

# **INEX 2002 Workshop Proceedings**

December 9-11, 2002  
Schloss Dagstuhl  
International Conference and Research  
Center for Computer Science

**Organizers:**

Norbert Fuhr  
University of Duisburg

Norbert Gövert  
University of Dortmund

Gabriella Kazai  
Queen Mary University of London

Mounia Lalmas  
Queen Mary University of London

<http://qmir.dcs.qmul.ac.uk/inex/>

# Workshop on the Evaluation of XML Retrieval

## Index

Preface	IV
Workshop Programme	V
<i>Ray R. Larson</i> Cheshire II at INEX: Using a Hybrid Logistic Regression and Boolean Model for XML Retrieval	1
<i>Benjamin Piwowarski, Georges-Etienne Faure, Patrick Gallinari</i> Bayesian Networks and INEX	7
<i>Norbert Gövert, Mohammad Abolhassani, Norbert Fuhr, Kai Grossjohan</i> Content-oriented XML retrieval with HyRex	13
<i>Paul Ogilvie, Jamie Callan</i> Language Models and Structured Document Retrieval	18
<i>Maarten Marx, Jaap Kamps, Maarten de Rijke</i> The University of Amsterdam at INEX 2002	24
<i>Gabriella Kazai, Thomas Rölleke</i> A Scalable Architecture for XML Retrieval	29
<i>Torsten Grabs, Hans-Jörg Schek</i> ETH Zürich at INEX: Flexible Information Retrieval from XML with PowerDB-XML	35
<i>Andreas Henrich, Günter Robbert</i> Applying the IRstream Retrieval Engine for Structured Documents to INEX	41
<i>Johan List, Arjen P.de Vries</i> CWI at INEX 2002	47
<i>Djoerd Hiemstra</i> A database approach to INEX	53
<i>Holger Meyer</i> The Xircus Search Engine	59
<i>Shinjae Yoo</i> An XML Retrieval Model based on Structural Proximities	60
<i>Kenji Hatano, Hiroko Kinutani, Masahiro Watanabe</i> An Appropriate Unit of Retrieval Results for XML Document Retrieval	66
<i>Anne-Marie Vercoustre, James Thom, Alexander Krumpholz, Ian Mathieson, Peter Wilkins, Mingfang Wu, Nick Craswell, David Hawking</i> CSIRO INEX experiments: XML search using PADRE	72
<i>Yosi Mass, Matan Mandelbrod, Einat Amitay, Yoelle Maarek, Aya Soffer</i> Xjuru - an XML retrieval system at INEX'02	78
<i>Antoine Doucet, Helena Ahonen-Myka</i> Naïve clustering of a large XML document collection	84

<i>Shlomo Geva</i> XML Retrieval by Extreme File Inversion	90
<i>Richard M Tong</i> INEX 2002 Experiments	96
<i>Carolyn J Crouch, S Apte, H Bapat</i> An IR Approach to XML Retrieval based on Extended Vector Space Model	98
<i>Vo Ngoc Anh, Alistair Moffat</i> Compression and IR Approach to XML Retrieval (draft)	100
<b>Appendix</b>	
Guidelines for topic development	106
Guidelines for retrieval result submission	110
Guidelines for relevance assessments	113
Assessments and evaluation measures	117

## Preface

The aim of the workshop is to bring together researchers in the field of XML retrieval and in particular researchers who participated in the Initiative for the Evaluation of XML retrieval (INEX) during 2002. 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 INEX topics, the specification of the retrieval result submission format, the definition of the two relevance dimensions and the use of the on-line assessment system. Finally the workshops on evaluation measures aim to provide a forum to develop guidelines and procedures for the evaluation of XML retrieval systems based on the relevance dimensions employed in INEX.

## Monday, December 09, 2002

- 09.00-09.30 **Opening**  
Norbert Fuhr
- 09.30-10.30 **Session 1: IR approaches I**  
chair: [Johan List](#)  
Ray R. Larson: Cheshire II at INEX: Using a Hybrid Logistic Regression and Boolean Model for XML Retrieval  
Holger Flörke: doctronic GmbH at INEX  
Benjamin Piwowarski, Georges-Etienne Faure, Patrick Gallinari: Bayesian Networks and INEX
- 10.30-10.50 Coffee
- 10.50-12.10 **Session 2: IR approaches II**  
chair: [Shlomo Geva](#)  
Norbert Gövert, Mohammad Abolhassani, Norbert Fuhr, Kai Grossjohan: Content-oriented XML retrieval with HyRex  
Paul Ogilvie, Jamie Callan: Language Models and Structured Document Retrieval  
Maarten Marx, Jaap Kamps, Maarten de Rijke: The University of Amsterdam at INEX 2002  
Gabriella Kazai, Thomas Rölleke: A Scalable Architecture for XML Retrieval
- 12.10-12.30 **Discussions**
- 12.30-13.50 Lunch
- 13.50-15.10 **Session 3: DB approaches**  
chair: [Anne-Marie Vercoustre](#)  
Torsten Grabs, Hans-Jörg Schek: ETH Zürich at INEX: Flexible Information Retrieval from XML with PowerDB-XML  
Andreas Henrich, Günter Robbert: Applying the IRstream Retrieval Engine for Structured Documents to INEX  
Johan List, Arjen P.de Vries: CWI at INEX 2002  
Djoerd Hiemstra: A database approach to INEX
- 15.10-15.30 **Discussions**
- 15.30-16.00 Coffee
- 16.00-17.30 **Workshop: On-line assessment tools**  
chair: [Norbert Gövert](#)
- 18.00-20.00 Dinner
- 20.00-21.30 **Experiments: Relevance assessments**  
Moderator: [Gabriella Kazai](#)

## Tuesday, December 10, 2002

- 09.00-10.20 **Session 1: XML-specific approaches I**  
chair: [Djoerd Hiemstra](#)  
Holger Meyer: The Xircus Search Engine  
Shinjae Yoo: An XML Retrieval Model based on Structural Proximities  
Kenji Hatano, Hiroko Kinutani, Masahiro Watanabe: An Appropriate Unit of Retrieval Results for XML Document Retrieval  
Anne-Marie Vercoustre, James Thom, Alexander Krumpholz, Ian Mathieson, Peter Wilkins, Mingfang Wu, Nick Craswell, David Hawking: CSIRO INEX experiments: XML search using PADRE
- 10.20-10.40 Coffee
- 10.40-11.40 **Session 2: XML-specific approaches II**  
chair: [Thomas Rölleke](#)  
Yosi Mass, Matan Mandelbrod, Einat Amitay, Yoelle Maarek, Aya Soffer: Xjuru - an XML retrieval system at INEX'02  
Antoine Doucet, Helena Ahonen-Myka: Naïve clustering of a large XML document collection  
Shlomo Geva: XML Retrieval by Extreme File Inversion
- 11.40-12.00 **Discussions**
- 12.00-12.30 **Workshop: Queries**  
chair: [Norbert Gövert](#)
- 12.30-13.50 Lunch
- 13.50-14.40 **Workshop: Relevance dimensions**  
chair: [Gabriella Kazai](#)
- 14.40-15.30 **Workshop: Evaluation measures I**  
chair: [Norbert Fuhr](#)
- 15.30-16.00 Coffee
- 16.00-17.30 **Workshop: Evaluation measures II**  
chair: [Norbert Fuhr](#)
- 18.00-20.00 Dinner

## Wednesday, December 11, 2002

- 09.00-10.00 **Reports on Monday's group experiments**  
chair: [Gabriella Kazai](#)
- 10.00-10.30 **Lessons learnt in INEX 2002**  
Gabriella Kazai
- 10.30-11.00 Coffee
- 11.00-11.30 **Workshop: INEX 2003**  
chair: [Norbert Fuhr](#)
- 11.30-12.30 **Any other business and Closing**  
Norbert Fuhr
- 12.30-14.00 Lunch
- 14.00- **Discussions (optional)**
- 15.30-16.00 Coffee
- 18.00-20.00 Dinner



# Cheshire II at INEX: Using A Hybrid Logistic Regression and Boolean Model 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 Berkeley used in the INEX evaluation. The primary approach is the combination of a probabilistic methods using a Logistic regression algorithm for estimation of collection relevance and element relevance, along with Boolean constraints. The paper also discusses our approach to XML component retrieval and how component and document retrieval are combined in the Cheshire II system.

## Keywords

Information Retrieval, IR Evaluation, XML Retrieval

## 1. INTRODUCTION

The Cheshire II system originally was developed to provide a bridge from conventional online library catalogs to full-text online resources. Early research (circa 1990) with the system concentrated on the application of probabilistic ranked retrieval to short documents consisting primarily of bibliographic metadata and not the kinds of full-text document collections encountered today.

Over the past several years we have started to use the system to implement production-level services providing access to full-text SGML and XML document for a number of digital library systems in the United States and the United Kingdom, including the UC Berkeley Digital Library Initiative project sponsored by NSF, NASA and ARPA, The Archives Hub sponsored by JISC in the UK, The History Data Service of AHDS in the UK and the Resource Discovery Network in the UK. The Cheshire system is also being used to provide scalable distributed retrieval for consortia of institutions providing access to online catalogs and archival collections (the WARM system and the Distributed Archives Hub).

This paper will review the characteristics of the Cheshire II system. It will also examine the approach taken in applying this system to a collection of large XML documents as part of the Initiative for the Evaluation of XML retrieval (INEX), some observations on its performance and behavior in this area will be presented as well.

## 2. THE CHESHIRE II SYSTEM

When the Cheshire system was first conceived (in the late 1980's) the aim was to develop a "next-generation" online

library catalog system that could provide ranked retrieval based on probabilistic IR methods, while still supporting Boolean retrieval methods expected in the online catalog systems of that era. The decision was made early on to employ SGML as the single format used in the database (with conversion utilities to generate, for example, SGML versions of MARC format records).

Since that time the system has been constantly redesigned and updated to accommodate the information retrieval needs of a much broader world. The early choice of SGML made use of X1cgpTML a natural growth path, and the system remains one of the few to accomodate both XML and its more complex parent, SGML. The Cheshire II system now finds its primary usage in full text or structured metadata collections based on SGML and XML, often as the search engine behind a variety of WWW-based "search pages" or as a Z39.50 [13] server for particular applications.

The Cheshire II system includes the following features:

1. It supports SGML or XML as the primary database format of the underlying search engine, and also provides support for raw-text data or HTML linked to SGML/XML metadata records. MARC format records for traditional online catalog databases are supported using MARC to SGML conversion software developed for the project.
2. It is a client/server application where the interfaces (clients) communicate with the search engine (server) using the Z39.50 v.3 Information Retrieval Protocol. The system also provides a general Z39.50 Gateway which supports mapping of Z39.50 structured queries to local Cheshire databases and to relational databases.
3. The system include multiple clients, all of which are scriptable using either Tcl/Tk[10] or the Python language. These include a programmable graphical direct manipulation interface under X windows on Unix and Linux systems as well as a Windows implementation. There is also CGI interpreter version that combines client and server capabilities for low-overhead access to local collections. All of the interfaces permit searches of the Cheshire II search engine as well as any other z39.50 compatible search engine on the network.
4. It permits users to enter natural language queries and

these may be combined with Boolean logic for users who wish to use it.

5. It uses probabilistic ranking methods based on the Logistic Regression research carried out at Berkeley to match the user's initial query with documents and document components in the database. In some databases it can provide two-stage searching where a set of "classification clusters"[5] for the database is first retrieved in decreasing order of probable relevance to the user's search statement. The clusters can then be used to provide feedback about the primary topical areas of the query, and retrieve documents within the topical area of the selected clusters. This aids the user in subject focusing and in discriminating between variant treatments of a topic.
6. It supports distributed search across multiple collections using features of the Z39.50 protocol to harvest and create collection representatives that can then be searched probabilistically to select the collections most likely to contain relevant documents.
7. It supports relevance feedback searching where a user's selection of relevant documents is used to expand upon the initial query and automatically construct a new query derived from the contents of the selected documents.
8. It allows parts or "components" of complete SGML or XML documents (e.g., paragraphs) to be defined, indexed and retrieved as if they were individual documents, with separate indexes and ranking statistics used during retrieval.
9. It provides flexible document retrieval, including the ability to request any individual XPATH specification from any document selected during searching.

The Cheshire II search engine supports both probabilistic and Boolean searching. The 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.

The Cheshire II search engine supports both Boolean and probabilistic searching on any indexed element of the database. In probabilistic searching, a natural language query can be used to retrieve the documents that are estimated to have the highest probability of being relevant given the user's query. The search engine supports a simple form of relevance feedback, where any items found in an initial search (Boolean or probabilistic) can be selected and used as queries in a relevance feedback search.

The search engine also 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[11]) 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.

However, the primary functionality that distinguishes the Cheshire II search engine is support for probabilistic searching on any indexed element of the database. This means that a natural language query can be used to retrieve the documents or document components that have of highest probability of being relevant given the user's query. In both cluster searching and direct probabilistic searching of the database, the Cheshire II search engine supports a very simple form of *relevance feedback*, where any items found in an initial search (Boolean or probabilistic) can be selected and used as queries in a relevance feedback search.

The probabilistic retrieval algorithm used in the Cheshire II search engine is based on the *logistic regression* algorithms developed by Berkeley researchers and shown to provide excellent full-text retrieval performance in the TREC evaluation of full-text IR systems[3, 2, 1]. Formally, the probability of relevance given a particular query and a particular record in the database  $P(R | Q, D)$  is calculated and the documents or components are presented to the user ranked in order of decreasing values of that probability. In the Cheshire II system  $P(R | Q, D)$  is calculated as the "log odds" of relevance  $\log O(R | Q, D)$ , where for any events  $A$  and  $B$  the odds  $O(A | B)$  is a simple transformation of the probabilities  $\frac{P(A|B)}{P(\bar{A}|B)}$ . 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

$$\log O(R | Q, D) \approx c_0 \sum_{i=1}^S c_i X_i \quad (1)$$

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 [2] where the coefficients were estimated using relevance judgements from the TIPSTER test collection:

$X_1 = \frac{1}{M} \sum_{j=1}^M \log QAF_{t_j}$  . 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.

$X_3 = \frac{1}{M} \sum_{j=1}^M \log DAF_{t_j}$  . This 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.

$X_5 = \frac{1}{M} \sum_{j=1}^M \log IDF_{t_j}$ . This 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.

$X_6 = \log M$ . 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.

In recent work we have developed alternative forms of the ranking algorithms discussed above for application in distributed search collection selection. Because the "collection documents" used for our method of distributed search [6, 7] represent collections of documents and not individual documents, a number of differences from the usual logistic regression measures were used. In addition, analysis showed that different forms of the TREC queries (short titles only, longer queries including the concepts fields and the very long title, concepts, description and narrative) behaved quite differently in searching the distributed test collections, so three different regression equations were derived and applied automatically based on the length of the query.

Probabilistic searching, as noted above, requires only a natural language statement of the searcher's topic, and thus no formal query language or Boolean logic is needed for such searches. However, the Cheshire II search engine also supports complete Boolean operations on indexed elements in the database, and supports searches that combine probabilistic and Boolean elements. Although these are implemented within a single process, they comprise 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 the parallel search strategies merge the two result sets as a single set.

At present, combined probabilistic and Boolean search results are evaluated using the assumption that the Boolean retrieved set has an estimated  $P(R | Q_{bool}, D) = 1.0$  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 Boolean and probabilistic strategies is simply:

$$P(R | Q, D) = P(R | Q_{bool}, D)P(R | Q_{prob}, D)$$

where  $P(R | Q_{prob}, D)$  is the probability estimate from the probabilistic portion of the search, and  $P(R | Q_{bool}, D)$  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.

Besides allowing users greater flexibility, the motivation for having two search methods follows from the observation that no single retrieval algorithm has been consistently proven to be better than any other algorithm for all types of searches. By combining the retrieved sets from these two search strategies, can leverage the strengths and reduce the limitations of each type of retrieval system. In general, 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[4]. 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[12]. 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. The Boolean result set is combined with the ranked result set from the probabilistic query to form a single ranked result set using evidence from both logical retrieval engines to determine a more accurate probability of relevance.

