article//fig//no >* 9 and 2) *//article/fm//yr > 1998*. The relaxation order between them is to relax the first condition and then the second one. As a result, the approximate answers for the first relaxation condition are ranked before the approximate answers for the second relaxation condition.

For the query results in the same category, they are ranked according to the following formula:

$$rank_u = \sum_{\mathrm{i = P, N, K, R}} \left(\frac{w_i}{|C_i|} \sum_{j=1}^{|C_i|} \frac{frequency\ of\ term_{ij}}{Text\ Size\ of\ node\ u} \right)$$

where w_i is the weight assigned to one of the four categories C_i (i = P, N, K R); $|C_i|$ is the total number of stems (a phrase is counted as a term) in the category; *frequency of term*$_{ij}$ is the number of occurrence of $term_j$ from category C_i in node *u*; and *Text Size of node u* refers to the total number of words in node *u*, which can be accessed from the text size index.

5. Experimental Observations

We shall now discuss the experimental results based on two performance measurements: index size and query execution times.

The indices for all the INEX document collections occupy about 1.2GB, which is roughly about twice the size of the XML document collections. Four types of indices are built by the *Index Builder*: tag name index, node index, text size index, and inverted stem index. The first three are relatively small and the last one is quite large.

Query processing time depends on the following factors:

1) Number of stems and phrases in a query and their corresponding frequency in the XML data.

The query processing time depends on the number of stems and phrases a query contains and their corresponding frequencies in XML documents. More frequent stems and phrases require longer query processing time than less frequent ones.

2) Number of structure constraints in a query and their corresponding frequency in the XML data.

The required query processing time is sensitive to the number of structure constraints a query contains. It is also sensitive to their frequencies in XML documents. For example, a less frequent structure constraint, Q_1 *//article/fm//pdt*, can be processed much faster than a more frequent one Q_2 *//article/bdy//p*. (Q_1 returns the publication date (*pdt*) element of an article in its front matter part (*fm*) and Q_2 returns the paragraph (*p*) elements of an article in its body part (*bdy*)).

3) The level of query relaxation and the number of relaxable conditions existed in the query.

The more relaxable query conditions a query topic contains, the longer it takes to derive the approximate answers.

Depending on the complexity of its value and structure constraints, a content-and-structure (CAS) query takes from several seconds to over a minute to get exact matched answers. For a relaxable query, it might take several minutes to generated the relaxed queries and derive approximate answers.

6. Summary and Future Works

In this paper, we describe how we index INEX XML documents and extend the query relaxation technique to the XML model to support cooperative XML query answering.

During our INEX 03 investigation, several problems were discovered, which needs future investigations:

1) Index Configurations

Our current index configuration only contains a list of ignorable tags. We plan to support other index configurations, such as ignorable annotations in which both elements and their value can be ignored.

2) Uniform Value Index Scheme

In our current system, we index the elements' text content and attributes' values in XML documents uniformly. Each non-stop word is stemmed and is built an inverted stem index without considering of the value's characteristics. Such an index approach sometimes may derive undesirable results. For example, for a content-only (CO) query "*web, internet*", the document fragment "*<author> <snm>webb </snm></author>*" will be returned as an answer since "webb" and "web" share the same stem: "web". To avoid such undesirable results, we plan to work on a configurable value index framework which supports multiple value treatment options and index types based on the value's characteristics.

3) Ranking Functions

Our current system only supports relative ranking. Ranking functions for query results needed to be investigated to provide more user and context sensitive ranking.

4) Query Relaxation Language

No explicit relaxation constructs is available in a query topic for specifying the relaxable query conditions as well as their relaxation order. We plan to develop a cooperative query language that enables users to specify relaxation constructs in the queries.

ACKNOWLEDGEMENT

This work is supported by NSF Award IIS#: 0219442.

REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. Data on the web: from relations to semistructured data and XML. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1999.

[2] S. Amer-Yahia, M. Fernandez, D. Srivastava, Y. Xu. Phrase Matching in XML, VLDB 2003

[3] D. Chamberlin, J. Robie, and D. Florescu. Quit: An XML query language for heterogeneous data sources. In WebDB, May 2000

[4] S. Chien, Z. Vagena, D. Zhang, V. J. Tsotras, C. Zaniolo. Efficient Structural Joins on Indexed XML Documents, VLDB 02

[5] W. W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson. CoBase: A Scalable and Extensible Cooperative Information System. J. Intelligent Information Systems (JIIS), 6(2/3):223-259, May 1996.

[6] S. Liu and W. W. Chu. A Knowledge-Based Approach for Cooperative XML Query Answering, UCLA CS Dept. Technical Report, 2003

[7] J.B. Lovins. Development of a Stemming Algorithm. In Mechanical Translation and Computational Liguistics, 11(1-2), 11-31, 1968

[8] Y. Kanza, W. Nutt, and Y.Sagiv. Queries with Incomplete Answers over Semi-structured Data. In ACM PODS, 1999

[9] Y. Kanza and Y.Sagiv, Flexible Queries over Semi-structured Data, In ACM PODS, 2001

[10] D. Srivastava, S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. In ICDE 2002

[11] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, G. Lohman. On Supporting Containment Queries in Relational Database Systems, SIGMOD 2001

[12] XML http://www.w3.org/XML/

[13] XML-QL http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/

[14] XML Schema http://www.w3.org/xml/Schema

[15] XPATH http://www.w3.org/TR/xpath

[16] XQuery http://www.w3.org/TR/xquery

Appendix

Topic Development Guide 150

Retrieval Tasks and Run Submission Specification 158

Relevance Assessment Guide 162

Workshop Schedule 168

INEX'03 Guidelines for Topic Development

The aim of the INEX initiative is to provide means, in the form of a large test collection and appropriate scoring methods, for the evaluation of content-oriented XML retrieval. Within the INEX initiative it is the task of the participating organisations to provide the topics and relevance assessments that will contribute to the test collection. Each participating organisation therefore plays a vital role in this collaborative effort.

1. Introduction

Test collections, as traditionally used in information retrieval (IR), consist of three parts: a set of documents, a set of information needs called topics, and a set of relevance assessments listing for each topic the set of relevant documents.

A test collection for XML retrieval differs from traditional IR test collections in many respects. Although it still consists of the same three parts, the nature of these parts is fundamentally different. In IR test collections, documents are considered as units of unstructured text, topic statements are generally treated as collections of terms and/or phrases, and relevance assessments provide judgements whether a document as a whole is relevant to a query or not. XML documents, on the other hand, organise their content into smaller, nested structural elements. Each of these elements in the document's hierarchy, along with the document itself, is a retrievable unit. Regarding the topics, with the use of XML query languages, users of an XML retrieval system are able to combine both content and structural conditions within their information need and restrict their search to specific structural elements within an XML collection. Finally the relevance assessments for an XML collection must also consider the structural nature of the documents and provide assessments at different structural levels.

This guide deals only with the topics of the test collection and provides detailed guidelines for their creation for INEX 2003.