In addition we have implemented a "Fusion Search" facility in the Cheshire II system that can be used to merge the result sets from multiple searches. These will typically be from different indexes and different elements of the collection which are then 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. We have exploited this facility in our retrieval processing for INEX (as discussed below). When the same documents, or document components, have been retrieved in differing searches, their final ranking value is based on combining the weights from each of the source sets. It should be noted, however that this final ranking value is not a probability but a combination of probabilistic weights and weighted Boolean values.

Relevance feedback has been implemented quite simply in the Cheshire II system, as probabilistic retrieval based on extraction of content-bearing elements (such as titles, subject headings, etc.) from items that have been seen and selected by a user. Because the INEX runs were done as a batch process, no relevance feedback was performed for them.

The following section describes the approach taken using the Cheshire II system to construct the INEX database and conduct to searches based on the INEX structured and content queries.

### 3. INEX APPROACH

Our approach in the INEX evaluation was to use all of the features of the cheshire system required to support the searches produced by the participant in the evaluation. This section will describe the indexing process and the search processing along with specific comments on particular searches and the special approaches taken in some cases. In this discussion we will describe some additional features of the Cheshire II system that were applied in processing the INEX queries.

#### 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. There is also support for extraction of proximity information as well (true proximity indexes where not used for the INEX evaluation). 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 would have a different set of stopwords from document titles or personal names.

Most of the indexes used in the INEX evaluation used KEYWORD 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

Name	Description	Contents
docno	Digital Object ID	//doi
pauthor	Author Names	//fm/au/snm //fm/au/fnm
title	Article Title	//fm/tig/at1
topic	Content Words	//fm/tig/at1 //abs //bdy //bibl/bb/at1 //app
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	//at1, //st
affil	Author Affiliations	//fm/aff
fno	IEEE Article ID	//fno

Table 1: Cheshire Article-Level Indexes for INEX

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 the INEX Evaluation and the document elements from which the contents of those indexes were extracted. Naturally the indexes created for the INEX collection were tailored to the needs of the retrieval task. Because it is simple to add a new index in the Cheshire system without re-indexing the entire collection, indexes were added incrementally to support all of the specified content elements from the 60 INEX topics (i.e., the <ce> tags from the topic documents). Many of the indexes were document-level indexes, but, given the combination of target elements and content elements specified in some of the topics, a set of defined components and indexes to those components were created also.

Name	Description	Contents
COMP_SECTION	Sections	//sec
COMP_BIB	Bib Entries	//bm/bib/bibl/bb
COMP_PARAS	Paragraphs	//ilrj //ip1 //ip2  //ip3 //ip4 //ip5  //item-none //p //p1  //p2 //p3 //tmath  //tf
COMP_FIG	Figures	//fig

Table 2: Cheshire Components for INEX

As noted above the Cheshire system permits parts of the document subtree to be treated as separated documents with their own separate indexes. Tables 2 & 3 describe the XML components created for the INEX Evaluation and the

component-level indexes that were created for them.

Table 2 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 2), and figure (<fig> ... </fig>) as individual documents that can be retrieved separately from the entire document.

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

**Table 3: Cheshire Component Indexes for INEX**  
†Includes all subelements of paragraph elements.

Table 3 describes the XML component indexes created for the components described in Table 2. These indexes make individual sections (COMP\_SECTION) of the INEX documents retrievable by their titles, or by any terms occurring in the section. Bibliographic references in the articles (COMP\_BIB) are made accessible by the author names, titles, and publication date of the individual bibliographic entry. Individual paragraphs (COMP\_PARAS) are searchable by any of the terms in the paragraph, and individual figures (COMP\_FIG) are indexed by their captions.

All of these indexes and components were used during Berkeley’s search evaluation runs of the 60 INEX topics. The runs and scripts used in the INEX evaluation are described in the next section.

## 3.2 The INEX Search Approach

Berkeley submitted three retrieval runs for the INEX Evaluation. This section will describe the general approach taken in creating the queries submitted against the INEX database and the scripts used to do the submission. Then the differences between the three runs will be examined, including the handling of some special cases where the default query processing provided by the scripts did not appear to provide effective results.

### 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 topic-id, query type, title (identifying content words (<cw>), content elements (<ce>), and target elements (<te>)), description, narrative, and keywords, concatenating multi-line elements and store for each topic.
  - (b) Duplicate British spellings in queries to include both British and U.S. spelling (e.g. “colour” becomes “colour color”).
  - (c) Based on the query type (CO or CAS):
    - i. For CO-type queries, construct 7 queries (run1 and run3) or 5 queries (run2) that include:
      - A. Boolean search of topic index for all terms from query title and keywords (run1 and run3).
      - B. Probabilistic search of topic index for all terms from query title and keywords (run1 and run3).
      - C. Probabilistic search of kwd index for all terms from query title and keywords (all runs).
      - D. Probabilistic search of abstract index for all terms from query title and keywords (all runs).
      - E. Probabilistic search of title index for all terms from query title and keywords (all runs).
      - F. Probabilistic search of alltitles index for all terms from query title and keywords (all runs).
      - G. Boolean search of alltitles index for all terms from query title (all runs).
    - ii. For CAS-type queries, construct all of the CO queries as in A-G above, but only for the keywords, then...
      - A. For each content element (<ce>) specified in the title of query construct both a probabilistic query and a boolean query of the index matching that content element, using the content words (<cw>) specified in the topic title for that content element.
    - iii. Construct extra or alternate queries for special cases (see below).
3. Submit queries and capture resultsets
  - (a) Each query constructed in the previous step is submitted to the system, and the resultsets with one or more matching documents are stored.
  - (b) All stored resultsets are combined using the resultset SORT/MERGE facility (discussed above), resulting in a single ranked list of the top-ranked 100 documents.

(c) The requested document elements (<te>) are extracted from the top-ranked 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)

### 3.2.2 "Fusion Search" and INEX Retrieval

As noted above, our INEX runs used the Cheshire "Fusion Search" facility in merging the result sets from multiple individual searches of different indexes. In the case of Berkeley's INEX runs, this 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.

The primary reason for this approach was largely 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. As described in the earlier section on Cheshire search, when the same documents, or document components, have been retrieved in differing searches, their final ranking value is based on combining the weights from each of the source resultsets. Therefore, a document that matches multiple searches will typically end up with a higher final rank than a document that matches fewer of the individual searches.

Thus, the goal in the search approach used in all of Berkeley's entries for INEX has been to try to achieve a good level of precision, without sacrificing too much recall.

### 3.2.3 Special Case handling

In reviewing the INEX topics, it was obvious that some of them would require special handling, because of unusual result requirements (e.g. topic #14 specifies that figures are to be retrieved along with paragraphs describing the figure). Others required special handling because of Boolean constraints on the requested results, unfortunately with inconsistent syntax for specifying those constraints (e.g. Topic #9 specifies that calendars are NOT to be retrieved by using "<cw> -calendar </cw> <ce> tig/at1 </ce>" while Topic #17 uses "<cw>not(W. Bruce Croft) </cw> <ce> fm/au </ce>" for the same type of constraint.

In these situations special handling of the queries to apply the appropriate constraints was carried out by the run scripts for the Berkeley runs. The topics that were handled in this way were numbers 02, 04, 07, 09, 12, 16, 17, 20, 26, 27 and 30. All other queries were handled without special processing.

## 4. CONCLUSION

The results of the full INEX evaluation have not yet been revealed, but from "eyeballing" the results of the Cheshire runs for individual topics, and from the experience of evaluating the pooled results from all of the participating systems for two of the topics, it appears that the approach taken in the Berkeley runs was fairly effective. Although precision of results is usually a poor guide to overall performance, it appears that in many cases (from an admittedly biased view of the results) that the results using our "Fusion Search"

approach have been quite good for many of the queries in the INEX test collection. We await the official analysis of the results to see if this belief is justified.

## 5. 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.

## 6. REFERENCES

- [1] 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.
- [2] 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.
- [3] 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, Denmark, June 21-24*, pages 198-210, New York, 1992. ACM.
- [4] 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.
- [5] R. R. Larson. Classification clustering, probabilistic information retrieval, and the online catalog. *Library Quarterly*, 61(2):133-173, 1991.
- [6] R. R. Larson. Distributed resource discovery: Using Z39.50 to build cross-domain information servers. In *JCDL '01*, pages 52-53. ACM, 2001.
- [7] R. R. Larson. A logistic regression approach to distributed ir. In *SIGIR 2002: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 11-15, 2002, Tampere, Finland*, pages 399-400. ACM, 2002.
- [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. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, Mass., 1994.
- [11] M. Porter and V. Galpin. Relevance feedback in a public access catalogue for a research library: Muscat at the scott polar research institute. *Program*, 22:1-20, 1988.
- [12] 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.
- [13] A. Z39.50-1995. *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification (ANSI/NISO Z39.50-1995)*. NISO, Bethesda, MD, 1995.

# Bayesian Networks and INEX

Benjamin Piwowarski  
LIP6  
8 rue du capitaine Scott  
75015 Paris, France  
Benjamin.Piwowarski@lip6.fr

Georges-Etienne Faure  
LIP6  
8 rue du capitaine Scott  
75015 Paris, France  
Georges-  
Etienne.Faure@poleia.lip6.fr

Patrick Gallinari  
LIP6  
8 rue du capitaine Scott  
75015 Paris, France  
Patrick.Gallinari@lip6.fr

## ABSTRACT

We present a bayesian framework for XML document retrieval. This framework allows us to consider content only and structure and content queries. We perform the retrieval task using inference in our network. Our model can adapt to a specific corpora through parameter learning.

## Categories and Subject Descriptors

H.3.3 [Information Storage and retrieval]: Information Research and Retrieval—*Retrieval models, Relevance feedback, Search process*; I.2.6 [Artificial Intelligence]: Learning—*Parameter learning*

## Keywords

Bayesian networks, INEX, XML, Focused retrieval, Structured retrieval

## 1. STRUCTURED DOCUMENTS AND INFORMATION RETRIEVAL

The goal of our model is to provide a new 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 *content only* (CO) queries and *content and structure* queries (CAS). We use a probabilistic model based on bayesian networks (BN), whose parameters are learnt so that the model may adapt to different corpora. For CO queries, we consider the task is a *focused retrieval*, first described in [5, 13].

The organization of this paper is as follow. We introduce bayesian network in section 2 and our model in section 3. We describe the three modes in which our model can be used: retrieval with CO and CAS queries and learning. Finally, in section 4 we describe related works.

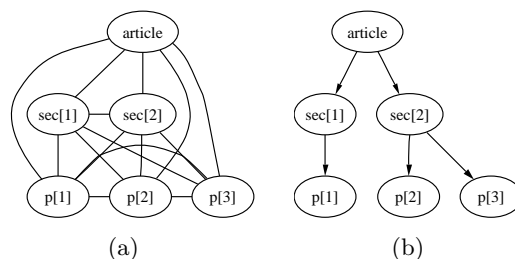
## 2. BAYESIAN NETWORKS FOR STRUCTURED DOCUMENT RETRIEVAL

Bayesian networks [9, 10, 15, 18] are a probabilistic framework where conditional independence relationships between random variables are exploited, in order to simplify or/and to model decision problems. For textual data, the seminal work of Turtle & Croft [20] raised interest in this framework, and since that, simple BN have been used for IR tasks (see section 2). A bayesian network is a directed acyclic graph (DAG) whose nodes represent variables of the problem and arcs (in)dependence relations between variables. Let  $\{x_i\}$ ,  $i = 1$  to  $N$ , denote the variables of a BN. Their joint probability is given by:

$$P(\{x_i\}) = \prod_i P(x_i | x_i \text{ parents})$$

where  $x_i$  parents denotes the parents of  $x_i$  in the DAG. Each random variable can take values from a set  $\mathcal{S}$ . For simplicity, in this section we consider binary variables. Note that strong simplifying assumptions on the structure of the BNs are needed for modeling textual data, since documents are represented in very large feature spaces.

Let us now present using a simple illustrative case how BN could be used to model and perform inference on structured documents. We will suppose that for retrieving documents,  $P(d|q)$  is used as the relevance score of document  $d$  with respect to query  $q$ .



**Figure 1: Two different modelizations of a same document. (a) All parts are dependent, (b) a simple BN inspired from the hierarchical document structure.**

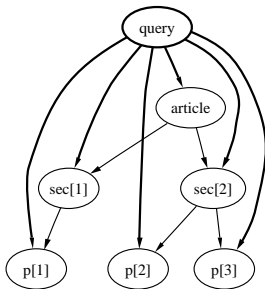
Consider the simple document of figure 1-a, composed of two sections and three paragraphs. A simple way to take into account the structure of  $d$  is to decompose the score

$P(d|q)$  as follows:

$$P(d|q) = \sum_{s_1, s_2, s_3, p_1, p_2, p_3} P(s_1, s_2, s_3, p_1, p_2, p_3|q)$$

Where  $s$  and  $p$  are random variables associated respectively to sections and paragraphs. Suppose now that each random variable (node) in this network can take two values: with respect to a given query,  $R$  means relevant and  $I$  irrelevant. To compute the joint probability values  $P(d, s_1, s_2, p_1, p_2, p_3)$ , we need  $2^6 - 1$  values for this simple document, and summations with up to  $2^5$  terms in order to compute  $P(d|q)$  or  $P(s_1|q)$ . This is clearly infeasible with documents with many structural entities.

In our model, BN are used to represent documents, one specific BN being associated to each document. Each node of a BN document model is a boolean variable which indicates whether or not the information associated to this node is relevant to the query. Different BN may be considered for modeling a document. Figure 1 shows a simple model for document 1-a, where the dependences go from parts to sub-parts so that section relevance depends on document relevance and paragraph relevance depends on section relevance.



**Figure 2: Retrieval network (model 1-b)**

For retrieval, when a new query  $Q$  has to be answered, we add the evidence (the query) to the network as shown in figure 2. Relevance thus depends upon both the query and the englobing structural part.

The relevance of a document or document part is computed using the conditional independence assumptions encoded in the BN. As an example, the probability of relevance of section 1 with the model 2 is given by:

$$P(s_1|q) = \sum_d P(d|q)P(s_1|d, q)$$

Where the summation is over the values random variable  $d$  (document) can take. With such a model, complexity drops from  $O(K^N)$  where  $N$  is the number of random variables to  $O(NK^{N_{max}})$  where  $N_{max}$ <sup>1</sup> is the maximal number of parents for a given random variable in the bayesian network and  $K$  is the number of values the random variables can take in our BN.

<sup>1</sup>For type (b), complexity is thus  $O(KN)$ .

### 3. MODEL

Our work is an attempt to develop a formal model for structured document access. Our model relies on bayesian networks instead of evidence theory in [11] or probabilistic datalog in [7] and thus provides an alternative approach to the problem. 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, whereas none of the other approaches relies on machine learning. This allows to rapidly adapt the model to different document collections and IR tasks.

The BN structure directly reflects the document hierarchy, i.e. we consider that each random variable is associated to a structural part within that hierarchy. The root of the BN is thus a variable "corpus", its childs 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.

Three different models were considered, they correspond to three different sets of values  $S$  for the BN variables:

**Model I** Relevant ( $R$ ) or not relevant ( $I$ ). This was the only model where each structural element relevance is independant of any other<sup>2</sup>. It is a simple model that was used in models II and III;

**Model II** Relevant ( $R$ ), too generic ( $G$ ), not relevant ( $I$ );

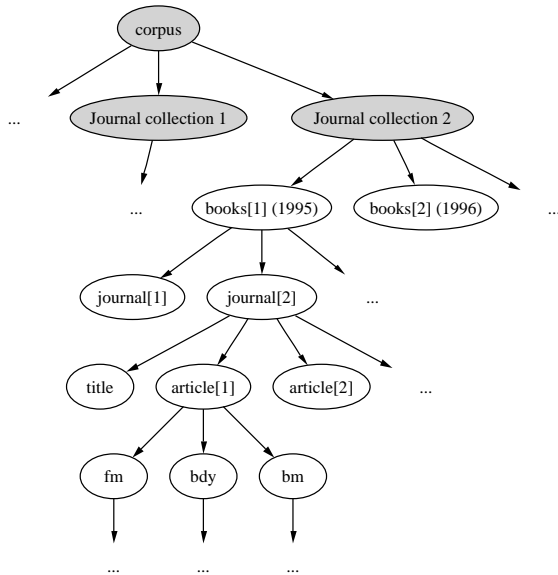
**Model III** Relevant ( $R$ ), too generic ( $G$ ), too specific( $S$ ) or not relevant ( $I$ )

This definition of relevance is related to several definitions of what should be information retrieval with free text queries on structured documents, as proposed by Chiamarella et al. [5] and Lalmas [13].

In order perform the inference steps in the BN, needed for retrieval or learning, we need to compute  $P(e|p, q)$  where  $e$  is the element,  $p$  its parent and  $q$  the query. For discrete variables, conditional probabilities in BNs are usually stored in tables. Here, these conditional probabilities should be computed for each  $Q$ , and they have to be learnt. For a given  $Q$ , we first compute a score  $F_{e,a,b}$  for each structural element  $e$ . In this instance of the model, this score will depend on the element  $e$  type (a tag in the XML document) and on the value  $a$  (among  $R, G, S, I$  according to model I, II or III) of the element  $e$  and the value  $b$  of its parent:

$$F_{e,a,b}(q) = \alpha_{e,a,b}F_{rel}^\alpha(e) + \beta_{e,a,b}F_{rel}^\beta(e) + \gamma_{e,a,b}F_{rel}^\gamma(e, a, b)$$

<sup>2</sup>It can't therefore truly be considered as a BN model



**Figure 3: The document collection: each structured document is located in a specific part of the hierarchically organized collection. Here, each document is a collection of journals, each journal contains structured articles (on the figure are indicated the tags for ...). The query  $q$  is added to this network while retrieving or learning as in figure 2. Below article[1], we have indicated some tags used in the INEX collection. fm, bdy and bm respectively hold for "front matter", "body" and "back matter", each being composed of sub-elements not represented on the figure.**

where  $F_{rel}^\diamond$  is the relevance of  $e$  content measured by a given flat retrieval model - in the experiments presented here, we have used a slightly modified version of OKAPI [21] as well as two other simple models. The first one gives a score that is the ratio between the number of occurrences of the query terms into the element and into its parent. The second one is similar but takes into account the size of the element. The peculiar form of  $F(e, a, b)$  has been chosen empirically and the two models have been chosen and tuned empirically.

This score is then used for computing the conditional probabilities  $P(e = a|p = b, q)$  using a softmax function that gives values between 0 and 1.

$$P(e = a|p = b, q) \propto \frac{1}{1 + e^{F_{e,a,b}(q)}}$$

For each possible value  $a$  of  $e$ , we then get a score which is interpreted as a probability.  $\alpha$  and  $\beta$  are to be learnt by the BN.  $\beta$  is the threshold while  $\alpha$  scales the score given by the model.

This model operates in two modes, *training* and *retrieval*, which we now describe.

### 3.1 Retrieval with CO queries

Answering CO queries was considered as *focused retrieval*. Focused retrieval consists in retrieving the most relevant structural elements in a document for a given query. Retrieval should focus on the smallest units that fulfill the query [5]. This unit should be the most relevant and should have a higher score than more generic or more specific units in the document.

When a new query  $Q$  has to be answered, we first compute  $F_{e,a,b}(q)$  score for each element  $e$  and values  $a$  and  $b$ . The tree structure of this BN allows to use a fast and simple inference algorithm. We compute the relevance  $P(e_i = R|q)$  for each element by marginalizing the conditional probability:

$$P(e_i = R|q) = \sum_{\{e_k\}_{k \neq i}} P(e_1, \dots, e_i, \dots, e_N|q)$$

where  $N$  is number of structural elements in the corpus and where the summation is taken over all combinations of values in the set  $\mathcal{S}$  for all variables except  $e_i$ . This formula factorizes according to the conditional independance structure of the network:

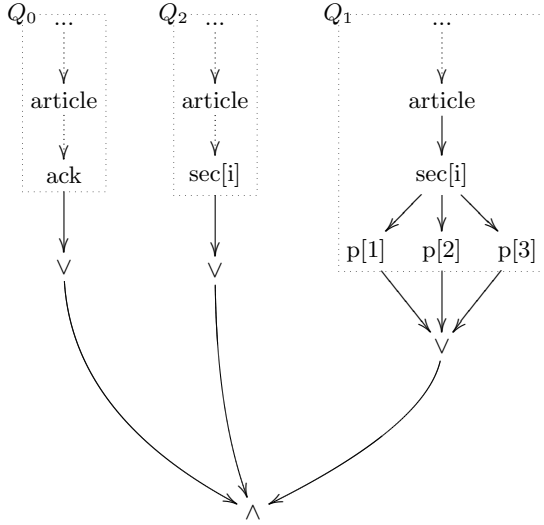
$$P(e_i = R|q) = \sum_{\{e_k\}_{k \neq i}} \prod_{j=1}^N P(e_j|e_j\text{'s parent}, q)$$

In the absence of evidence in the network (i.e. when there is no external indication on the relevance value of any node), this formula can be furthermore simplified: the summation will be only over all combinations for the set of  $e_i$  ancestors. Elements with highest values are then presented to the user.

### 3.2 Retrieval with CAS queries

For CAS retrieval, we extend our bayesian network to handle multiple subqueries and use on sub-network for each one. Those networks are then connected to form one big network that represents the whole CAS query.

In order to describe CAS query processing, we make use of an example (figure 4). Each CAS query is first decomposed into different subqueries (here  $Q_0$ ,  $Q_1$  and  $Q_2$ ). Each of these refers to a structural entity and an information need. Each information need is modeled by a BN constructed as for CO queries.



**Figure 4: An example of BN for a CAS query: retrieval of sections of article. The article must contain an acknowledgment relevant to query  $Q_0$ . The section relevant to query  $Q_1$ . The acknowledgment must be relevant to query  $Q_3$ . Here only the part network involved in the relevance scoring of one ack element is shown.**

Those BN are then connected for each target element in order to give this element a global score. Two different sub-query type were distinguished:

1. sub-queries that were relative to the target element ( $Q_0$ );
2. sub-queries that were relative to the article element ( $Q_1$  and  $Q_2$ ).

Type 2 sub-queries networks are always constructed after finding a target element.

Two different types of inference are used to connect bayesian networks between them, namely "or" ( $\vee$ ) and "and" ( $\wedge$ ) functions. For  $\wedge$  nodes we have:

$$P(\wedge = R | \text{parents}) = \begin{cases} 1 & \text{if one parent is } R \\ 0 & \text{otherwise} \end{cases}$$

and for  $\vee$  nodes we have:

$$P(\vee = R | \text{parents}) = \begin{cases} 0 & \text{if one parent is } \neq R \\ 1 & \text{otherwise} \end{cases}$$

### 3.3 Training

In order to fit a specific corpus, parameters are learnt from observations using the Estimation Maximization (EM) algorithm. An observation  $O^{(i)}$  is a query with its associated

relevance assessments (document/part is relevant or not relevant to the query). EM [6] optimizes the model parameters  $\Theta$  with respect to the likelihood  $\mathcal{L}$  of the observed data :

$$\mathcal{L}(O, \Theta) = \log P(O | \Theta)$$

where  $O = \{O^{(1)}, \dots, O^{(|O|)}\}$  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 in  $E^{(i)}$  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 more or less 200 assessments from CO queries that were obtained by taking only the browse keywords of CAS queries.

## 4. RELATED WORKS

In this section, we make a short review of previous works in IR related structured retrieval and emphasis on BN information retrieval systems.

One of the pioneer work on document structure and IR, is that of Wilkinson [22] who attempted to use the document division into sections of different types (abstract, purpose, title, misc., ...) in order to improve the performances of IR engines. For that he proposed several heuristics for weighting the relative importance of document parts and aggregating their contributions in the computation of the similarity score between a query and a document. He was then able to improve a baseline IR system.

A more recent and more principled approach is the one followed by Lalmas and co-workers [11, 12, 13, 14]. Their work is based on the theory of evidence which provides a formal framework for handling uncertain information and aggregating scores from different relevance scores. In this approach, when retrieving documents for a given query, evidence about documents is computed by aggregating evidence of sub-document elements.

Another important contribution is the HySpirit system developed by Fuhr and colleagues which was described in a series of papers, see e.g. [7]. Their model is based on a probabilistic version of datalog. When complex objects like structured documents are to be retrieved, they use rules modeling how a document part is accessible from another part. The more accessible this part is, the more it will influence the relevance of the other part.

A series of papers describing on-going research on different aspects of structured document storage and access, ranging from database problems to query languages and IR algorithms is available in the special issue of JASIST and in the proceedings of two SIGIR XML workshops[4, 1, 2].



Since Inquiry [3, 20], bayesian networks have proved to be a theoretically sounded IR model, which allows to reach state of the art performances and encompasses different classical IR models. The simple network presented by Croft, Callan and Turtle computes the probability that a query is satisfied by a document. More precisely, the probability that the document represents the query. This model has been derived and used for flat documents. Ribeiro and Muntz [19] and Indrawan et al. [8] proposed slightly different approaches also based on belief networks, with flat documents in minds. An extension of the Inquiry model, designed for incorporating structural and textual information has been recently proposed by Myaeng et al. [16]. In this approach, a document is represented by a tree. Each node of the tree represents a structural entity of this document (a chapter, a section, a paragraph and so on). This network is thus a tree representation of the internal structure of the document with the whole document as the root and the terms as leaves. The relevance information goes from the document node down to the term nodes. When a new query is processed by this model, the probability that each query term represents the document is computed. In order to obtain this probability, one has to compute the probability that a section represents well the document, then the probability that a term represents well this section and finally the probability that a query represents well this term. In order to keep computations feasible, the authors make several simplifying assumptions. Other approaches consider the use of structural queries (i.e. queries that specifies constraints on the document structure). Textual information in those models is usually boolean (term presence or absence). Such a well known approach is the Proximal Nodes model [17]. The main purpose of these models is to cope with structure in databases. Results here are boolean: a document matches or doesn't match the query.

## 5. CONCLUSION

We have described a new model for performing IR on structured documents. It is based on BN whose conditional probability functions are learned from the data via EM.

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. For example, we chose to discard textual information from the bayesian network (we use external models). A wiser choice would be to include terms within the bayesian network in order to give more expression power to our model. Other limitations are more technical and are related to the model speed.

Nevertheless some aspects of this model are interesting enough to continue investigating this model. Bayesian networks can handle different sources of information. Multimedia data can be integrated in our model by the mean of their relevance to a specific user need. Interactive navigation is also permitted. Our model is also able to learn its parameters from a training set. Since the relevance relationship between structural elements may change with the database, this seems to be an important feature.

## 6. REFERENCES

- [1] R. Baeza-Yates, D. Carmel, Y. Maarek, and A. Soffer, editors. *Journal of the American Society for Information Science and Technology (JASIST)*, number 53(6), Mar. 2002.
- [2] R. Baeza-Yates, N. Fuhr, and Y. S. Maarek, editors. *ACM SIGIR 2002 Workshop on XML*, Aug. 2002.
- [3] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY Retrieval System. In A. M. Tjoa and I. Ramos, editors, *Database and Expert Systems Applications, Proceedings of the International Conference*, pages 78–83, Valencia, Spain, 1992. Springer-Verlag.
- [4] D. Carmel, Y. Maarek, and A. Soffer, editors. *ACM SIGIR 2000 Workshop on XML*, July 2000.
- [5] Y. Chiaramella, P. Mulhem, and F. Fourel. A Model for Multimedia Information Retrieval. Technical report, IMAG, Grenoble, France, July 1996.
- [6] 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.
- [7] 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.
- [8] M. Indrawan, D. Ghazfan, and B. Srinivasan. Using Bayesian Networks as Retrieval Engines. In *ACIS 5th Australasian Conference on Information Systems*, pages 259–271, Melbourne, Australia, 1994.
- [9] F. V. Jensen. *An introduction to Bayesian Networks*. UCL Press, London, England, 1996.
- [10] P. Krause. *Learning Probabilistic Networks*. 1998.
- [11] 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.
- [12] M. Lalmas. Uniform representation of content and structure for structured document retrieval. Technical report, Queen Mary & Westfield College, University of London, London, England, 2000.
- [13] 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.
- [14] M. Lalmas, I. Ruthven, and M. Theophylactou. Structured document retrieval using Dempster-Shafer's Theory of Evidence: Implementation and evaluation. Technical report, University of Glasgow, UK, Aug. 1997.

- [15] K. P. Murphy. A Brief Introduction to Graphical Models and Bayesian Networks. web: <http://www.cs.berkeley.edu/~murphyk/Bayes/bayes.html>, Oct. 2000.
- [16] S. H. Myaeng, D.-H. Jang, M.-S. Kim, and Z.-C. Zhou. A Flexible Model for Retrieval of SGML documents. In W. B. Croft, A. Moffat, C. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 138–140, Melbourne, Australia, Aug. 1998. ACM Press, New York.
- [17] G. Navarro and R. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. *ACM TOIS*, 15(4):401–435, Oct. 1997.
- [18] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [19] B. A. N. Ribeiro and R. Muntz. A Belief Network Model for IR. In *Proceedings of the 19th ACM-SIGIR conference*, pages 253–260, 1996.
- [20] H. R. Turtle and W. B. Croft. Evaluation of an Inference Network-Based Retrieval Model. *ACM Transactions On Information Systems*, 9(3):187–222, 1991.
- [21] S. Walker and S. E. Robertson. Okapi/Keenbow at TREC-8. In E. M. Voorhees and D. K. Harman, editors, *NIST Special Publication 500-246: The Eighth Text REtrieval Conference (TREC-8)*, Gaithersburg, Maryland, USA, Nov. 1999.
- [22] R. Wilkinson. Effective retrieval of structured documents. In W. Croft and C. van Rijsbergen, editors, *Proceedings of the 17th Annual International Conference on Research and Development in Information Retrieval*, pages 311–317, Dublin, Ireland, July 1994. Springer-Verlag.

# Content-oriented XML retrieval with HyREX

Norbert Gövert    Mohammad Abolhassani  
University of Dortmund  
Germany

{goevert,abolhassani}@ls6.cs.uni-dortmund.de

Norbert Fuhr    Kai Großjohann  
University of Duisburg  
Germany

{fuhr,grossjohann}@informatik.uni-duisburg.de

## 1 Introduction

HyREX<sup>1</sup> is the hyper-media retrieval engine for XML [Abolhassani et al. 02]. Here we describe its implementation with respect to content-oriented XML retrieval within the INEX initiative for the evaluation of XML retrieval. HyREX implements the XIRQL query language which extends the XPath [Clark & DeRose 99] subset of XQuery [Chamberlin et al. 01] by functionality important in IR style applications.

For instance, IR research has shown that document term weighting as well as query term weighting are crucial concepts for effective information retrieval. These weights are used to compute a *retrieval status value* for a document component to be retrieved, thus resulting in a ranked list of components for a given query. In Section 2 we show how ranking is implemented in HyREX, such that it serves the need of content-oriented XML retrieval. Section 3 details the algorithm used to produce such a ranking of document components.

Given the logical structure inherent to XML documents, users of an XML retrieval engine want to be able to pose queries not only on content but also on the structure of the documents. As an extension of XPath, the XIRQL query language serves this purpose. Section 4 shows how this is used in order to process the INEX content-and-structure topics. In addition we give a brief overview on the concepts of data types and vague predicates which can lead to high precision searches, in combination with structural retrieval.

Section 5 displays very preliminary results in terms of effectiveness for content-oriented search. A conclusion and an outlook on further research is given in Section 6.