2. Topic creation criteria

Creating a set of topics for a test collection requires a balance between competing interests. It is a well-known fact that the performance of retrieval systems varies largely for different topics. This variation is usually greater than the performance variation of different retrieval methods on the same topic. Thus, to judge whether one retrieval strategy is in general more effective than another strategy, the retrieval performance must be averaged over a large, diverse set of topics. In addition, to be a useful diagnostic tool, the average performance of the retrieval systems on the topics can be neither too good nor too bad as little can be learned about retrieval strategies if systems retrieve no or only relevant documents.

When creating topics, a number of factors should be taken into account.

1. **The author of a topic should be either an expert or the very least be familiar with the subject area covered by the collection!** (Note that the author of a topic should also be the assessor of relevance!)
2. Topics should reflect what real users of operational systems might ask.
3. Topics should be representative of the type of service that operational systems might provide.
4. Topics should be diverse.
5. Topics may also differ in their coverage, e.g. broad or narrow topic queries.

3. Query types

As last year, in INEX 2003 we distinguish two types of query:

- *Content-only (CO) queries*: are requests that ignore the document structure and contain only content related conditions, e.g. only specify what a document/component should be about (without specifying what that component is). The need for this type of query for the evaluation of XML retrieval stems from the fact that users either do not care about the structure of the result components or are not familiar with the exact structure of the XML documents.

- *Content-and-structure (CAS) queries*: are topic statements, which contain explicit references to the XML structure, and restrict the context of interest and/or the context of certain search concepts.

4. Topic format

Both CO and CAS topics are made up of four parts:

- *Topic title*: a short version of the topic statement. It serves as a summary of both the content and structural requirements of the user's information need. The exact format of the topic title is discussed in more detail later in this section.
- *Topic description*: a one or two sentence natural language definition of an information need.
- *Narrative*: a detailed explanation of the topic statement and the description of what makes a document/component relevant or not.
- *Keywords*: a set of comma-separated scan terms that are used in the collection exploration phase of the topic development process (see Section 5.2) to retrieve relevant documents/components. Scan terms may be single words or phrases and may include synonyms, broader or narrower terms from those listed in the topic description or topic title.

The format of the topic title in 2003 is different to that used in INEX 2002. This year, the format is based on XPath, the proposed language for addressing parts of XML documents. The XPath notation is adopted in INEX 2003 to refer to the logical structure and the attributes of the XML documents. However, since XPath is a very rich and powerful language, we restrict ourselves to a subset of XPath, which has been identified by the INEX 2002 Topic Format working group as providing an "IR minimum". This subset corresponds (mainly) to the use of path expressions as described in Section 2 of the document XML Path Language (XPath) Version 1.0, W3C Working Draft 16 November 1999 (available at http://www.w3.org/TR/xpath). More precisely, the topic format will make use of Axes (Section 2.2), Predicates (Section 2.4), and will use the abbreviated syntax described in Section 2.5 of the aforementioned document.

Below are examples of path expressions (taken from Section 2.5 of the XPath 1.0 standard):

- `para` selects the `para` element children of the context node
- `*` selects all element children of the context node
- `@attr` selects the `attr` attribute of the context node
- `@*` selects all the attributes of the context node
- `para[1]` selects the first `para` child of the context node
- `*/para` selects all `para` grandchildren of the context node
- `/doc/chapter[5]/section[2]` selects the second `section` of the fifth `chapter` of `doc`
- `chapter//para` selects the `para` descendants element of the `chapter` element children of the context node
- `//para` selects all the `para` descendants of the document root and thus selects all `para` elements in the same document as the context node
- `//olist/item` selects all the `item` elements in the same document as the context node that have an `olist` parent
- `.` selects the context node
- `.//para` selects the `para` element descendants of the context node
- `..` selects the parent of the context node
- `../@lang` selects the `lang` attribute of the parent of the context node
- `para[@type='warning']` selects all `para` children of the context node that have a `type` attribute with value `warning`
- `para[@type='warning'][5]` selects the fifth `para` child of the context node that has a `type` attribute with value `warning`
- `para[5][@type='warning']` selects the fifth `para` child of the context node if that child has a `type` attribute with value `warning`
- `chapter[title='Introduction']` selects the `chapter` children of the context node that have one or more `title` children with string-value equal to `Introduction`

- `chapter[title]` selects the `chapter` children of the context node that have one or more `title` children
- `employee[@secretary and @assistant]` selects all the `employee` children of the context node that have both a `secretary` attribute and an `assistant` attribute

4.1. The *about()* function

In INEX, an "aboutness" concept, in the form of an *about(path, string)* function, has been added to the standard XPath syntax to deal with the content aspect of a user query. This concept was necessary in order to introduce the uncertainty inherent in IR into the world of the more exact-match XPath principle. The *about()* function should be used as the basis to provide a ranking of the retrieved elements with respect to content. Note that the *about(path,string)* clause is different from the *contains(path,string)* function of the XPath standard (see XPath 1.0, http://www.w3.org/TR/xpath). The latter returns true if the text value of the element defined by the path contains the string argument, and otherwise returns false. On the other hand, the *about()* function returns true if the element defined by the path argument is "about" the concept(s) defined by the string argument without having to actually contain the exact string value.

The *about()* function is usually applied to a context element, `CE`. This is described by the following syntax: `CE[about(path, string)]`. A context element is described using a standard XPath path expression (see the examples of path expressions in Section 4). It defines a "base node" against which relative paths, using the "." notation, can be defined within the *path* argument of the *about()* function. For example, `//article[about(.//sec,'"XML retrieval"')]` represents the request to retrieve articles that contain within them a section about "XML retrieval". Another example is `//article[about(.//sec, '"XML retrieval"') and about(.//sec,'evaluation')]`, which is a representation of the request to retrieve articles, which contain a section about "XML retrieval" and also a section on evaluation (where the two sections may be different or may be the same). We will look at more complex structures when we discuss the format of the CAS topic titles. The *string* parameter may contain a number of space-separated terms, where a term may be a single word or a phrase encapsulated in double-quotes. Furthermore, the symbols + and − may be used to express additional preferences for certain terms, where + is used to emphasise a concept and − is used to denote an unwanted concept. In summary, a *string* parameter may incorporate the following components:

- Terms (single words or phrases)
- "" (double-quotes to encapsulate phrases)
- \+ (expressing "must be about")
- − (denoting "must not be about")

The syntax of a *string* argument is:

```
String   ::=   term ' '
             | '+'term ' '
             | '-'term ' '
Term     ::=   single word
             | '"'phrase'"'
```

A *string* must be enclosed between single quotes. For example, `//article[about(.//sec, '"XML retrieval" +XML -"information retrieval"')]` would correspond to the request to retrieve articles that contain a section which is about XML retrieval but not about information retrieval, and where XML is characterised as an important concept.

Although at this point we are not talking about relevance assessment we would like to make a note here to emphasise that for relevance assessments the symbols + and − should be interpreted with a fuzzy "flavour" and not simply as must contain or must not contain conditions. Following on from the definition of the *about()* function above, a component may be considered relevant even if it does not contain the query term(s), but is "about" the concept(s) expressed by the query term(s). Similarly a component may be relevant even if it contains, for example, only one half of a phrase.