## 2 Weighting and ranking

Classical IR models treat documents as atomic units, whereas XML suggests a tree-like view on documents. Given an information need without structural constraints, the FERMI multimedia model for IR [Chiaramella et al. 96] suggests that a system should always retrieve those document components (elements) which answer the information need in the *most specific* way.

This retrieval strategy has been implemented in HyREX in order to process the INEX content-only topics. Here we

outline how classical weighting formulas (for plain document retrieval) can be generalised for structured document retrieval. Further details can be found in [Fuhr & Großjohann 01] and [Fuhr & Großjohann 02].

In analogy to the traditional plain documents, we first have to define the “atomic” units within structured documents. Such a definition serves two purposes:

- Given these units, we can apply e.g. some kind of tf·idf formula for term weighting.
- For relevance-oriented search, where no type of result element is specified, only these units can be returned as answers, whereas other elements are not considered as meaningful results.

We start from the observation that text is contained in the leaf nodes of the XML tree only. These leaves would be an obvious choice as atomic units. However, this structure may be too fine-grained (e.g. markup of each item in an enumeration list, or markup of a single word in order to emphasise it). A more appropriate solution is based on the concept of *index nodes* from the FERMI multimedia model: Given a hierarchic document structure, only nodes of specific types form the roots of index nodes. In the case of XML, this means that the database administrator has to specify the names of the elements that are to be treated as index nodes.

From the weighting point of view, index nodes should be disjoint, such that each term occurrence is considered only once. On the other hand, we should allow for retrieval of results of different granularity: For very specific queries, a single paragraph may contain the right answer, whereas more general questions could be answered best by returning a whole chapter of a book. Thus, nesting of index nodes should be possible. In order to combine these two views, we first start with the most specific index nodes. For the higher-level index nodes comprising other index nodes, only the text that is not contained within the other index nodes is indexed. Using this notion of index nodes an index node tree structure is induced onto the documents. As an example, assume that we have defined *section*, *chapter*, and *book* elements as index nodes; the corresponding disjoint text units are marked as dashed boxes in the example document tree in Figure 1.

So we have a method for computing term weights, and thus we can do relevance-based search. For this, we must be able to retrieve index nodes at all levels. The index weights

<sup>1</sup><http://ls6-www.cs.uni-dortmund.de/ir/projects/hyrex/>

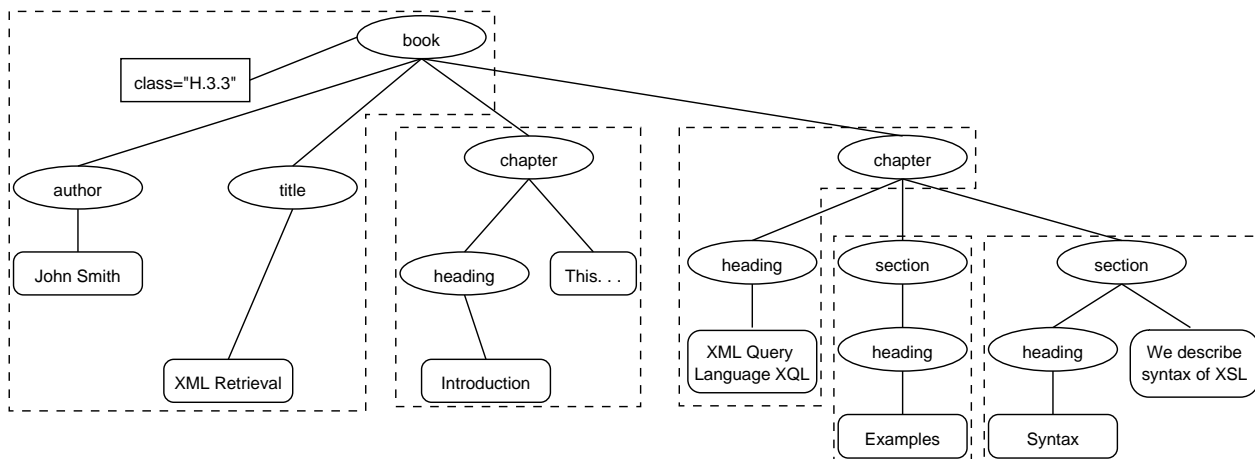


Figure 1: Example XML document tree with index nodes

of the most specific index nodes are given directly. For retrieval of the higher-level objects, we have to combine the weights of the different text units contained. For example, assume the following document structure, where we list the weighted terms instead of the original text:

```
<chapter> 0.3 XQL
  <section> 0.5 example </section>
  <section> 0.8 XQL 0.7 syntax </section>
</chapter>
```

A straightforward possibility would be the OR-combination of the different weights for a single term. However, searching for the term 'XQL' in this example would retrieve the whole chapter in the top rank, whereas the second section would be given a lower weight. It can be easily shown that this strategy always assigns the highest weight to the most general element. This result contradicts the structured document retrieval principle mentioned before. Thus, we adopt the concept of augmentation from [Fuhr et al. 98]. For this purpose, index term weights are down weighted (multiplied by an augmentation weight) when they are propagated upwards to the next index object. In our example, using an augmentation weight of 0.6, the retrieval weight of the chapter w.r.t. to the query 'XQL' would be  $0.3 + 0.6 \cdot 0.8 - 0.3 \cdot 0.6 \cdot 0.8 = 0.596$ , thus ranking the section ahead of the chapter.

### 3 Retrieval algorithm

For doing relevance-based searches, the XIRQL query language defines the respective `inode()` operator. However, in our INEX experiments we bypassed the XIRQL logical layer and directly accessed HyREX's physical layer in order to develop an efficient retrieval strategy for processing the INEX content-only topics.

The parallel algorithm which is described in the following, uses direct access to the inverted lists of the query terms in a given topic. As a prerequisite for the algorithm it is assumed that the inverted lists contain all the details necessary to describe a term occurrence for our index node retrieval approach:

**Index node identifier:** Each index node is assigned an ID during indexing.

**Index node description:** An index node is described by a path, beginning from the document root to the index node itself. The path contains the index node identifiers of all the index nodes of which borders are crossed, together with their respective augmentation weights.

**Weight:** This is the indexing weight for the given term within the index node represented.

Furthermore it is assumed that the entries in the inverted lists are ordered by document identifiers on the first level, and preordering of the index nodes (as they appear in the documents) on the second level.

Given that, the algorithm processes the occurrence descriptions within the various inverted lists until all of them are read. Due to the ordering in which the occurrence descriptions are read from the inverted lists, we reach that RSV computation for a given index node can be finished as early as possible. The `read_term` method observes the inverted lists beginning at their head and delivers the occurrence description from all of the inverted lists which is next according to the ordering described above:

**readterm() : inode\_id, inode\_path, augmentation, term, weight**

Method that implements a priority queue for the candidate set of occurrence descriptions to be processed next; these are read directly from the inverted lists of the query terms.

**inode\_path[l]** Array variable that lists the index node ids which make up the path from the document root towards the index node considered.

**augmentation[l]** array variable that lists the augmentation weights belonging to the index nodes in the `inode_path` described before

```
while ( inode_id, inode_path, augmentation,
        term, weight ) = readterm() do
  level = length inode_path
  ...
od
```

Figure 2 displays the inner part of the loop. First, it is checked whether there are index nodes, for which all information for computation of the RSV is available. Where

this is the case, the RSV is computed and the index node is pushed into the set of result candidates for the ranking. The following variables are needed for this:

**qterm\_weights[t]** Array variable which lists the query term weights.

**cumulated\_weights[l, t]** Matrix variable for cumulated index weights for *t* query terms at *l* index node levels.

**lastlevel** Level of the index node which has been processed in the previous iteration of the **while** loop.

**lastnodes[lastlevel]** Array variable representing the path of index nodes leading to the index node which has been processed in the previous iteration of the **while** loop.

**add\_result(inode\_id, weight)** Method to add an index node together with its respective RSV to the set of result candidates.

```

for j = 0 to min(level, lastlevel) do
  // check if some index nodes are finished
  if lastnodes[j] <> inodes[j] then
    // compute RSVs for finished index nodes
    for i = j to lastlevel do
      // apply linear retrieval function
      // (scalar value)
      rsv = cumulated_weights[i]
        * qterm_weights
      add_result(lastnode[i], rsv)
      // reset cumulated weights
      cumulated_weights[i] = 0
    od
  last // exit loop
fi
od
lastnodes = inodes
lastlevel = level
// propagate term weight of current term
// towards the root
for j = level downto 0 do
  cumulated_weights[j, term] =
    cumulated_weights[j, term] | weight
  weight = weight & augmentation[j]
od

```

Figure 2: Parallel algorithm for processing content-only topics

After all occurrence descriptions are processed, the result can be delivered to the user. If there is a maximum number *n* of result items to be retrieved (for INEX this was 100), the **add\_result** method can use a heap structure for selecting the *n* top ranking elements from the set of all index nodes processed.

The algorithm described here is quite efficient in terms of memory usage. By processing the inverted lists in parallel we achieve that retrieval status values for an index node once touched can be computed as early as possible. It follows that the number of accumulators for intermediary results is bounded by the maximum level an index node can have. An alternative algorithm which processes the inverted

lists sequentially would not be able to compute the final retrieval status values until all inverted lists are read. Thus it would have to allocate accumulators for all index nodes ever touched within the inverted lists of the query terms.

## 4 XIRQL: Processing content-and-structure topics

The XIRQL query language can be used to query on structured document collections using content *and* structural conditions. Given a fine-grained markup of XML documents, a mapping of the elements to specific data types (e.g. person names, dates, technical measurement values, names of geographic regions) can be done. For these data types special search predicates are provided, most of which are vague (e.g. phonetic similarity of names, approximate matching of dates, closeness of geographic locations). The concept of data types and vague search predicates [Fuhr 99] can thus be used to enhance the precision of a given information need.

```

<INEX-Topic topic-id="24" query-type="CAS">
  <Title>
    <te>article</te>
    <cw>Smith Jones</cw>
    <ce>au</ce>
    <cw>
      software engineering and
      process improvement
    </cw>
    <ce>bdy</ce>
  </Title>
  <Description>
    Find articles about software process
    improvement by the programming industry
    that are written by an author we believe
    is named either Smith or Jones.
  </Description>
  <Narrative>
    Only documents about software engineering
    written by Capers Jones are relevant.
  </Narrative>
  <Keywords>
    Smith Jones software engineering and
    process improvement programming
  </Keywords>
</INEX-Topic>

```

Figure 3: CAS topic 24 in XML format

These features have been used to process the INEX content-and-structure topics. For this, the CAS topics have been converted to XIRQL in a fully automatic way and then have been processed with HyREX. As an example, topic 24 is displayed in Figure 3. Figure 4 shows the result of the conversion of into XIRQL syntax. The topic is about retrieval of articles, thus the respective XPath expression `/article` starts the query. The further constraints are specified by two filters. In the first filtering section of the XIRQL query we represent the structural conditions within the title section of the original topic. For different elements specific search predicates are applied (phonetic similarity on author names and stemmed search for other query terms). The various conditions are combined using the weighted sum

operator. Another filter results from the query terms in the description and keywords section of topic 24. Again these are combined in a single weighted sum operator. The figures in front of the various conditions denote the query term weight.

```

/article[
  wsum(
    1, ./au//#PCDATA $soundex$ "Jones",
    1, ./au//#PCDATA $soundex$ "Smith",
    1, ./bdy//#PCDATA $stemen$ "engineering",
    1, ./bdy//#PCDATA $stemen$ "improvement",
    1, ./bdy//#PCDATA $stemen$ "process",
    1, ./bdy//#PCDATA $stemen$ "software"
  )
] [
  wsum(
    1, ./#PCDATA $stemen$ "Find",
    2, ./#PCDATA $stemen$ "Jones",
    2, ./#PCDATA $stemen$ "Smith",
    1, ./#PCDATA $stemen$ "articles",
    1, ./#PCDATA $stemen$ "author",
    1, ./#PCDATA $stemen$ "believe",
    1, ./#PCDATA $stemen$ "engineering",
    2, ./#PCDATA $stemen$ "improvement",
    1, ./#PCDATA $stemen$ "industry",
    1, ./#PCDATA $stemen$ "named",
    2, ./#PCDATA $stemen$ "process",
    2, ./#PCDATA $stemen$ "programming",
    2, ./#PCDATA $stemen$ "software",
    1, ./#PCDATA $stemen$ "written"
  )
]

```

Figure 4: CAS topic 24 in XIRQL syntax

## 5 Preliminary evaluation

A preliminary evaluation<sup>2</sup> has been conducted using the INEX assessments. Our focus has been on experimenting with different augmentation factors when doing the relevance-based retrieval described in Section 2. Figure 5 show the recall/precision curves for six different augmentation factors from 0.0 to 1.0, step 0.2. Recall/precision curves for the 13 content-only topics for which assessments are available, contributed to the curves.

The results do not favour any given augmentation factor. Instead the curves just give us the hint that small augmentation might be preferred if precision in the first ranks is important, while other factors are appropriate to enhance precision in the other ranks.

## 6 Conclusion

We have shown how HyREX has been utilised to process the INEX tasks. For dealing with the content-only topics an algorithm based on the notion of index nodes and

<sup>2</sup>Evaluation measures are to be discussed at the INEX workshop; we used a measure based on recall and precision to obtain our results (details on the implementation of this measure are part of these workshop notes).

Impact of augmentation factors

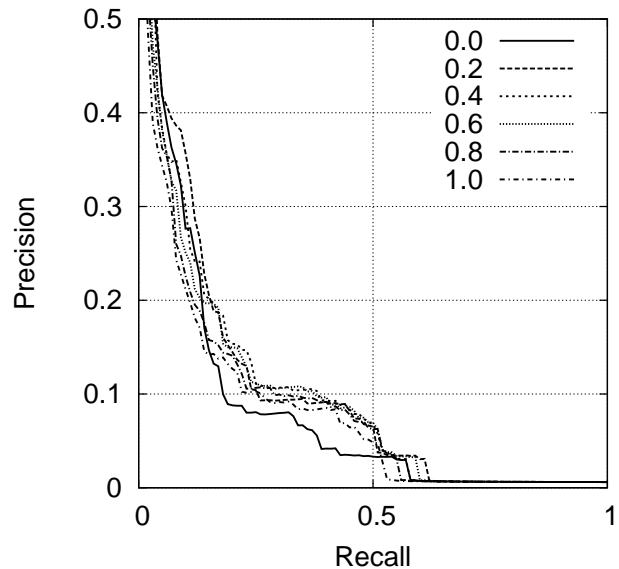


Figure 5: Recall/precision curves for different augmentation factors (content-only topics).

augmentation of index term weights has been developed. The XIRQL query language has been used to process the content-and-structure topics.

A preliminary evaluation could not yet show how index term weights should be augmented. Here alternative approaches for selecting appropriate augmentation factors are to be tested. In principle, augmentation factors may be different for each index node. A good compromise between these specific weights and a single global weight may be the definition of type-specific weights, i.e. depending on the name of the index node root element. The optimum choice between these possibilities will be subject to empirical investigations. Another way to derive augmentation factors could include information about the size of index nodes and the number of siblings. Finally, having relevance assessments for structured document retrieval now, one could even think of relevance feedback methods for estimating the augmentation factors. Further research will go into that direction.

Another issue is efficiency. In this article we describe an algorithm that uses all information from the inverted lists in order to compute RSVs. In order to become more efficient one can think of variants which terminate earlier. Here, the trade-off between efficiency and effectiveness has to be considered.

## References

Abolhassani, M.; Fuhr, N.; Gövert, N.; Großjohann, K. (2002). *HyREX: Hypermedia Retrieval Engine for XML*. Research report, University of Dortmund, Department of Computer Science, Dortmund, Germany. <http://ls6-www.cs.uni-dortmund.de/ir/projects/hyrex/>.

Chamberlin, D.; Florescu, D.; Robie, J.; Siméon, J.; Stefanescu, M. (2001). *XQuery: A Query Language for*

*XML*. Technical report, W3C. <http://www.w3.org/TR/xquery/>.

**Chiaramella, Y.; Mulhem, P.; Fourel, F.** (1996). *A Model for Multimedia Information Retrieval*. Technical report, FERMI ESPRIT BRA 8134, University of Glasgow. [http://www.dcs.gla.ac.uk/fermi/tech\\_reports/reports/fermi96-4.ps.gz](http://www.dcs.gla.ac.uk/fermi/tech_reports/reports/fermi96-4.ps.gz).

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

**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.** (2002). *XIRQL: An XML Query Language Based on Information Retrieval Concepts*. (submitted for publication).

**Fuhr, N.** (1999). Towards Data Abstraction in Networked Information Retrieval Systems. *Information Processing and Management* 35(2), pages 101–119.

**Fuhr, N.; Gövert, N.; Rölleke, T.** (1998). DOLORES: A System for Logic-Based Retrieval of Multimedia Objects. In: Croft et al. (ed.): *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 257–265. ACM, New York.

# Language Models and Structured Document Retrieval

Paul Ogilvie, Jamie Callan  
Carnegie Mellon University  
Pittsburgh, PA USA  
{pto,callan}@cs.cmu.edu

## ABSTRACT

We discuss possibilities for the use of language models in structured document retrieval. We use a tree-based generative language model for ranking documents and components. Nodes in the tree correspond to document components such as titles, sections, and paragraphs. At each node in the document tree, there is a language model. The language model for a leaf node is estimated directly from the text present in the document component associated with the node. Inner nodes in the tree are estimated using a linear interpolation among the children nodes. This paper also describes how some common structural queries would be satisfied within this model.

## 1. INTRODUCTION

With the growth of XML, there has been increasing interest in studying structured document retrieval. XML provides a standard for structured-document markup, and is increasingly being used. With the spread in the availability of structured documents, it is increasingly unclear whether the standard information retrieval algorithms are appropriate for retrieval on structured documents.

In this paper, we discuss how the generative language model approach to information retrieval could be extended to model and support queries on structured documents. We propose a tree-based language model to represent a structured document and its components. This structure is similar to many previous models for structured document retrieval [4][5][6][8][9][11], but differs in that language modeling provides some guidance in combining information from nodes in the tree and estimating term weights. The approach presented in this paper allows for structured queries and allows ranking of document components. It also matches some of our intuitions about coverage, which we discuss in Section 4.3.

The rest of the paper is structured as follows. Section 2 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 5 we briefly discuss parameter training. We discuss relationships to other approaches to structured document retrieval in Section 6, and Section 7 concludes the paper.

## 2. BACKGROUND IN LANGUAGE MODELS FOR DOCUMENT RETRIEVAL

Language modeling was developed by the speech recognition community as a means of estimating the probability of a word sequence (such as a sentence) given a sequence of phonemes recognized from an audio signal. The speech recognition community has developed sophisticated methods for estimating these probabilities. Their most important contributions to the use of language models in information retrieval are smoothing and methods for combining language models.

In information retrieval, documents and sometimes queries are represented using language models. These are typically unigram language models, which are much like bags-of-words, where word order is ignored. The unigram language model specifically estimates the probability of a word given a chunk of text. It is a “unigram” language model because it ignores word order. Document ranking is done one of two ways: by measuring how much a query language model diverges from document language models [10][12], or by estimating the probability that each document generated the query string [13][7][14][15].

### 2.1 Kullback-Leibler Divergence

The first method ranks by the negative of the Kullback-Leibler (KL) divergence of the query from each document [10]:

$$\begin{aligned} -KL(\theta_Q \parallel \theta_D) &= -\sum_w P(w \mid \theta_Q) \log \frac{P(w \mid \theta_Q)}{P(w \mid \theta_D)} \\ &\propto \sum_w P(w \mid \theta_Q) \log P(w \mid \theta_D) \end{aligned}$$

where  $\theta_D$  is the language model estimated from the document,  $\theta_Q$  is the language model estimated from the query, and  $P(w \mid \theta)$  estimates the probability of the word  $w$  given the language model  $\theta$ . The  $P(w \mid \theta_Q)$  within the log can be dropped in ranking because it is a constant with respect to the query. Documents where the query’s model diverges less from the document’s model are ranked higher.

### 2.2 The Generative Language Model

The generative method ranks documents by directly estimating the probability of the query using the documents’ language models [13][7][14][15]:

$$\begin{aligned} P(Q \mid \theta_D) &= \prod_{w \in (q_1, q_2, \dots, q_n)} P(w \mid \theta_D) \\ &\propto \sum_{w \in (q_1, q_2, \dots, q_n)} \log P(w \mid \theta_D) \end{aligned}$$

where  $Q = (q_1, q_2, \dots, q_n)$  is the query string. Documents more likely to have produced the query are ranked higher. 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 [12].

### 2.3 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:

$$P(w|\theta_T) = \frac{\text{count}(w;T)}{|T|}$$

where T is the observed text,  $\text{count}(w;T)$  is the number of times the word  $w$  occurs in T, and  $|T|$  is the length of the text. 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 serious 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.

## 2.4 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 both approaches to ranking documents, the document score is a sum of logarithms of the probability of a word given the document’s model. 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.

The smoothing technique most commonly used is linear interpolation. Linear interpolation is a simple approach to combining estimates from different language models:

$$P(w|\theta) = \sum_{i=1}^k \lambda_i P(w|\theta_i)$$

where  $k$  is the number of language models we are combining, and  $\lambda_i$  is the weight on the model  $\theta_i$ . To ensure that this is a valid probability distribution, we must place these constraints on the lambdas:

$$\sum_{i=1}^k \lambda_i = 1 \quad \text{and for } 1 \leq i \leq k, \lambda_i \geq 0$$

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. A specific form of linearly interpolating a document and a collection language model is called Bayesian smoothing using Dirichlet priors [15]. The document is modeled using maximum likelihood estimate.  $\theta_1$  is the document language model,  $\theta_2$  is the collection language model, and the linear interpolation parameters are:

$$\lambda_1 = \frac{|D|}{|D| + \mu} \quad \lambda_2 = \frac{\mu}{|D| + \mu}$$

where the parameter  $\mu$  is set according to the collection and is typically close to the average document length. This smoothing technique has been found effective for ad-hoc document retrieval on several collections [12] [14][15].

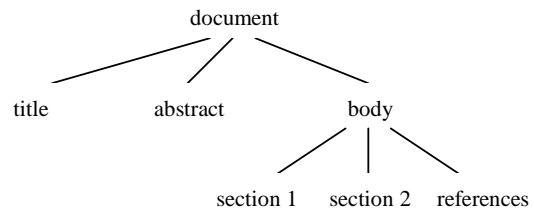
## 3. MODELING STRUCTURED DOCUMENTS

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.

The method we propose borrows from natural language processing. Probabilistic context free grammars (PCFGs) [1] are used to estimate the probability of parse trees of sentences. A PCFG is a context free grammar that has a probability associated with each rule. The probability of a specific parse tree is the product of the probabilities of all rules applied in creating the tree. The analogy we draw from PCFGs to structured documents is that the structure contained in the document can be represented as a context free grammar. The parse tree for the document is given by the structure. For example, if an XML schema specifies that a document is a title, abstract, and body text, then a corresponding rule in the grammar would be:

document  $\rightarrow$  title abstract body

Similarly, a partial tree for a document might look like:



Certain nodes, such as title and abstract, would be designated leaf nodes. In a traditional context-free grammar, a leaf node would be a word. In this model of documents, a leaf node would be a unit of text that does not have additional structure embedded in it. A language model for the leaf node would be estimated from the text.

An important distinction of the document tree language model from PCFGs used for parsing sentences is that we know the tree of the document. This is given directly by the document structure. Since we know the structure, it does not make sense to estimate the probability of a rule. Instead, we feel that we should view the rule as stating that the language model for the parent node consists of the language models of the children nodes.

The example rule given above states that a document language model consists of a title, an abstract, and a body language model. We next must specify how to combine the children language models. We suggest that linear interpolation is an appropriate method of combining the children language models.

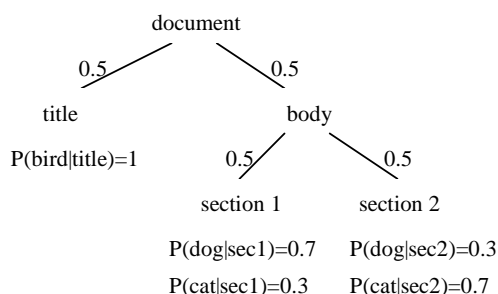
We believe that the optimal parameters for the linear interpolations in the rules depend on the task at hand and on the corpus. Training these parameters is a difficult problem which we will discuss more in Section 5.

This model as described assumes that all leaf nodes contain textual data only. However, it is common to have non-text data present in a document, such as dates, numbers, and pictures. As a language model is a probability distribution over a vocabulary, there really isn't anything stopping us from modeling non-text data in a language model. Appropriate smoothing methods for dates and numbers may be different than for text. For example, we may assume that a number may be normally distributed and taking the mean to be the observed value, using some reasonable estimate of variance. Images may also be modeled in this setting, though the approach may be more complex. Westerveld [13] proposes a method modeling images using a Gaussian Mixture Model, which he argues provides a framework for combining image retrieval with text-based language modeling. Combining the language models of mixed field types as prescribed by a rule may seem a little odd. Here, it may make sense to think of the interpolation weights as measures of relative importance. Additionally, we do not have to explicitly flatten the tree to a single language model; we can preserve the structure in our system and traverse the tree at query time.

The resulting tree for a given document would have a language model associated with every node and weight on the tree branches given by linear interpolation parameters specified in the rules. This provides a rich description of the document, which may be used for comparison to queries. The following section will discuss methods for querying.

## 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. We will sometimes refer to the following toy document model:



In this diagram, we specified the linear interpolation parameters on the edges. To keep things simple, we use equal parameters for the interpolation. We also specified the language models for the leaf nodes. 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 can use either traditional 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.

We would probably wish to remove document components in the ranking where a parent or child component is present higher in the ranking. This would prevent returning the same component multiple times. Other strategies for filtering the ranking have been proposed. An empirical study comparing techniques for filtering rankings is needed.

### 4.2 Structured Queries

Processing structure queries requires some adaptation of the language model retrieval approaches, as they do not currently allow for structural constraints. We will work with the generative language model here, as it is easier to adapt to structured queries. Following [7], Boolean style operators can be incorporated as follows:

- a AND b: Multiply  $P(a|\theta)$  and  $P(b|\theta)$ . This is the default operator in the generative language model.
- a OR b: Add  $P(a|\theta)$  and  $P(b|\theta)$ . This is interpreted as the probability that the language model  $\theta$  generated either a or b (or both).
- NOT a: Take  $1 - P(a|\theta)$ . This is the probability that the model  $\theta$  did not generate a.

Note that these Boolean operators enforce exact matches only when the MLE is used and no smoothing is applied to the leaf nodes. When smoothing the leaf nodes, the Boolean operators are soft matches.

There are many structural constraints that could be supported within this model, but we will only discuss how we would support a few constraints. A more thorough and complete description would be needed to implement a real system. Some constraints could be modeled as described below.

A simple constraint on which document components could be returned would be interpreted literally. For instance, if a query specifies the user wishes titles only to be returned, the system would only rank document titles.

The next constraint is of the form “return components of type  $x$  where it has component  $y$  that contains the query term  $w$ .” We first consider the constraint where  $y$  is a direct descendent of  $x$ . An example is “return *documents* where the *title* is contains the word *bird*.” This constraint can be viewed as measuring the probability that the document language model would generate the word *bird* from its title model. We observe that the linear interpolation weights can be viewed as probabilities. These correspond to the probability that the model was selected to produce a query term during generation. Formally, this constraint is given by  $P(w|y) \cdot P(y)$ , where  $P(y)$  is the linear interpolation weight for the document component  $y$ . For our example document and query, this would be

$$P(\text{bird}|\text{title}) \cdot P(\text{title}) = 1 \cdot 0.5 = 0.5.$$

Constraints that are nested more than one level deep can be modeled in a similar manner. However, instead of including only the linear interpolation weight for the constraint component, we include each weight in the path of the query constraint. Consider ranking the query “return *documents* where the *body's first section* contains the word *dog*” on our example document. This query would be ranked according to

$$\begin{aligned} & P(\text{dog}|\text{section 1}) \cdot P(\text{section 1}) \cdot P(\text{body}) \\ &= 0.7 \cdot 0.5 \cdot 0.5 \\ &= 0.175. \end{aligned}$$

We now have the mechanism to remove the constraint on which component to return in the previous examples. For the example query “return *components* where *section 1* contains the word *dog*.” A system would rank each component in the document that had section 1 component somewhere in its tree. A decision would need to be made whether a section 1 component could be returned for the query. In our example document, both the document and body components would be ranked (and possibly the section 1 component). For the document component, the score would be

$$P(\text{dog}|\text{section 1}) \cdot P(\text{section 1}) \cdot P(\text{body}),$$

and the body component would have a score of

$$P(\text{dog}|\text{section 1}) \cdot P(\text{section 1}).$$

The body component’s score will be greater than or equal to the document component’s score. It may seem odd to have a query of this form, but when combined with other query components, then the document may be preferred. For instance, the document component would be preferred over the body component for the query such “*bird* and *section 1* contains *dog*.”

A constraint that specifies a set of document components would treated as an OR operation. An example of this is “return *body components* where *any section* contains *dog*.” For the example document, this would be evaluated as

$$\begin{aligned} & P(\text{dog}|\text{section 1}) \cdot P(\text{section 1}) \\ &+ P(\text{dog}|\text{section 2}) \cdot P(\text{section 2}) \\ &= 0.7 \cdot 0.5 + 0.3 \cdot 0.5 \\ &= 0.5. \end{aligned}$$

This provides a sample of query operations that can be accommodated in the tree-based language model of documents. Any of the above operations can be combined into more complex queries, giving us the ability to represent and rank rather intricate queries.

### 4.3 Discussion

One nice benefit of the language modeling approach is that it implicitly deals with some of our intuitions about coverage. This is a result of how the language models estimate probabilities. To illustrate this, consider ranking the query Q = “dog cat” on our toy document. We will use the generative language model approach for this example. The probabilities for the leaf nodes are:

$$\begin{aligned} P(Q|\text{title}) &= 0 \\ P(Q|\text{section 1}) &= P(\text{dog}|\text{section 1}) \cdot P(\text{cat}|\text{section 1}) \\ &= 0.7 \cdot 0.3 \\ &= 0.21 \\ P(Q|\text{section 2}) &= P(\text{dog}|\text{section 2}) \cdot P(\text{cat}|\text{section 2}) \\ &= 0.3 \cdot 0.7 \\ &= 0.21 \end{aligned}$$

The language model for the body node is a linear interpolation of the section 1 and section 2 nodes. Similarly, the language model for the document node is a linear interpolation of the body and title nodes. These probabilities associated with these language models are:

$$\begin{aligned} P(\text{dog}|\text{body}) &= 0.5 \\ P(\text{cat}|\text{body}) &= 0.5 \\ P(\text{dog}|\text{document}) &= 0.25 \\ P(\text{cat}|\text{document}) &= 0.25 \\ P(\text{bird}|\text{document}) &= 0.5 \end{aligned}$$

Using these language models, we can now compute the probabilities that the body and the document generated the query:

$$\begin{aligned} P(Q|\text{body}) &= P(\text{dog}|\text{body}) \cdot P(\text{cat}|\text{body}) \\ &= 0.5 \cdot 0.5 \\ &= 0.25 \\ P(Q|\text{document}) &= P(\text{dog}|\text{document}) \cdot P(\text{cat}|\text{document}) \\ &= 0.25 \cdot 0.25 \\ &= 0.125 \end{aligned}$$

We see that the highest ranking document component for the query is the body component. This follows our intuition that the body component is probably better than either of the section components alone. Another favorable benefit is that the body component is ranked above the document component, which includes extra unrelated information.