4.2 CO Topics

The topic title of a CO topic is a short, usually a 2-5 terms representation of the topic statement. Since CO topics ignore the document structure, their topic title will only consist of one *about()* clause applied to any context elements denoted by the path `//*`. The *path* argument of the *about()* function must be set to "." (dot) to refer to the context element. The *string* argument is made up of terms that best describe what the user is looking for. Take as an example the topic title `//*[about(., '"XML retrieval"')`, which is the representation of the request to retrieve any elements that are about "XML retrieval".

In order to simplify this syntax, we remove all components of the topic title that are the same for all CO topics (e.g. the context element, the *path* argument, etc.). As a result, we end up with just the *string* argument of the *about()* function, e.g. replacing `//*[about(path, string)]` with `string`, where we also ignore the single quotes.

The topic title of a CO topic is therefore defined as a set of space separated terms, optionally associated with the symbols + and −, where a term may be a single word or a phrase encapsulated in double-quotes. The syntax of the CO topic titles hence matches the syntax of the *string* argument specified above (Section 4.1).

Examples of CO topic titles

1. Retrieve documents/components about computer science degrees that are not master degrees:

```
<title>"computer science" +degree –master</title>
```

2. Retrieve document/components about summer holidays in England:

```
<title>"summer holiday" +England</title>
```

Example of a CO topic

```
<inex_topic topic_id="1" query_type="CO">
       <title>
              "summer holiday" "winter holiday" +"England"
       </title>
       <description>
              Winter or summer holidays in England.
       </description>
       <narrative>
              To be relevant, a document or component must contain
              information about winter or summer holidays in England.
       </narrative>
       <keywords>
              summer, winter, holiday, England, skiing, beach
       </keywords>
</inex_topic>
```

4.3 CAS Topic

The general structure of a CAS topic title is as follows:

```
CE [ filter ] CE [ filter ] … CE [filter] CE [filter]
```

`CE` refers to the context element. The series of context elements, where the first `CE` acts as the root node, describes a branch of an XML tree. Each context element is relative to the context element that precedes it in the sequence. This branch forms the path of the target element that is to be returned to the user. A filter is defined as a set of about clauses (e.g. *about(path, string)*) and other predicate clauses (e.g. `@yr = '2001'`), which are joined by Boolean expressions. The *path* argument of the *about()* function can be expressed relative to the context element by using the "." notation. For example, `//article[.//@yr = '2001']//sec[about(.,'+"XML retrieval"')`, is the expression of a request to retrieve sections about "XML retrieval" of articles written in 2001. This query has two context elements, namely `//article` and `//sec`, which together define the target element. `//article//sec`.

A filter may contain a set of *about()* functions and/or a set of standard XPath string operators: =, !=, >, <, >= and <=. The conditions expressed by these functions and operators can be combined using the Boolean operators: AND and OR, together with the use of parenthesis to group such conditions

together. For example, `//article[about(.//p,'+"holiday"') AND .//@yr='2002']`, retrieves articles that contain paragraphs about "holiday" and have a published date of 2002. Note that while the series of context elements must describe a branch of the XML tree, the filter components allow for the definition of content conditions on different branches of a tree within the context element. Take the earlier mentioned example (Section 4.1) of `//article[about(.//sec,'"XML retrieval"') and about(.//sec,'evaluation')]` requesting article elements, which contain a section about "XML retrieval" and also a section on "evaluation" (where the two sections may be different or may be the same). Here two independent branches of the tree rooted in `//article` are described.
`NOTE THAT FOR AN INEX CAS TOPIC, IT IS A REQUIREMENT THAT A FILTER CONTAINING AN ABOUT() FUNCTION MUST BE SPECIFIED FOR THE LAST CONTEXT ELEMENT!` Multiple target elements are not allowed in INEX 2003. Also note that specifying one context element only, and setting it to `//*`, while setting the *path* argument of the *about()* functions to ".", we arrive back at a CO topic title.

Examples of CAS topic titles[1]

1. Return section elements, which are about summer holidays, where the section element is a descendent of article element, and the article is from 2001 or 2002:

```
<title>
    //article[.//@yr = '2001' OR .//@yr = '2002']//sec[about(.,
    '"summer holidays"')]
</title>
```

The above query has two context elements, `//article` and `//sec`, each with their own filters, one containing a standard Xpath predicate and the other containing an *about()* clause. The target element defined by the above query is `//article//sec`.

Note that the following query is not a valid INEX query as it does not contain an *about()* function:

```
<title>
    //article[.//@yr = '2001' OR .//@yr = '2002']
</title>
```

The following query is not valid because there is no filter applied to the last context element (e.g. //sec):

```
<title>
    //article[.//@yr = '2001' OR about(., '"summer holiday"']//sec
 </title>
```

In the remainder of the examples for simplicity we ignore the `<title> </title>` tags.

2. Retrieve all articles that were published in 2001 and are about summer holidays:

```
//article[.//@yr = '2001' AND about(./, '"summer holidays"')]
```

3. Return article elements published in 2001 that contain section elements about summer holidays:

```
//article[.//@yr = '2001' AND about(.//sec, '"summer holidays"')]
```

4. Return articles from 2001, which contain section elements about summer holidays or section elements about winter holidays:

```
//article[.//@yr = '2001'  AND (about(.//sec, '"summer holidays"')OR
about (.//sec, '"winter holidays"'))]
```

A query requesting articles from 2001 containing section elements about summer and winter holidays would be as follows:

```
//article[.//@yr = '2001' AND (about(.//sec, '+"summer holidays"
+"winter holidays"')]
```

5. Return section elements, which are about summer holidays and that are the grandchildren of article elements, where the article is from 2001 or 2002:

[1] Note that these examples do not conform to the structure or content of the INEX document collection

```
//article[.//@yr = '2001' or .//@yr = '2002']/*/sec[about(., '"summer
holidays"')]
```

6. Return articles on XML retrieval, where the article contains a section on evaluation:

```
//article[about(., '"XML retrieval"') AND about(.//sec, 'evaluation')]
```

7. Retrieve articles that were published in 2002 and contain a section about “XML retrieval”:

```
//article[about(.//sec, '"XML retrieval"') AND .//@yr='2002']
```

8. Retrieve those sections of articles published in 2002 that are about “XML retrieval”:

```
//article[.//@yr='2002']//sec[about(.//sec, '"XML retrieval"')]
```

9. Retrieve those sections of articles that contain both a figure about “CORBA” and a figure caption about “XML”:

```
//article//sec[about(.//fig, 'CORBA') AND about(.//figc, 'XML')]
```

Example of a CAS topic

```
<inex_topic topic_id="2" query_type="CAS">
	<title>
		//article[.//@yr = '2001' OR .//@yr = '2002']//sec[about(.,
		'"summer holidays"')]
	</title>
	<description>
		Summer holidays either of 2001 or of 2002.
	</description>
	<narrative>
		Return section elements, which are about summer holidays, where
		the sections is descendent of article element, and the article
		is from 2001 or 2002.
	</narrative>
	<keywords>
		summer, holiday, 2001,2002
	</keywords>
</inex_topic>
```

4.4. Equivalent tags

This section lists the defined set of "equivalent" tags (alias/role/metedata) in the INEX test collection. We are proposing aliases for the following classes of nodes (identified directly from the DTD):

Paragraph-like nodes: ilrj|ip1|ip2|ip3|ip4|ip5|item-none|p|p1|p2|p3
Section nodes: sec|ss1|ss2|ss3
List environments: dl|l1|l2|l3|l4|l5|l6|l7|l8|l9|la|lb|lc|ld|le|list|numeric-list|numeric-rbrace|bullet-list
Headings: h|h1|h1a|h2|h2a|h3|h4

4.5. Topics DTD

The overall structure of the INEX topics is given in the DTD below (Note that additional attributes may be added at a later stage).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT inex_topic (title, description, narrative, keywords)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT narrative (#PCDATA)>
<!ELEMENT keywords (#PCDATA)>
<!ATTLIST inex_topic
  topic_id   CDATA  #REQUIRED
  query_type CDATA  #REQUIRED
>
```

5. Procedure for topic development

Each participating group will have to submit **3 CO and 3 CAS** queries by the **30 May 2003** by filling in the Candidate Topic Form (one per topic) at

http://inex.is.informatik.uni-duisburg.de:2003/internal/TopicSubmission.html

This section outlines the procedures involved in the development of candidate topics. There are four steps in creating topics for a test collection: 1) creating the initial topic statements, 2) exploring the collection, 3) selecting final set of topics, and 4) refining the topic statements.

5.1. Initial topic statements

In this step, you should create a one or two sentence description of the information you are seeking. This should be a simple description of the information need without regard to retrieval system capabilities or document collection peculiarities. This should be recorded in the topic description field.

Use either a printout or directly the on-line version of the Candidate Topic Form to record all information on a topic you are creating.

5.2. Collection exploration

In this step the initial topic statements are used to explore the document collection in order to obtain an estimate of the number of relevant documents/elements in the collection and to evaluate whether this topic can be judged consistently in the assessment phase. You may use any retrieval engine for this task, including your own or HyRex (HyRex can be accessed via http://inex.is.informatik.uni-duisburg.de:2003/internal/#topics).

Using the Candidate Topic Form record the set of keywords that you use for retrieval (make sure to record all the keywords from all iteration of your search or if you use query expansion strategies the query terms generated by the process). You should try and make your search queries (e.g. set of keywords) as expressive as possible for the kind of documents you wish to retrieve: think of the words that would make good scan words when assessing, and use those as your query keywords.

Next, judge the top 25 documents/components of your retrieval result. Using the Candidate Topic Form record the number of found relevant components and the XPath path representing each relevant element. If you have found less than 2 or more than 20 relevant components within the top 25 results, you should abandon the topic and start with a new one! If you have found at least 2 relevant components and no more than 20, perform a feedback search (don't forget to record the terms (if any) that you decide to add to your query keywords). Judge the top 100 (some of them you will have judged already), and record the number of relevant documents/components in Candidate Topic Form.

Finally write your detailed explanation on what makes a document/component relevant and record this in the narrative field of the topic. Make sure your description is as exhaustive as possible as there will be a couple of months gap before you will return to the topic for relevance assessments. The expectation is that by judging 100 documents/components you will have determined how you will judge the topic in the assessment phase. The narrative of the topic should reflect this.

To assess the relevance of a retrieved document/component use the following working definition: mark a document/component relevant if it would be useful if you were writing a report on the subject of the topic, or if it contributes towards satisfying your information need. Each document/component should be judged on it own merits. That is, a document/component is still relevant even if it is the thirtieth document/component you have seen with the same information. It is crucial to obtain exhaustive relevance judgements. It is also very important that your judgement of relevance is consistent throughout this task.

5.3. Refining topic statements

Refining the topic statement means finalising the topic title, description, keywords and narrative. Note that it should be possible to use each of the four parts of a topic in a stand-alone fashion (e.g. using only the title for retrieval, or only the description for filtering etc.).

Once you finished, submit the on-line Candidate Topic Form at

http://inex.is.informatik.uni-duisburg.de:2003/internal/TopicSubmission.html.

Make sure you submit all **6** candidate topics no later than the 30 May 2003.

5.4. Topic selection

From the received candidate topics, we (the clearinghouse) will then decide which topics to use such that a wide range of likely number of relevant documents is included. The data obtained from the collection exploration phase will be used as input to the topic selection process. We will then distribute final set of topics back to you to be used for the retrieval and evaluation.

We would like to thank you for your contribution.

Acknowledgements

The topic format proposed in this document is based on the outcome of a working group set up during the INEX 2002 workshop in Dagstuhl and the intense discussions on the INEX 2003 mailing list. We are very grateful for their contribution.

Authors:
Gabriella Kazai, Mounia Lalmas and Saadia Malik

06 May, 2003
Updated 12 May 2003

INEX’03 Retrieval Task and Result Submission Format Specification

Retrieval Task

The retrieval task to be performed by the participating groups of INEX'03 is defined as the ad-hoc retrieval of XML documents. In information retrieval literature, ad-hoc retrieval is described as a simulation of how a library might be used, and it involves the searching of a static set of documents using a new set of topics. While the principle is the same, the difference for INEX is that the library consists of XML documents, the queries may contain both content and structural conditions and, in response to a query, arbitrary XML elements may be retrieved from the library. Within the ad-hoc retrieval task we define the following three sub-tasks:

CO: Content-oriented XML retrieval using content-only (CO) queries. As described in the INEX’03 Topic Development Guide, CO queries are requests that ignore the document structure and contain only content related conditions, e.g. only specify what a document/component should be about (without specifying what that component is). The need for this type of query for the evaluation of XML retrieval stems from the fact that users may not care about the structure of the result components or may not be familiar with the exact structure of the XML documents. In this task, it is left to the retrieval system to identify the most appropriate XML elements to return to the user. These elements are components that are most specific and most exhaustive with respect to the topic of request. Most specific here means that the component is highly focused on the topic, while exhaustive reflects that the topic is exhaustively discussed within the component.

SCAS: Content-oriented XML retrieval based on content-and-structure (CAS) queries, where the structural constraints of a query must be strictly matched. 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). In this task, the user’s query is considered as an exact formulation of his/her information need, where the structural conditions specified within the query must be satisfied exactly by the retrieved components.

VCAS: Content-oriented XML retrieval based on content-and-structure (CAS) queries, where the structural constraints of a query can be treated as vague conditions. This task deviates from the previous one in that XML elements ‘structurally similar’ to those specified in the query may be considered correct answers. The idea behind this sub-task is to allow the evaluation of XML retrieval systems that aim to implement a more fuzzy approach to XML retrieval, where not only the content conditions within a user query are treated with uncertainty but also the expressed structural conditions. These systems aim to return components that contain the information sought after by the user even if the result elements do not exactly meet the structural conditions expressed in the query.

The actual search queries put to the retrieval engines (e.g. used to search the document collection) may be generated either manually or automatically from any part of the topics, with the exception of the narrative. Please note that at least one submitted run for each sub-task must be with the use of automatic queries.

Result Submission

For each sub-task up to 3 runs may be submitted. The results of one run must be contained in one submission file (e.g. up to 9 files can be submitted in total). A submission may contain up to **1500** retrieval results for each of the INEX topics included within that sub-task (e.g. for the CO sub-task only submit the search results obtained for the CO topics).

Submission format

For relevance assessments and the evaluation of the results we require submission files to be in the format described in this section. The overall submission format is defined in the following DTD:

```
<!ELEMENT inex-submission          (description, topic+)>
<!ATTLIST inex-submission
     participant-id          CDATA        #REQUIRED
     run-id                  CDATA        #REQUIRED
     task      ( CO | SCAS | VCAS )       #REQUIRED
     query     (automatic | manual)       #REQUIRED
     topic-part (T | D | K | TD | TK | DK | TDK)  #REQUIRED
>
<!ELEMENT description        (#PCDATA)>
<!ELEMENT topic              (result*)>
<!ATTLIST topic
     topic-id                CDATA        #REQUIRED
>
<!ELEMENT result             (file, path, rank?, rsv?)>
<!ELEMENT file               (#PCDATA)>
<!ELEMENT path               (#PCDATA)>
<!ELEMENT rank               (#PCDATA)>
<!ELEMENT rsv                (#PCDATA)>
```

Each submission must specify the following information:

- `participant-id`: the participant ID of the submitting institute (available at http://inex.is.informatik.uni-duisburg.de:2003/inex03/servlet/ShowParticipants),
- `run-id`: a run ID (which must be unique for the submissions sent from one organisation – also please use meaningful names as much as possible),
- `task`: the identification of the task (e.g. `CO`, `SCAS` or `VCAS`),
- `query`: the identification of whether the query was constructed automatically or manually from the topic,
- `topic-part`: the specification of whether the automatic or manual query was generated from the topic title only (`T`), the topic description only (`D`), the keywords only (`K`), the combination of the topic title and the topic description (`TD`), the combination of the topic title and the keywords (`TK`), the combination of the topic description and keywords (`DK`), or the combination of the topic title, topic description and keywords (`TDK`).

Furthermore each submitted run must contain a (brief) `description` of the retrieval approach applied to generate the search results.

A submission should then contain a number of `topics`, each identified by its topic ID (as provided with the topics). For each topic a maximum of **1500** `result` elements may be included. A result element is described by a `file` name and an element `path` and it may include `rank` and/or retrieval status value (`rsv`) information.

Before detailing these elements, below is a sample submission file:

```
<inex-submission participant-id="12" run-id="VSM_Aggr_06" task=”CO”
query=”automatic” topic-part=”TK”>
    <description>Using VSM to compute RSV at leaf level combined with
          aggregation at retrieval time, assuming independence and using
          acc=0.6.
    </description>
```

```
    <topic topic-id="01">
        <result>
            <file>tc/2001/t0111</file>
             <path>/article[1]/bm[1]/ack[1]</path>
            <rsv>0.67</rsv>
        </result>
        <result>
            <file>an/1995/a1004</file>
            <path>/article[1]/bdy[1]/sec[1]/p[3]</path>
            <rsv>0.1</rsv>
        </result>
        [ ... ]
    </topic>
    <topic topic-id="02">
        [ ... ]
    </topic>
    [ ... ]
</inex-submission>
```

Rank and RSV

The `rank` and `rsv` elements are provided for submissions based on a retrieval approach producing ranked output. The ranking of the result elements can be described in terms of

- Rank values, which are consecutive natural numbers, starting with 1. Note that there can be more than one element per rank.
- Retrieval status values (RSVs), which are positive real numbers. Note that there may be several elements having the same RSV value.

Either of these methods may be used to describe the ranking within a submission. If both rank and rsv are given, the rank value is used for evaluation. These elements may be omitted from a submission if a retrieval approach does not produce ranked output.

File and path

Since XML retrieval approaches may return arbitrary XML nodes from the documents of the INEX collection, we need a way to identify these nodes without ambiguity. Within INEX submissions, elements are identified by means of a `file` name and an element (node) `path` specification, which must be given in XPath syntax.

File names must be given relative to the INEX collection's "xml" directory (excluding the "xml" directory itself from the file path). The file path should use '/' for separating directories. Note that only article files (e.g. no "volume.xml" files) can be referenced here. The extension ".xml" must be left out. Example:

```
an/1995/a1004
```

Element paths are given in XPath syntax. To be more precise, only fully specified paths are allowed, as described by the following grammar:

Path	::=	'/' ElementNode Path | '/' ElementNode '/' AttributeNode | '/' ElementNode
ElementNode	::=	ElementName Index
AttributeNode	::=	'@' AttributeName
Index	::=	'[' integer ']'

Example:

```
/article[1]/bdy[1]/sec[4]/p[3]
```

This path identifies the element which can be found if we start at the document root, select the first "article" element, then within that, select the first "bdy" element, within which we select the fourth "sec" element, and finally within that element we select the third "p" element. Note that XPath counts elements starting with 1 and takes into account the element type, e.g. if a section had a title and two paragraphs then their paths would be given as: `../title[1], ../p[1]` and `../p[2]`.

When producing the XPath expressions of result elements, the equivalent-tags rules (see INEX'03 Guidelines for Topic Development) must be ignored, e.g. result elements must be identified in line with the original structure of the XML documents! For example, given the structure: `<sec><p>..</p><ip5>..</ip5><p>..</p></sec>)` the following XPaths should be generated: `/sec[1], /sec[1]/p[1], /sec[1]/ip5[1],` and `/sec[1]/p[2]`. Note that the same structure, taking into account the equivalent-tags rules, would result in the XPaths: `/sec[1], /sec[1]/p[1], /sec[1]/p[2],` and `/sec[1]/p[3]`. However, result elements identified by the latter XPaths will lead to incorrect evaluations of the submitted runs.

A result element is identified unambiguously using the combination of its file name and element path. Example:

```
<result>
    <file>an/1995/a1004</file>
    <path>/article[1]/bdy[1]/sec[1]/p[3]</path>
</result>
```

An application that can be used to check the correctness of a given path specification is available at
http://inex.is.informatik.uni-duisburg.de:2003/browse.html
Note that this application requires the input of a file name and element path. If these are correctly given, the specified XML element within its container article element will be displayed.

Result Submission Procedure

An online submission tool will be provided. Details on how to submit will be circulated as part of a separate document in the near future.

July 23, 2003
Gabriella Kazai, Mounia Lalmas, Norbert Goevert and Saadia Malik

INEX'03 Relevance Assessment Guide

1. Introduction

During the retrieval runs, participating organisations evaluated the 66 INEX'03 topics (36 content-only and 30 content-and-structure queries) against the IEEE Computer Society document collection and produced a list (or set) of document components (XML elements[1]) as their retrieval results for each topic. The top 1500 components in a topic's retrieval results were then submitted to INEX. The submissions received from the different participating groups have now been pooled and redistributed to the participating groups (to the topic authors whenever possible) for relevance assessment. Note that the assessment of a given topic should not be regarded as a group task, but should be provided by one person only (e.g. by the topic author or the assigned assessor).

The aim of this guide is to outline the process of providing relevance assessments for the INEX'03 test collection. This requires first a definition of relevance for XML retrieval (Section 2), followed by details of what (Sections 3) and how (Section 4) to assess. Finally, we describe the on-line relevance assessment system that should be used to record your assessments (Section 5).

2. Relevance dimensions: exhaustivity and specificity

Relevance in INEX is defined according to the following two dimensions:

- ***Exhaustivity (e-value for short),*** *which describes the extent to which the document component discusses the topic of request.*
- ***Specificity (s-value for short),*** *which describes the extent to which the document component focuses on the topic of request.*

To assess exhaustivity, we adopt the following 4-point scale:

> **0: Not exhaustive,** the document component does not discuss the topic of request at all.
> **1: Marginally exhaustive**, the document component discusses only few aspects of the topic of request.
> **2: Fairly exhaustive**, the document component discusses many aspects of the topic of request.
> **3: Highly exhaustive**, the document component discusses most or all aspects of the topic of request.

To assess specificity, we adopt the following 4-point scale:

> **0: Not specific**, the topic of request is not a theme of the document component.
> **1: Marginally specific**, the topic of request is a minor theme of the document component
> **2: Fairly specific**, the topic of request is a major theme of the document component.
> **3: Highly specific**, the topic of request is the only theme of the document component.

A document component can be assessed as highly exhaustive (e-value 3) even if it is not specific to the topic of request – that is, the topic of request can be a major theme (s-value 2) or a minor theme (s-value 1) of the component – as long as all or most aspects of the topic is discussed (e.g. a component may be highly exhaustive to the topic regardless of how much additional, irrelevant information it contains). Similarly, a document component can be assessed as highly specific (s-value 3) even if it discusses many (e-value 2) or only a few (e-value 1) aspects of the topic - as long as the topic of request is the only theme of the component. However, a document component that does not discuss the topic of request at all (e-value 0) must have an s-value of 0, and vice versa.

[1] The terms document component and XML element are used interchangeably.

3. What to judge

Depending on the topic, a pooled result set may contain initially between 500 and 1,500 document components of 500 - 510 articles, where a component may be a title, paragraph, section, or whole article etc.

Traditionally, in evaluation initiatives for information retrieval, like TREC, relevance is judged on document level, which is treated as the atomic unit of retrieval. In XML retrieval, the retrieval results may contain document components of varying granularity, e.g. paragraphs, subsections, sections, articles etc. Therefore, to provide comprehensive relevance assessment for an XML test collection, *it is necessary to obtain assessment for the different levels of granularity*.

This means that if you find, say, a section of an article relevant to the topic of the request, you will then need to provide assessment - both with regards to exhaustivity and specificity - for the found relevant component, for all its ascendant elements until you reach the article component, and for all its descendant elements until you have identified all relevant sub-components.

Such comprehensive assessments are necessary as it is demonstrated by the following example. Consider the XML structure in Figure 1. Let us say that you judged Section C, the document component that encapsulates all text fragments relevant to the topic, as highly exhaustive (e-value 3) and fairly specific (s-value 2). Given only this single assessment it would not be possible to deduce the exhaustivity and specificity levels of the ascending or descending elements. For example, Body D and Article E may be judged fairly or marginally specific depending on the volume of additional, irrelevant information contained within the sections other than Section C. Looking at the sub-components of Section C, it is clear that no conclusions can be drawn from Section C's assessment regarding the exhaustivity or specificity levels of its sub-components. For instance, both Sub-Sections A and B may be marginally, fairly or highly exhaustive, and smaller components, such as Paragraph 3, could even be irrelevant.

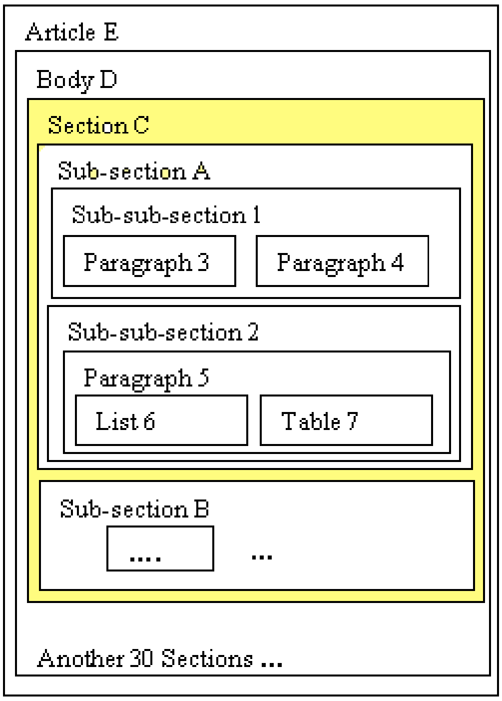


Figure 1. Example XML structure and result element

As a general rule it can be said that the exhaustivity level of a parent element is always equal to or greater than the exhaustivity level of its children elements. This is due to the cumulative characteristics of exhaustiveness. For example, the parent of a highly exhaustive element will always be highly exhaustive since the child element already discusses all or most aspects of the topic. Another rule for the exhaustivity dimension is that the parent of non-exhaustive child elements (i.e. all with e-value 0) will also be not exhaustive (e-value 0). A rule regarding specificity is that an element has an s-value that is greater than 0 if one of its child elements has an s-value different from 0, and less or equal to the maximum s-value of all its child elements. For instance, suppose that a parent element has tiny child element with s-value 1 and a large child element with s-value 2, then the s-value of that parent element will be 1 or 2. However, besides these general rules, no specific rules exist that would automate all the assessment of ascendant and descendant elements of relevant components. Therefore, you will need to explicitly judge all elements that contain relevant information. This is the only way to ensure both exhaustive and consistent relevance assessments.