Unfortunately, the model does not always behave as desired. Reconsider the query “dog cat.” If there is a document node containing only “dog cat”, then this leaf node will preferred over other nodes. This is undesirable, as there no context, resulting in an incoherent result. A way to deal with this issue is to rank by the probability of the document given the query. Using Bayes rule, this would allow us incorporate 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:

$$\begin{aligned} P(D|Q) &= P(Q|\theta_D) P(D) / P(Q) \\ &\propto P(Q|\theta_D) P(D) \end{aligned}$$

This would result in ranking by the probability of the document given the query, rather than the other way around. An example prior may be some function of the number of words subsumed by that node in the tree.

## 5. TRAINING THE MODEL

Training the linear interpolation parameters in the grammar is a difficult problem. For a task where there are often many relevant documents for a query, such as ad-hoc retrieval, an Expectation-Maximization approach may work well. Given a training set of queries and relevance judgments, an EM approach to training the parameters would be:

- 1) Initialize the linear interpolation parameters for each rule to random values. These values must satisfy the constraints for correct linear interpolation.
- 2) For each rule, update the parameters using:

$$\lambda_j^{[t+1]} = \frac{1}{z} \sum_{(Q,D) \in R} \sum_{w \in (q_1, q_2, \dots, q_n)} \frac{\lambda_j^{[t]} P(w|\theta_{j,D})}{\sum_{i=1}^k \lambda_i^{[t]} P(w|\theta_{i,D})}$$

where  $z$  is the normalizing constant that makes the new lambdas sum to one, the superscript  $t$  is used to denote values at the  $t^{\text{th}}$  iteration, and  $(Q,D) \in R$  represents the

pairs of queries and documents marked relevant in the training set. For learning linear interpolation parameters, the expectation and the maximization steps can be combined.

- 3) Repeat step 2 until some convergence criterion is met or for a fixed number of iterations.

This strategy will not work for all tasks. For some tasks, such as named-page or known-item finding, there is only one relevant document per query. Using EM to maximize the relevant documents for the queries runs the risk of also maximizing the probability of other non-relevant documents. While it is true that this is also a risk for ad-hoc retrieval, the effects of this on the evaluation measures are more pronounced for named-page and known-item finding. This is in part due to the choice of evaluation measures commonly used for named-page finding (such as mean-reciprocal rank). Mean-reciprocal rank is very sensitive to changes in rank near the top of the ranking. For these other tasks, it is desirable to have a learning technique that allows the system to directly optimize the evaluation function. Algorithms that may be easily adapted to this without the calculation of difficult gradients include genetic algorithms [16] and simulated annealing.

The parameter training is not an intractable task, nor may it be as difficult as we have suggested. Simple techniques like hand-tuning the parameters may work well, and it is unclear just how sensitive the model is to different parameters. We have had some success with hand-chosen linear interpolation coefficients for a simpler model [3].

## 6. RELATED WORK

Fuhr and Großjohann proposed XIRQL [4], which is an extension of XQL. They model queries as events which are represented in a Boolean algebra. The queries are converted into Boolean expressions in disjunctive normal form. The queries are evaluated on documents using the inclusion-exclusion formula. The event probabilities are estimated using weights derived from the text. These event probabilities are different from those in the language models, as they do not have to sum to one across all terms. Augmentation weights are used to allow inclusion of the weights from children nodes. These weights are in the range [0:1], which down-weight the children nodes' influence as the weights are propagated upward. Augmentation is a generalization of linear interpolation, where the constraint that the weights sum to one is relaxed. Their model does not assume independence among events, while the model presented here does assume independence of query terms.

Kazai et al [8][9] represent documents as graphs. The document structure is represented using a tree, but horizontal links are allowed among neighbor nodes in the tree. They model nodes in the tree using vectors of term weights. They call combining information in the tree aggregation, and use ordered weighted averaging (OWA) to combine node vectors. OWA is essentially the same as linear interpolation. While our model does not explicitly model links among neighbor nodes, this effect could be achieved by smoothing a node's language model with those of its neighbors.

Grabs and Schek [5] compute term vectors dynamically and use idf values based on the node type. Similarly, we smooth the nodes using information from the nodes of the same type. Their method of creating the term vectors dynamically may prove useful when implementing our approach. Structural constraints

in query terms are supported using augmentation weights similar to those used by Fuhr [4].

In [2], the authors present the ELIXER query language for XML document retrieval. They adapt XML-QL and WHIRL to allow for similarity matches on document components in the queries. The similarity scores are computed using the cosine similarity on tf-idf weighted vectors representing the query and the document component. Scores for multiple query components are combined by taking the product of the scores.

Myaeng et al [11] represent documents using Bayesian inference networks. The document components act as different document representations, and are combined in the network to produce a structure sensitive score for documents. Only document scores are computed; document components are not ranked.

Hatano et al [6] match compute tf-idf vectors for each node in the tree. They compute similarities of text components using cosine similarity, and they use the  $p$ -norm function to combine the similarities of the children nodes. The document frequencies are not element specific, while our language model smoothing is element specific.

## 7. CLOSING REMARKS

We proposed a tree-based language model for the modeling of structured documents. We described methods of querying structured documents using the model we described, and gave examples of how this is accomplished.

One benefit of the model include guidance from language modeling on how to the probabilities used in ranking. Another benefit is that the model captures some of our intuitions about selecting which components are most appropriate to return. The model also allows for including priors on components that can be used to model additional beliefs about coverage.

A disadvantage of the approach is that the linear interpolation parameters should be trained for best performance. These parameters may be corpus or task specific. However, we also present methods for training the parameters, such as EM or genetic algorithms.

The next steps for this work are to implement and test the model. Additionally, we will need to address concerns of efficiency and storage.

## 8. ACKNOWLEDGMENTS

We thank Yi Zhang and Victor Lavrenko for their insight and thoughts on structured documents and language modeling. 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.

## 9. REFERENCES

- [1] Allen, J. *Natural Language Understanding* (1995), 2<sup>nd</sup> edition, Benjamin/Cummings Publishing.
- [2] Chinenyanga, T.T. and N. Kushmerik. Expressive retrieval from XML documents. In *Proceedings of the 24<sup>th</sup> Annual International ACM SIGIR Conference on Research*

- and Development in Information Retrieval* (2001), ACM Press, 163-171.
- [3] Collins-Thompson, K., P. Ogilvie, Y. Zhang, and J. Callan. Information filtering, novelty detection, and named-page finding. In *Proceedings of the Eleventh Text Retrieval Conference, TREC 2002*, notebook version, 338-349.
- [4] Fuhr, N. and K. Großjohann. XIRQL: A query language for information retrieval in XML documents. In *Proceedings of the 24<sup>th</sup> Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 172-180.
- [5] Grabs, T. and H.J. Schek. Generating vector spaces on-the-fly for flexible XML retrieval. In *Proceedings of the 25<sup>th</sup> Annual International ACM SIGIR Workshop on XML Information Retrieval* (2002), ACM.
- [6] 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.
- [7] Hiemstra, D. *Using language models for information retrieval*, Ph.D. Thesis (2001), University of Twente.
- [8] Kazai, G., M. Lalmas, and T. Rölleke. A model for the representation and focused retrieval of structured documents based on fuzzy aggregation. In *The 8<sup>th</sup> Symposium on String Processing and Information Retrieval (SPIRE 2001)*, IEEE, 123-135.
- [9] Kazai, G., M. Lalmas, and T. Rölleke. Focussed Structured Document Retrieval. In *Proceedings of the 9<sup>th</sup> Symposium on String Processing and Information Retrieval (SPIRE 2002)*, Springer, 241-247.
- [10] Lafferty, J., and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24<sup>th</sup> Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 111-119.
- [11] Myaeng, S.H., D.H. Jang, M.S. Kim, and Z.C. Zhou. A flexible model for retrieval of SGML documents. In *Proceedings of the 21<sup>st</sup> Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1998), ACM Press, 138-145.
- [12] Ogilvie, P. and J. Callan. Experiments using the Lemur Toolkit. In *Proceedings of the Tenth Text Retrieval Conference, TREC 2001*, NIST Special publication 500-250 (2002), 103-108.
- [13] Ponte, J., and W.B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21<sup>st</sup> Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1998), ACM Press, 275-281.
- [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 24<sup>th</sup> Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 334-342.
- [16] Zhang, M., R. Song, C. Lin, L. Ma, Z. Jiang, Y. Jin, Y. Liu, L. Zhao, and S. Ma. THU at TREC 2002: novelty, web and filtering (draft). In *Proceedings of the Eleventh Text Retrieval Conference, TREC 2002*, notebook version, 29-42.

# The University of Amsterdam at INEX–2002

Maarten Marx, Jaap Kamps, Maarten de Rijke  
Language and Inference Technology  
ILLC, Universiteit van Amsterdam  
Nieuwe Achtergracht 166, 1018 WV Amsterdam  
The Netherlands  
{marx,kamps,mdr}@science.uva.nl  
<http://www.science.uva.nl/LIT/>

## ABSTRACT

This document describes the runs for the INEX–2002 task submitted by the Language and Inference Technology Group at the University of Amsterdam. Besides a description of our experiments some logical problems with the INEX format of the content and structure topics are discussed and an alternative is proposed.

## 1. INTRODUCTION

The aim of our official runs was to experiment with the effectiveness of different types of morphological normalization for structured corpora. Morphological normalization proved successful for plain text collections [5, 6]. The XML retrieval task departs from the strict Boolean query matching used in traditional database theory, allowing for various gradations of relevance. In particular, related words like morphological variants should share some of their relevance. In order to study the precise effect of morphological normalization, we created plain-word, stemmed, and ngrammed indexes that preserve the XML-structure of the original documents. This allows for both the content-only and content-and-structure topics to be evaluated against all three indexes.

All experiments were carried out with the FlexIR system developed at the University of Amsterdam [5], using the Lnu.ltc weighting scheme. Our indices follow the classical IR model: the documents (in this case the articles in the collection) are the atomic units.

For the content only topics, the XML structure of the documents was not used. For the content and structure topics, we used a two step strategy. We first treated the topic as a content only topic and selected the 1000 highest ranking documents. Then we directly processed (a morphological normalization of) these documents.

Our experiments are described in more detail in Sections 2

and 3. In the remaining sections we discuss encountered problems which are specific to IR with XML documents. In particular we discuss an alternative to the format in which the content and structure topics had to be specified.

## 2. SYSTEM DESCRIPTION

### 2.1 The INEX collection

The 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. The ten most frequent words in the collection are the following: data (169886), time (161415), system (149249), pp (137334), computer (119732), systems (116221), software (106552), fig (103141), set (99865), and ensp (99723).

### 2.2 The FlexIR information retrieval system

All submitted runs used FlexIR, an information retrieval system developed at the University of Amsterdam [5]. The main goal underlying FlexIR's design is to facilitate flexible experimentation with a wide variety of retrieval components and techniques. FlexIR is implemented in Perl; as it is built around the standard UNIX pipeline architecture, and supports many types of preprocessing, scoring, indexing, and retrieval tools, which proved to be a major asset for INEX task. The retrieval model underlying FlexIR is the standard vector space model. All our runs used the Lnu.ltc weighting scheme [1] to compute the similarity between a query and a document. For the experiments on which we report in this note, we fixed *slope* at 0.2; the pivot was set to the average number of unique words per document.

### 2.3 Morphological Normalization

Our aim was to study the effect of morphological normalization on the retrieval. To obtain a zero-knowledge language independent approach to morphological normalization, we implemented an ngram-based method in addition to a linguistically informed method. The ngram length was set to five. For each word we stored both the word itself and all possible ngrams that can be obtained from it without crossing word boundaries. For instance, topic 01 contained the phrase *description logics*. Using ngrams of length five, this becomes:

*description descr escri scrip cript ripti iptio ption  
logics logic ogics*

For the linguistically informed method with which we wanted to contrast the effect of the ngram-method we used Porter stemming [7]. In both approaches we removed words occurring on a stop list with 391 words. Figure 1 contains the original topic 31 and the stemmed version used as FlexIR input.

```
<INEX-Topic topic-id="31" query-type="C0" ct-no="003">
  <Title>
    <cw>computational biology</cw>
  </Title>
  <Description>
    Challenges that arise, and approaches being
    explored, in the interdisciplinary field of
    computational biology.
  </Description>
  ...
</INEX-Topic>
```

(a)

```
.i 31
comput biologi challeng aris approach
explor interdisziplinari field comput
biologi
```

(b)

Figure 1: Topic 31, original (a) and stemmed (b)

## 2.4 Combined Runs

We also wanted to experiment with combinations of (what we believed to be) different kinds of runs in an attempt to determine their impact on retrieval effectiveness. More specifically, we created a base run using the Porter stemmer and one in which we used ngrams in the manner described above. We then combined these two runs in the following manner. First, we normalized the retrieval status values (RSVs), since different runs may have radically different RSVs. For each run we reranked these values in [0, 1] using:

$$RSV'_i = \frac{RSV_i - \min_i}{\max_i - \min_i}$$

and assigned the value 0 to all documents not occurring in the top 1000. This is the Min\_Max\_Norm considered in [4]. Next, we assigned new weights to the documents using a linear interpolation factor  $\lambda$  representing the relative weight of a run:

$$RSV_{new} = \lambda \cdot RSV_1 + (1 - \lambda) \cdot RSV_2.$$

For  $\lambda = 0.5$  this is similar to the simple (but effective) combSUM function used by Fox and Shaw [3]. The interpolation factor  $\lambda$  was set to 0.6 for the ngram run. This higher weight for the ngram run was motivated by experiments on the CLEF [2] data sets.

## 2.5 Basic Architecture

In this subsection we describe how the runs were performed. The documents were processed as follows. The original xml-docs are standardized, tokenized and, if needed, stemmed or

ngrammed. Words in the documents are transformed into records and inverted. This results in an index for running the FlexIR retrieval program.

For the content-only topics, we follow the classical IR approach: Only the words in the title and description fields are selected. These words are, stopped and if needed, stemmed or ngrammed. For an example of this transformation after stemming, see Figure 1.

For the content-and-structure topics, we made two translations: First they are processed similar to the content-only topics: only the words in the title and description fields are selected; from the title field we only select the content of the `<cw>` field. Then the `<title>` field is processed to preserve the structural part of the query: the first line contains the topic number, the second line gives the xml-field that needs to be returned, the next line(s) give conditions for the document, consisting of a field name, and the words that are sought. For example, topic 01 with the following `<title>` field

```
<Title>
  <te>au</te>
  <cw>description logics</cw><ce>abs, kwd</ce>
</Title>
```

becomes after stemming

```
.i 01
article/fm/au
abs|kwd, descript logic
```

This should be read as: retrieve the content of `article/fm/au` if `article/fm/abs` or `article/fm/kwd` contains the words `descript` or `logic`.

For the content-only topics, we simply run the (naive, stemmed, or ngrammed) topics on the (naive, stemmed, or ngrammed) document index. The 100 documents with the highest FlexIR relevance assessment (RSV) are selected.

The runs for the content-and-structure topics are more complex: First we run the first translation of the (naive, stemmed, or ngrammed) topics on the (naive, stemmed, or ngrammed) document index, preselecting the most promising 1000 documents per topic. Our working hypothesis is that all relevant document are in this top 1000. For each topic, we create a special xml-file containing these top 1000 documents. On these so-called docpiles, we run an xml-parser based on Perl's `XML::Twig` that handles xpath expressions, and select the required field(s) from the documents that satisfy the conditions as specified in the second translation of the topic. In addition, we count the number of matching search words. The result is a `twig` file having the raw scored xml-elements in non-sorted order. Finally we select the (maximally) 100 xml-elements that have the highest number of matches, and sort them according to the original FlexIR relevance assessment.

### 3. RUNS

For the INEX 2002, we submitted three official runs:

**UAmsI02Stem** Stemmed index and topics, using Lnu.ltc weighting, and feedback.

For the ‘content and structure’ topics, the stemmed documents were used to create docpiles of the top 1000 documents of the stemmed run. The stemmed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

**UAmsI02NGram** Ngrammed index and topics, using Lnu.ltc weighting, and feedback. We used ngram-length 5, adding ngrams for words with length  $\geq 4$ , while also keeping the the originals words.

For the ‘content and structure’ topics, the stemmed documents were used to create docpiles of the top 1000 documents of the ngrammed run. The stemmed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

**UAmsI02NGiSt** Combined run using 0.6 Ngram, and 0.4 Stemmed.

For the ‘content and structure’ topics, the stemmed documents were used to create docpiles of the top 1000 documents of the combined ngram-stemmed run. The stemmed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

#### 3.1 Post submission runs for INEX

**UAmsI02Word** We create a naive, word-based run (still stopping, and lowercasing strings) by using a ngram-length of 100. Again, we use Lnu.ltc weighting, and feedback.

For the ‘content and structure’ topics, the stemmed documents were used to create docpiles of the top 1000 documents of the word-based run. The stemmed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

**UAmsI02NGramOnNGram** Ngrammed index and topics, using Lnu.ltc weighting, and feedback. We used ngram-length 5, adding ngrams for words with length  $\geq 4$ , while also keeping the the originals words.

For the ‘content and structure’ topics, the ngrammed documents were used to create docpiles of the top 1000 documents of the ngrammed run. The ngrammed structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

**UAmsI02WordOnWord** We create a naive, word-based run (still stopping, and lowercasing strings) by using a ngram-length of 100. Again, we use Lnu.ltc weighting, and feedback.

For the ‘content and structure’ topics, the word-based documents were used to create docpiles of the top 1000 documents of the word-based run. The word-based

structured topics were used to filter out the asked xml-elements from documents satisfying the asked conditions.

### 4. PROBLEMS WITH XML SYNTAX

The first few releases of the collection had a number of problems related to the XML syntax. In our approach we needed to do a morphological transform of the free text part of the documents but leave the XML structure intact. Because we used non forgiving XML parsers like TWIG it was very important to have and to keep valid XML documents. Here are some of the encountered problems:

- Deciding which broken tags to repair. Sometimes authors use tricks like `<tag>` to indicate that a tag should not be evaluated, but taken literally. This trick works with forgiving web browsers, but it is incorrect XML. The correct ‘trick’ would be `&lt;tag&gt;`, but what quite frequently the author actually meant to write was `<tag>`. If this is the case then translating `&lt;tag` to `&lt;tag&gt;` breaks XML validity because one is left with a `</tag>` somewhere that has no opening `<tag>` anymore.
- Dealing with embedded  $\TeX$  and  $\LaTeX$  proved to be quite difficult, because there were math formulas of the shape  $\$i\langle k\rangle 2\$\$$ , which leads an XML parser to believe that `<k>` is a tag. We decided to remove embedded mathematics using the XML-tags, i.e., `<tf>...</tf>` and `<math>...</math>`. This helps avoiding XML parser errors due to use of sequences, `<...>`, in math portions of the documents.
- Some documents have tags that contain newlines. For example, `so/2001/s5071` has tags like

```
<
  fig>
```

that span two lines. These tags are lost in our index, and gave parser errors. We rectify this by removing newlines that occur inside a tag.

- We used the following forgiving, yet not too liberal regular expression for XML tags:

```
<\/?w+s*(w+s*=\s*[\'\"]?[^\'\">]+[\'\"]?\s*)*\/?>
```

It works fine but matches things like `<p[\n]+>`, which could cause trouble when operating on the file line by line. The collection contains tag attribute values like `<li t="(3.9) ">` which the regular expression should catch.

### 5. TRANSFORMING INEX TOPICS INTO XPATH EXPRESSIONS

Our initial strategy was to use an XML query engine like Kweelt or Twig for the content and structure topics. For this reason we sought a way of automatically translating the INEX topic format into XPATH expressions. This turned out impossible for a number of reasons, one was that the



topic authors used operators for Boolean expressions and joins, but not in a uniform way, nor using uniform notation. Also, as evidenced in the discussion list, the meaning of “,” was not always clear.

It seems that these reasons can be overcome once a fixed topic language is given to the authors. We found though two deeper reasons why INEX topics cannot be transformed automatically into XPATH expressions. The first is that the use of (implicit) descendant axis in paths leads to problems of ambiguity and under-specification. The second is due to the fact that one cannot specify a connection between the `<te>` field (which is to be returned) and the `<cw>` and `<ce>` fields (which contain the conditions to be checked). From this we conclude that the INEX topic format is not a suitable question format and propose an alternative.

### Incomplete Paths

A very simple INEX topic is

```
<te> article </te>
<cw> logic </cw> <ce> kwd</ce>
```

This means *retrieve articles with “logic” as a keyword*. The equivalent XPATH expression (assuming all articles are in one file and all articles are contained in the XPATH expression `/article`) would be

```
/article[contains(./kwd,'logic')]
```

Another simple INEX topic is

```
<te> au </te>
<cw> logic </cw> <ce> kwd</ce>
```

This could be rephrased as *retrieve all authors of articles with “logic” among the keywords*. The corresponding XPATH expression is not possible to give without knowing the exact DTD and the paths to `au` and `kwd`. The naive solution

```
/article//au[
  contains(./ancestor::article//kwd,'logic')]
```

might be too general. It contains for instance

```
/article/bm/bib/bibl/bb/
  au[contains(../../../../fm/kwd,'logic')]
```

but these are authors whose work is cited from articles containing “logic” as a keyword.

With the DTD for the INEX collection, the following seems the correct translation. It retrieves authors of articles in the INEX databases with logic among the keywords.

```
/article/fm/au[contains(./kwd,'logic')]
```

But of course this is an interpretation of the original topic.

### Comma’s

Inside the `te`, `cw` and `ce` tags a comma separated list may occur. According to the instructions comma should be read as disjunction. This may lead to ambiguity, as the following example shows. Consider the topic *retrieve all author or editor names containing “John”*. The following XPATH expression just gives that

```
//author/name[contains(.,'John')] |
//editor/name[contains(.,'John')]
```

Note that “|” denotes the join (union) of the two sets of author names.

It seems impossible to formulate this as an INEX topic. The obvious

```
<te> //author/name, //editor/name</te>
<cw> John </cw> <ce> //author/name, //editor/name</ce>
```

can not be correct. How could we translate this to an XPATH expression while keeping the connection between what is being returned and what is being checked? The join has to be done after the two (independent) retrievals. We can not specify this topic in the INEX format.

### Our translation

We translated the INEX topics to XPATH expressions as follows:

- we replaced `<te>A,B,...</te>` (disjunctive search tags) by `<te>/article</te>`.
- we expanded element names in `te` and `ce` tags to unique paths (as in the example above). In case of a choice we used the whole topic to determine which path was meant.
- As the files contained at most one article (at least that was our assumption) we could work with the following translation. This has to be adjusted in case there are more articles in one file. Consider the following example INEX topic

```
<te> T </te>
<cw> K1,K2</cw><ce> CT1</ce>
<cw> L1</cw><ce> CT2,CT3</ce>
```

This topic would be translated into the XPATH expression

```
T[(contains(CT1,K1) or contains(CT1,K2))
  and (contains(CT2,L1) or contains(CT3,L1))].
```

### Proposal for INEX content-and-structure topic format

A union of xpath expressions in the last format is a good alternative to INEX topics. It provides more expressive power (because of the use of the context node in the contains expressions), and it is not ambiguous (because the implicit use

of descendant axis is forbidden; only complete paths which are valid under the DTD can be used).

We think that such a strict format yields better results, both in retrieval and in assessment. With the INEX format, topic translation was a creative process. Even the topic description was often not complete enough to yield a unique interpretation.

## 6. ACKNOWLEDGMENTS

We want to thank Willem van Hage for his technical support. Jaap Kamps was supported by the Netherlands Organization for Scientific Research (NWO), grant # 400-20-036. Maarten Marx received support from NWO grant 612.000.106. Maarten de Rijke was supported by grants from NWO, under project numbers 612-13-001, 365-20-005, 612.069.006, 612.000.106, 220-80-001, and 612.000.207.

## 7. REFERENCES

- [1] C. Buckley, A. Singhal, and M. Mitra. New retrieval approaches using SMART: TREC 4. In D. Harman, editor, *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*, pages 25–48. NIST Special Publication 500-236, 1995.
- [2] CLEF. Cross language evaluation forum, 2002. <http://www.clef-campaign.org/>.
- [3] E. Fox and J. Shaw. Combination of multiple searches. In *Proceedings TREC-2*, pages 243–252, 1994.
- [4] J. Lee. Combining multiple evidence from different relevant feedback networks. In *Database Systems for Advanced Applications*, pages 421–430, 1997.
- [5] C. Monz and M. de Rijke. Shallow morphological analysis in monolingual information retrieval for Dutch, German and Italian. In C. Peters, M. Braschler, J. Gonzalo, and M. Kluck, editors, *Evaluation of Cross-Language Information Retrieval Systems, CLEF 2001*, volume 2406 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2002.
- [6] C. Monz, J. Kamps, and M. de Rijke. The University of Amsterdam at CLEF-2002. In C. Peters, editor, *Results of the CLEF 2002 Cross-Language System Evaluation Campaign*, pages 73–84, 2002.
- [7] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

# A scalable architecture for XML retrieval

Gabriella Kazai      Thomas Rölleke  
Department Computer Science  
Queen Mary University London  
{gabs,thor}@dcs.qmul.ac.uk

## Abstract

Whereas in classical text collections, documents are considered as atomic units, we consider in XML collections elements in documents. This augmented view increases the number of potentially retrieved objects. A retrieved object can be a document, an element in a document, an aggregation of elements or of documents or the whole collection itself. The increase in the number of objects to be indexed and retrieved, and the generation of augmented representations leads for XML collections of comparably small size (several 100 MB) already to the necessity to apply strategies such as parallel and distributed indexing and retrieval, term, document and database pre-selection. We report in this paper on our approach for dealing with XML collections in general and with the INEX collection in particular using a scalable architecture.

## 1 Introduction

The widespread use of XML is one of the driving forces behind the prompt development of structured document retrieval systems. A growing number of approaches exists now that specifically deal with structured documents, such as XML. They can be classified into three main groups: database, IR and XML-specific approaches. Within IR, passage retrieval approaches long aimed to address the shortcomings of traditional IR, that it ignores document structure, by retrieving documents based on the most relevant syntactic or semantic passage(s) in documents. Data modeling approaches aim at developing data models for representing and querying with respect to the content and structure of documents. Aggregation-based approaches represent or estimate the relevance of document parts based on the aggregation of the representation or estimated relevance of their structurally related parts [4],

[2].

With the growth of the amount of available data, it is increasingly important to consider aspects of efficiency within structured document retrieval systems. Although computer hardware are becoming faster, data and approaches require scalable strategies to support the increasing requirements on data processing. [3].

In this paper we describe a retrieval system for structured documents that employs a scalable architecture for collection indexing and retrieval. The retrieval system is implemented using HySpirit, a software development kit that provides a descriptive approach for modeling complex information retrieval tasks such as hypermedia and knowledge retrieval by combining database models, probability theory, logic and object oriented concepts. HySpirit builds on a number of knowledge modelling languages including a probabilistic object oriented logic and a probabilistic relation algebra, and supports scalability in both the indexing and retrieval processes.

The paper is structured as follows. In section 2, we describe our general approach for increasing the efficiency in indexing and retrieval of XML objects. In sections 2.1 and 2.5 we report on the architecture of our distributed indexing of a collection and the strategy to “localise” the augmented representation of XML elements. In section 3, we relate the strategies to the INEX collection and experiments.

## 2 Scalability Approaches

In this section we describe several approaches that address the problem of efficient processing of large distributed collections for the task of structured document retrieval. We focus mainly on distributed and parallel collection indexing and retrieval and optimized augmentation for the representation of retrievable units.

## 2.1 Distributed and parallel processing

In a networked environment the documents of a text collection are distributed over several databases and processors, where a database and a processor itself can have a distributed and parallel architecture. Taking advantage of the distributed nature of the source data we can implement distributed and parallel indexing and retrieval mechanisms in order to increase the system's efficiency.

To demonstrate the distribution of an XML collection, consider the following collection structure:

```
<collection>
  <journal>
    <year>
      <volume>
        <article>
        </article>
      </volume>
    </year>
  </journal>
  <journal>
    <year>
      ...
    </year>
  </journal>
</collection>
```

A collection as such may be distributed according to a flat (linear) or complex (nested) architecture. In a complex architecture an XML element may contain sub-elements that are maintained in external databases, whereas in a flat structure only neighbour elements may be stored in different databases. Figures 1 and 2 illustrate the two architectures.

As an example for the complex case, a journal in the above XML collection may be stored in the following two databases.

```
(collection[1]/journal[1], database_1)
(collection[1]/journal[1]/year[1], database_2)
```

The relation associates pathnames with database identifiers, and shows that the journal, collection[1]/journal[1], is hosted in database\_1, whereas one of the year elements within the same journal, collection[1]/journal[1]/year[1], is located in database\_2.

From a practical point of view, we will often restrict ourselves to the flat distribution architecture, where the design of the distribution structure is simplified. The following is an example for a linear architecture.

```
(journal[1]/year[1], database_1)
(journal[1]/year[2], database_2)
(journal[2]/year[1], database_3)
...
```

Here we distribute the data with respect to the sibling year elements.

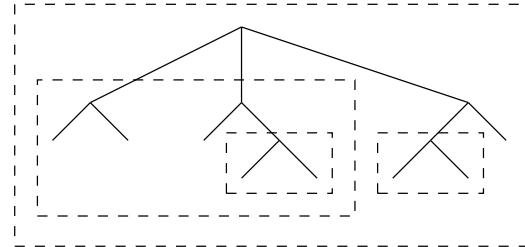


Figure 1: Complex (nested) distributed XML collection

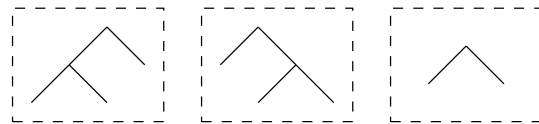


Figure 2: Flat (linear) distributed XML collection

Both architecture allows for distributed and parallel processing.

In the realm of structured document retrieval, the processing of a collection involves the tasks of indexing and retrieval. During indexing we derive representations for the XML elements of the collection. This representation includes both content and structure representation, which supports the content-oriented retrieval of XML documents where XML elements of varying granularity can be returned to a user. Ranking of the retrieved XML elements is based both on content-relevance and structure-coverage.

In a distributed environment, parallel indexing processes generate independent sub-collection representations, against which a user query is evaluated, in parallel, at retrieval time.

During indexing, for each of the databases, a space of document terms is computed. This termspace provides the basis for the local and global representation of the collection. The local representation refers to the representation of a given element within the collection (or sub-collection), whereas the global representation describes the collection (sub-collection) as a whole. In IR, these are often associated with the functions that

are used to estimate their respective probability weights within the representation, e.g. *tf* and *idf*.

There are two issues we address here:

1. The combination of the termspaces for obtaining an aggregated termspace of the sub-collections that are considered for a retrieval run. Here, we could consider a termspace that strictly conforms with the subset of databases or we could use a global termspace, i.e. a termspace that conforms with the set of all atomic elements.
2. The usage of the termspaces for the selection of databases during retrieval.

For the latter we can use a probabilistic representation of the sub-collections' termspaces, where the probability of a given term can be estimated using the standard *idf* calculations. Based on the individual termspaces of the different sub-collections we can then employ cost-based strategies to support the selection of sub-collections that are promising for retrieval.

For the first task we maintain an occurrence value of the terms within the sub-collections. This is needed to overcome the problem of information loss, which occurs when dealing with the probabilistic representations of the termspaces. This problem can be demonstrated by the following simple example. Given a collection of 10000 documents and a term that occurs in a 1000 of these documents, the probabilistic representation of this term in the collection's termspace may be given as  $\log(10000/1000)$ , when estimated using a standard *idf* function. Similarly in a collection of 10 documents where the same term occurs in 1 document, the term will be assigned the *idf* value of  $\log(10/1)$ . Aggregating these two collections based on the probabilistic (frequency-based) representation of their termspaces will obviously lead to incorrect weighting and hence retrieval results.