4. How to judge

To assess the exhaustivity and specificity of document components, we recommend a three-pass approach.

- During the first pass, you should skim-read the whole article (that a result element is a part of - even if the result element itself is not relevant!) and identify any relevant information as you go along. The on-line system will assist you in this task by highlighting keywords within the article (see Section 5).
- In the second pass, you should assess the exhaustivity and specificity of the relevant components (i.e. identified in the first phase), and that of their ascendant and descendant XML elements.
- To ensure exhaustive assessments, in the third phase, you should assess the exhaustivity and specificity of the descendant XML elements of all elements that have been assessed as relevant during the second phase.

The on-line assessment system (see Section 5) will identify for you all elements that have to be assessed for phases 2 and 3.

During the relevance assessment of a given topic, all parts of the topic specification should be consulted in the following order of priority: narrative, topic description, topic title and keywords. The narrative should be treated as the most authoritative description of the user's information need, and hence it serves as the main point of reference against which relevance should be assessed. In case there is conflicting information between the narrative and other parts of a topic, the information contained in the narrative is decisive. The keywords should be used strictly as a source of possibly relevant cue words and hence only as a means of aiding your assessment. You should not rely only on the presence or absence of these keywords in document components to judge their relevance. It may be that a component contains some or maybe all the keywords, but is irrelevant to the topic of the request. Also, there may be components that contain none of the keywords yet are relevant to the topic. The same applies to the terms listed within the topic title!

In the case of content-and-structure (CAS) topics, the topic titles contain structural constraints in the form of XPath expressions. Although the structural conditions are there to impose a constraint on the structure, you are asked as an assessor to assess the elements returned for a CAS topic as whether they satisfy your information need (as specified by the topic) mainly with respect to the content criterion. Therefore, you should not assess an element as “not relevant” because the structural condition is not satisfied. In fact, your assessment of CAS topic should be very similar to that of content-only (CO) topics, although in the former the structural conditions may influence your assessment (to a small extent).

Note that some result elements are related to each other (ascendant/descendant), e.g. an article and some sections or paragraphs within the article. This should not influence your assessment. For example if the pooled result contains Chapter 1 and then Section 1.3, you should not assume that Section 1.3 is more relevant than Sections 1.1, 1.2, and 1.4, or that Chapter 1 is more relevant than Section 1.3 or vice versa. Remember that the pooled results are the product of different retrieval engines, which warrants no assumptions about the level of relevance based on the number of retrieved related components!

You should judge each document component on its own merits! That is, a document component is still relevant even if it the twentieth you have seen with the same information! It is imperative that you maintain consistency in your judgement during assessment. Referring to the topic text from time to time will help you maintain judgement consistency.

5. Using the on-line assessment system

There is an on-line relevance assessment system provided at:

http://inex.lip6.fr

which allows you to view the pooled result set of the topics assigned to you for assessment, to browse the IEEE-CS document collection and to record your assessments. Use your username and password to access this system.

After logging in, you will be presented with the Home page (see Figure 2) enlisting the topic ID numbers of the topics assigned to you for assessment (under the title "Choose a pool"). This page can always be reached by clicking on the **Home** link on any subsequent pages.

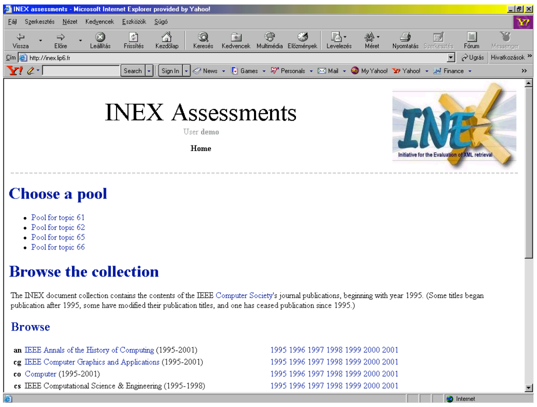


Figure 2. Home page of the assessment system

Clicking on a topic ID will display the pool main page for that topic (see Figure 3).

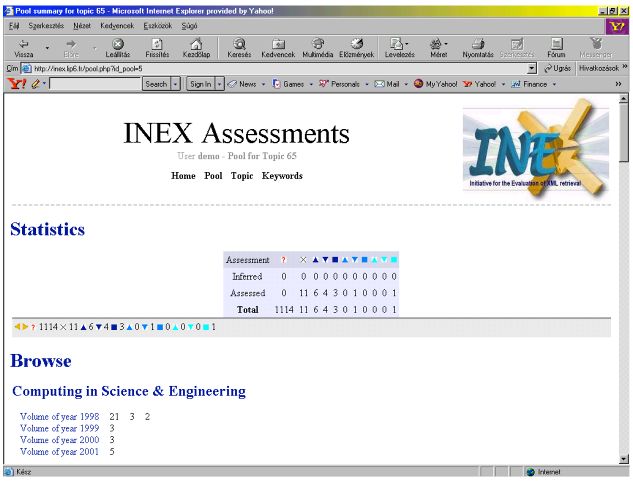


Figure 3. Pool main page

At the top of the pool main page the following links are shown: **Home**, **Pool**, **Topic** and **Keywords**. By clicking on the **Pool** link you can always return to this starting main pool page during your work. By selecting the **Topic** link you can display the topic text in a popup window. This is useful as it allows you to refer to the topic at any time during your assessment. The **Keywords** link allows you to edit a list of *coloured keywords* (cue words or phrases). This feature allows you to specify a list of words or phrases to be highlighted when viewing the contents of an article during assessment. These cue words or phrases can help you in locating potentially relevant texts within an article and will aid you in speeding up your assessment (so add as many relevant cue words as you can think of)! You may edit, add to or delete your list of keywords at any time during your assessment (remember, however, to reload the currently assessed document to reflect the changes). You may also specify the preferred highlighting colour for each and every keyword. After selecting the Keywords link, a popup window will appear showing a table of coloured cells. A border surrounding a cell signifies a colour that is already used for highlighting some keywords. You can move the mouse cursor over this cell to display the list of keywords that will be highlighted in that colour. To edit the list of words or phrases for a given colour, click on the cell of your choice. You will be prompted to enter a list of words or phrases (one per line) to highlight. Note that the words or phrases you specify will be matched against the text in the assessed documents in their exact form, i.e. no stemming is performed.

In the on-line assessment system, the following scheme is used:

1. *Exhaustivity level* is displayed in different shades of blue.
2. Geometric shapes are used for *specificity level*.

The tables below show the different icons used to indicate the relevance value of an XML element.

	Element to assess
	Element is not relevant

Specificity \ **Exhaustivity**	Highly exhaustive	Fairly exhaustive	Marginally exhaustive
Highly specific			
Fairly specific			
Marginally specific			

Table 1: Icons used to indicate relevance values

Note that all icons except the ? icon can be used by assessors to specify the relevance value (the exhaustivity and specificity level) of an element. The ? icon is used by the on-line assessment system only to mark components that need to be assessed.

This year, the assessment system makes use of two types of inference mechanisms to ensure exhaustive and consistent assessments: we refer to these as passive and active inferences. The passive type simply identifies new elements to be assessed based on those already assessed. For example, for any relevant element (e.g. any component assessed other than "not relevant"), the relevance of its child elements must be assessed, even if these were not part of the original assessment pool (i.e. have not been retrieved). With the application of the passive inference rules, these need-to-be-assessed components will be marked with the ? icon. Unlike the passive rules, the active inference rules are able to derive the relevance value of some elements. These inferred relevance values will be marked using a red border. For example, ⊠ denotes "inferred as not relevant", which is assigned to a component if all its child elements have been assessed as "not relevant".

The on-line assessment system provides three main views:

1. The pool view
2. The volume view
3. The article view

In each of these views, a *status bar* appears at the bottom of the window and shows statistics on the current view: how many elements have been assessed as highly exhaustive and highly specific, as highly exhaustive and fairly specific, etc; how many elements have been assessed as not relevant (□); and how many elements remain to be assessed (?). Only when no more elements remain to be assessed is the assessment for that view (pool / volume / article) complete.

In the status bar, three arrows may be used to navigate quickly between the elements to be assessed. The *up arrow* enables you to move from the article view to the volume view or from the volume view to the pool view (you move in the opposite direction by selecting a volume and then an article from the displayed lists). The *left arrow* can be used to go to the previous element to be assessed, while the *right arrow* to go to the next element to be assessed.

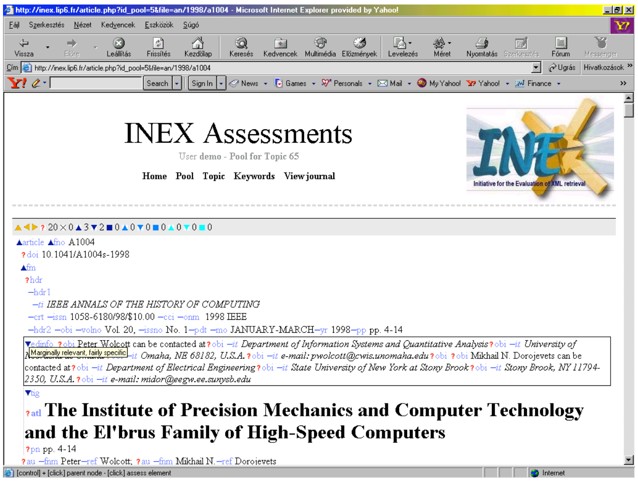


Figure 4. Article view

It is in the article view that elements can be assessed. The article view displays all the elements that form an article, whether these elements are to be assessed or not. In addition, the article view (see Figure 4) shows every XML tag in the article but tries to keep an eye-friendly view of the article. XML tags are displayed between brackets, in light blue, and according to their given (or inferred) assessments when applicable. For instance, an <abs> tag that has been assessed as "highly exhaustive and fairly specific" is displayed as follows:

[▼abs]

The mouse cursor becomes a cross when it is held over an XML tag name. You can then:

- Control-click to scroll to the parent element. The parent element will be highlighted in less than a second (in red).
- Click to display the assessment panel for the element. The assessment panel has three components: the path (first line), the current assessment (second line), and the set of 11 icons (reflecting all possible assignments shown in Table 1). Forbidden assessments (e.g. assessing a parent element as not relevant where one of its child elements is relevant) are displayed in a grey box. To assess the current element, click on the icon with the corresponding relevance value. To hide the panel, click anywhere else in the panel.

Note that you do not need to save your relevance assessments, as the on-line assessment system will automatically do this.

Acknowledgements

A working group was created to discuss the guidelines described in this document. Many thanks go to them for the lively discussion and many inputs. People involved include: Shlomo Geva, Norbert Goevert, Djoerd Hiemstra, Jaana Kekalainen, Shaorong Liu, Anne-Marie Vercoustre, and Arjen de Vries. Also, many thanks to Norbert Goevert for preparing the pools.

---

17 September 2003
Gabriella Kazai, Mounia Lalmas, and Benjamin Piwowarski

Monday, December 15, 2003

09.00-09.30 **Opening**
Norbert Fuhr

09.30-10.30 **Session 1: Australia**
Chair: Arjen de Vries

XPath Inverted File for Information Retrieval. Shlomo Geva, Murray Leo-Spork (Queensland University of Technology).

RMIT INEX experiments: XML Retrieval using Lucy/exist. Jovan Pehcevski, James Thom (RMIT University), Anne-Marie Vercoustre (CSIRO-ICT Centre).

Distributed XML Information Retrieval. Wayne Kelly, Shlomo Geva, Tony Sahama, Wengkai Loke (Queensland University of Technology).

10.30-11.00 Coffee

11.00-12.20 **Session 2: Israel and The Netherlands**
Chair: Patrick Gallinari

Retrieving the most relevant XML Components. Yosi Mass, Matan Mandelbrod (IBM Research Lab).

Searching in an XML corpus Using Content and Structure. Yiftah Ben-Aharon, Sara Cohen, Yael Grumbach, Yaron Kanza, Jonathan Mamou, Yehoshua Sagiv, Benjamin Sznajder, Efrat Twito (The Hebrew University).

The TIJAH XML-IR System at INEX 2003 (draft). J.A. List (CWI), V.Mihajlovic (University of Twente), A.P. De Vries (CWI), G. Ramírez (CWI), D. Hiemstra (University of Twente).

The University of Amsterdam at INEX 2003, Jaap Kamps, Maarten de Rijke, Börkur Sigurbjörnsson (University of Amsterdam)

12.30-14.00 Lunch

14.00-15.00 **Session 3: Japan and New Zealand**
Chair: Saadia Malik

Identifying and Ranking Relevant Document Elements. Andrew Trotman, Richard A. O'Keefe (University of Otago).

An Evaluation of INEX 2003 Relevance Assessments. Kenji Hatano (Nara Institute of Science and Technology), Hiroko Kinutani (Japan Science and Technology Agency), Masahiro Watanabe (The National Institute of Special Education), Yasuhiro Mori, Masatoshi Yoshikawa (Nogoya University), Shunsuke Uemura (Nara Institute of Science and Technology).

The Simplest Query Language That Could Possibly Work. Richard A. O'Keefe, Andrew Trotman (University of Otago).

15.00-15.30 **Workshop descriptions and grouping**
Mounia Lalmas

15.30-16.00 Coffee

16.00-17.30 **Workshop sessions**

18.00-20.00 Dinner

Tuesday, December 16, 2003

09.00-10.20 **Session 4: France and USA**
Chair: Birger Larsen

Bayesian Networks at INEX'03. Benjamin Piwowarski, Huyen-Trang Vu, Patrick Gallinari (LIP6).

IRIT at INEX 2003. Karen Sauvagnat, Gilles Hubert, Mohand Boughanem, Josiane Mothe (IRIT).

Using language Models for flat text queries in XML Retrieval. Paul Ogilvie, Jamie Callan (Carnegie Mellon University).

Cheshire II at INEX'03: Component and Algorithm Fusion for XML Retrieval. Ray R. Larson (University of California, Berkeley).

10.20-10.40 Coffee

10.40-12.00 **Session 5: Germany**
Chair: Tassos Tombros

XXL @ INEX 2003. Ralf Schenkel, Anja Theobald, Gerhard Weikum (Max-Planck Institüt für Informatik).

Applying the IRStream Retrieval Engine to INEX 2003. Andreas Henrich, Günter Robbert (University of Bayreuth), Volker Lüdecke (University of Bamberg).

HyREX at INEX 2003. Mohammad Abolhassani, Norbert Fuhr, Saadia Malik (University of Duisburg-Essen).

The SearX-Engine at INEX'03: XML enabled probabilistic retrieval. Holger Flörke (doctronic GmbH).

12.00-12.30 **Track Proposals for INEX'04**
Norbert Fuhr

12.30-14.00 Lunch

14.00-15.30 **Workshop sessions**

15.30-16.00 Coffee

16.00-17.30 **Workshop reports:**
Relevance (Jaana Kekalainen), Queries (Borkur Sigurbjornsson)

18.00-20.00 Dinner

Wednesday, December 17, 2003

09.00-10.30 **Workshop Reports**:
Metrics (Gabriella Kazai), On-line assessment tool (Benjamin Piwowarski)

10.30-11.00 Coffee

11.00-11.40 **Session 6: Scandinavia**
Chair: Shlomo Geva

Helsinki's EXTIRP @ INEX. Antoine Doucet, Lili Aunimo, Miro Lehtonen, Renaud Petit (University of Helsinki).

Using value-added document representations in INEX. Birger Larsen, Haakon Lund, Jacob K. Andresen and Peter Ingwersen (Royal School of Library and Information Science).

11.40-12.30 **INEX 2004, Any other business and Closing**
Mounia Lalmas

12.30-14.00 Lunch

14.00- **Walk (maybe)**

15.30-16.00 Coffee

18.00-20.00 Dinner