For example, the aggregated *idf*-value of a term "graphics" is computed as follows:

$$idf(graphics) = \log \frac{10000 + 10}{1000 + 1}$$

From the *idf*-values (global for all collections and for each sub-collection), we estimate so-called termspace probabilities. We base the estimation on the maximal *idf*-value ( $idf_{max}$ ).

$$P(graphics) := \frac{idf(graphics)}{idf_{max}}$$

Thus, terms that occur frequently in the collection have a low probability. The corresponding event in

the event space would be: "term graphics is informative/discriminative".

Aggregation based on the occurrence information, however, allows for transparent aggregation across heterogenous collections with different local representations. This ensures that the resulting global termspace is indifferent whether we aggregate based on the termspaces of the sub-collections or based on the termspaces of the elements.

With this approach we achieve a scalable distributed index that bears the same information and properties as an atomic index over the whole collection.

## 2.2 Database selection

For increasing the efficiency of a retrieval run, we perform a pre-selection of the promising databases based on a content-description of the database. Using a cost function (for example, based on the expected number of retrieved documents), we access the databases that allow us to stay within a given time and resource limit. This approach could be extended using [1], however, often, retrieval quality data are not available, and therefore we apply content-based and quantity-based measures.

## 2.3 Term and context selection

To further improve indexing and retrieval efficiency we reduce the number of terms and retrievable contexts. The removal stopwords is the classical strategy in IR, and in the same manner, we consider some contexts, for example those carrying only layout information as "stop-contexts". Although layout related tags should not be present in an XML source, often authors mix semantic and layout information in their documents. Other approaches that support a strategy to identify certain contexts as non-retrievable elements are methods that rely on defining a smallest retrievable unit.

Our approach here aims at identifying layout contexts from the information about the frequency of contexts within contexts. A possible criteria for identifying stop-contexts (non-semantic contexts) is to classify contexts after their occurrence within different super-context types and within the same actual context object.

In addition to stopword and stop-context removal, we skip the indexing of terms and contexts for reducing the use of resources. However, differently from stopwords and stop-contexts we risk here a decrease in retrieval quality in favour of efficiency. The challenge here is

to meet the best trade-off between quality and resource usage. Since several methods already exist that tackle this problem, including works on Latent Semantic Indexing, we do not address this issue in detail here.

We apply the term and context reduction strategies to both the document indexing terms and contexts, and the query terms and contexts.

Given this strategy, we view “intelligent” indexing as an indexing process that optimises the retrieval quality for a given amount of resources. (Index what is needed not what is possible.)

## 2.4 Parallel query processing

In a retrieval experiment, differently from ad-hoc retrieval, we deal with many queries. Under such circumstances we need to decide about the strategy for combining the query dimension and the database dimension. We distinguish two different batch retrieval strategies.

1. For each query, we retrieve from the set of databases.
2. For each database, we run the set of queries.

The design depends on the possibilities in parallelisation and the costs associated with a query evaluation or a database access.

Often, the access (in particular, the re-initialisation) to a database is very expensive. Therefore, it is often worthwhile to optimise with respect to database connectivity, which means, that we run the set of queries for a database. This is due to the assumption that a query switch is less expensive than a database switch. Also, a parallel access to queries is less of a bottleneck than a parallel access to a database.

In addition to the parallelisation with respect to databases and queries, each query can be parallelised by processing each query term independently.

## 2.5 Augmentation

With augmentation we refer to the feature in XML retrieval that the sub-contexts of a context constitute the content of a context.

Computing the augmented (aggregated) content of each retrievable context is an expensive computation, in particular since for very few terms, very few aggregated representations are actually retrieved (normally, far less documents of a collection are retrieved than

documents exist in the collection, and far more terms occur in the collection than do occur in queries).

In order to avoid this expensive use of resources, we try to restrict the aggregation to the query terms and the super-contexts of retrieved contexts. Of course, this means that the aggregation has to be performed during retrieval time. We refer to this strategy as “local” augmentation versus “global” augmentation where the latter would involve the augmentation of all retrievable contexts in the collection. Local augmentation puts emphasis on scalable strategies that reduce resource usage.

The augmentation of content is described in a deductive database approach. Let the relation “acc” contain the parent-child relationship in an XML collection. The transitive closure is formulated as follows:

```
acc(SuperContext, SubContext) :-
    acc(SuperContext, Context) &
    acc(Context, SubContext).
```

For evaluating the rule, a loop over a relational program is processed.

```
do {
    acc_previous = acc;
    acc =
        UNITE(
            acc,
            PROJECT[$1,$4](
                JOIN[$2=$1](acc, acc)));
} while (acc != acc_previous);
```

In each iteration, the “acc” relation is computed and compared with its previous instance. If the instance does not change anymore, then the transitive closure is completely computed.

This operation is very expensive for large data sources, even with the so-called semi-naive evaluation which considers only the increments of an iteration for computing the next increment.

Our strategy for cost reduction is to exploit the strict hierarchical nature of XML collections. The hierarchical nature allows for a stepwise computation of the transitive closure of the accessibility.

```
acc2 = PROJECT[$1,$4](
        JOIN[$2=$1](acc, acc))
acc3 = PROJECT[$1,$4](
        JOIN[$2=$1](acc2, acc))
acc4 = PROJECT[$1,$4](
        JOIN[$2=$1](acc3, acc))
...
```

The relation “acc2” contains all super-context sub-context relationships. Only those relationships are used for computing the relationships with distance three in “acc3”. Also, we restrict the augmentation to a maximal distance. For example, for a document structured in sections, subsections, paragraphs and sentences, with a distance of 5 the content of the sentences is aggregated to constitute the content of the document.

By exploiting the tree-structure of XML documents, we achieve smaller acc(i)-relations with increasing distance (i). Although the complexity of the join remains the same, the processing of the join becomes faster for high distance acc-relations, since the number of tuples has decreased. The repeated union and the comparison of acc-relations needed in the standard evaluation of a deductive formulation of augmentation can be omitted.

### 3 INEX Experiments

#### 3.1 Collection indexing

The collection of documents within INEX is made up of the IEEE Computer Society’s publications from 12 magazines and 6 transactions between 1995 and 2002, containing a total of 12107 articles. The full texts of the articles marked up in XML are stored as XML files in a directory structure that corresponds to the tree in Figure 3. The root of the directory structure is “INEX”, which contains 18 “journal” directories and 125 “year” sub-directories where the actual XML files are stored. Using the flat (linear) distribution architecture model, we can map the collection to a number of “journal/year” databases and one global augmented database. Given this structure, the task of indexing the whole collection can be broken down to the sub-tasks of indexing 125 sub-collections made up of articles given by their respective journal/year directory paths.

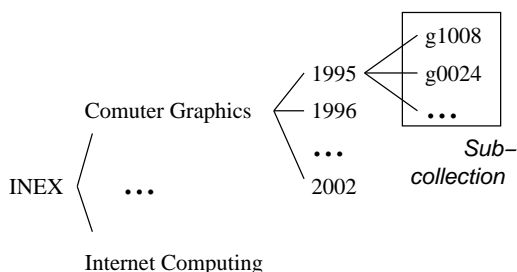


Figure 3: Collection tree

To index the INEX collection we used a probabilistic

aggregation-based approach, which views a document (or collection) as a tree and defines the representation of a document component (or sub-collection) within the document (collection) tree as the aggregated representation of its sub-components. The representation of a document includes aspects both regarding the representation of content and structure.

We used HySpirit as the platform on which we implemented both the indexing and retrieval functions. During the indexing process we derive a representation of the document’s structure via the transitive closure of the document tree, and the representation of the content for each leaf node within the document tree, which is then propagated up along the tree at retrieval time. The indexing process includes the conversion of the XML files into propositions in probabilistic object-oriented logic (POOL), then tuples in probabilistic relational algebra (PRA), which are then stored in relational databases. For example the XML fragment in Figure 4 is transformed to the POOL fragment shown in Figure 5 and then to the PRA code shown in Figure 6.

```
<article>
  <title>Graphics</title>
</article>
```

Figure 4: XML

```
article(article_1)
article_1[ title_1[graphics] ]
```

Figure 5: POOL

```
# tf(term, journal/year/article)
instance_of(article[1], article, cg/1997/g4042)
0.7 tf(graphics, cg/1997/g4042/article[1]/title[1])
0.5 acc(cg/1997/g4042/article[1],
cg/1997/g4042/article[1]/title[1])
```

Figure 6: PRA

In PRA, a document is represented using a number of relations, including *tf* (content) and *acc* (structure). The *tf* relation stores the occurrence of a term in a given context with a given probability, where the probability assigned to a term-context tuple can be estimated using standard *tf* calculations. For example, in Figure 6 the term “graphics” occurs in the title element of the article with the term frequency value of 0.7. The *acc* relation

represents the edges in the document tree. The probability assigned to an edge is the accessibility weight reflecting the strength of the structural relationship between a parent and child node. For example, the title element in Figure 6, which is accessible with 0.5 probability from its parent element.

The global termspace of the collection is computed using the augmentation method described in section 2, by aggregating the occurrence values of terms within the sub-collections. The probability of a term within the global termspace is then estimated using the maximal idf function. The following example shows the representation of the collection and a sub-collection related idf value of the term “graphics”.

```
0.2 idf(graphics, INEX)
0.5 idf(graphics, cg/1997)
```

As a result of our indexing process we created 125 distributed databases, where each database corresponds to a sub-collection (the articles within a year of a journal) of the INEX collection. Each sub-collection maintains a local termspace and structure information database. In addition a global database contains the global termspace and information on the collection’s structure.

During indexing we made use of distributed and parallel processing of the sub-collections and we employed our stopword and stop-context removal strategies.

### 3.2 Query processing and retrieval

We used HySpirit and an additional perl script to automatically parse and process the INEX topics, taking into account only the `title` and `keywords` components. The PRA representation of a query contains a representation for the query terms with associated term weights and a PRA program implementing a retrieval strategy. For content-only topics the retrieval strategy is based on a simple content-retrieval approach. For content-and-structure queries the retrieval strategy is a combination of content-retrieval functions and context-filters. We viewed the target elements of a query as a post-retrieval filtering task, which we did not implement.

Using HySpirit we evaluated a query against the distributed collection and applied our local augmentation strategy to the retrieval results. Within our approach content-retrieval based on the local and global representations (*tf* and *idf*) supports the relevance-oriented ranking and the augmentation process (*acc*) supports the coverage-oriented ranking of retrieved objects.

To implement parallel query processing we optimised with respect to database connectivity and for each database we evaluated the set of queries.

## 4 Conclusion

We identified in this paper an approach for scalable experiments with XML collections. The strategies (1) distributed and parallel indexing, (2) database selection, (3) term and retrievable context reduction and (4) distributed and parallel query processing are not specific to XML, whereas the fifth strategy regarding the augmentation is particular to the aggregated nature of XML collections.

In INEX we made most use of distributed and parallel indexing and retrieval. We also implemented the local augmentation strategy, simply because a global augmentation would have led to huge resource usage.

Our further steps will make greater use of database selection and “intelligent” reduction of indexing terms, both on the collection side and on the query side. In addition, we see potential in the parallel processing of query terms.

## References

- [1] N. Fuhr. Optimum database selection in networked ir. In J. Callan and N. Fuhr, editors, *NIR’96. Proceedings of the SIGIR’96 Workshop on Networked Information Retrieval*, 1996. <http://SunSite.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-7/>.
- [2] G. Kazai, M. Lalmas, and Thomas Rölleke. Focused structured document retrieval. In *Proceedings of String Processing and Information Retrieval (SPIRE), Lisbon, Portugal*, September 2001.
- [3] Berthier Ribeiro-Neto, Edleno S. Moura, Marden S. Neubert, and Nivio Ziviani. Efficient distributed algorithms to build inverted files. In SIGIR, editor, *SIGIR ’99, Proceedings of the 22nd International Conference on Research and Development in Information Retrieval*, pages 105–112, New York, 1999. ACM.
- [4] Thomas Rölleke, Mounia Lalmas, Gabriella Kazai, Ian Ruthven, and Stefan Quicker. The accessibility dimension for structured document retrieval. In *Proceedings of the BCS-IRSG, Glasgow*, March 2002.



# ETH Zürich at INEX: Flexible Information Retrieval from XML with PowerDB-XML

Torsten Grabs  
Database Research Group  
Institute of Information Systems  
ETH Zurich  
8092 Zurich, Switzerland  
grabs@inf.ethz.ch

Hans-Jörg Schek  
Database Research Group  
Institute of Information Systems  
ETH Zurich  
8092 Zurich, Switzerland  
schek@inf.ethz.ch

## ABSTRACT

When searching for relevant information in XML documents, users want to exploit the document structure when posing their queries. Therefore, queries over XML documents dynamically restrict the context of interest to arbitrary combinations of XML element types. State-of-the-art information retrieval (IR) however derives statistics such as document frequencies for the collection as a whole. With contexts of interest defined dynamically by user queries, this may lead to inconsistent rankings with XML documents that have heterogeneous content from different domains. To guarantee consistent retrieval, our XML engine PowerDB-XML derives the appropriate IR statistics that consistently reflect the scope of interest defined by the user query on-the-fly, i.e., at query runtime. To compute the dynamic IR statistics efficiently, our implementation relies on underlying basic indexes and statistics data. This paper reports on our experiences from participating in INEX, the INitiative for the Evaluation of XML retrieval.

## 1. INTRODUCTION

Since it became a recommendation of the World Wide Web Consortium (W3C) in 1998, the eXtended Markup Language (XML [10]) has been very successful as a format for data interchange. A common distinction regarding processing of documents marked up in XML is between *data-centric processing* and *document-centric processing*. Data-centric processing stands for processing of highly structured XML content with workloads using exact predicates similar to those of database systems. Document-centric processing in turn denotes processing of less rigidly structured content, and users compose queries with vague predicates and expect ranked results in the sense of information retrieval. Surprisingly, XML so far has mainly been used as a *data* format in data-centric settings, although its primary intention was as a *document* format for document-centric applications. Therefore, little support for information retrieval from XML documents has been available until recently.

INEX, the INitiative for the Evaluation of XML retrieval, is a joint international effort that addresses this issue. Next to promoting research on XML retrieval in general, it aims at developing appropriate testbeds and evaluation methods for information retrieval from XML [2]. Currently, the framework provided by the INEX organizers comprises a collection of about 12,000 XML documents with scientific publications of the IEEE Computer Society as well as a set of 60 topics with queries against the collection.

Important research questions that clearly need to be addressed for meaningful and flexible retrieval from XML are functionality of query languages and suitability of retrieval models. With respect to query languages, users want to exploit the structure of XML documents to perform fine-grained and flexible retrieval. This is in contrast to conventional IR where the retrieval granularity usually is restricted to predefined entities such as 'title', 'abstract', or 'fulltext'. With XML instead, users may want to pose queries on arbitrary combinations of XML element types. Hence, more flexible mechanisms to define the context of interest are required.

Regarding retrieval models, information retrieval systems should exploit the XML document structure for better relevance ranking. Moreover, conventional information retrieval systems so far have made the assumption that all the contents of a collection is from the same domain. With XML documents however, even a single document may have heterogeneous content from different domains in different parts of the document. This may lead to inconsistent rankings with weighted retrieval models if term weights differ between domains, as the following example illustrates.

**Example 1:** Figure 1 shows an exemplary document from the INEX document collection (left) and its representation as a tree-structure (right). Consider a user who is interested in database transaction processing. Assume that he composes a query that searches for the most specific XML element in the document collection using the keyword 'transaction'. Obviously, the paragraph element `/article/body/sec/p` in the example document could be a promising candidate since it comprises the term 'transaction'. But, the journal title element `/article/fm/ti` also contains the term 'transaction'. Nevertheless, it is intuitively less relevant than the section paragraph since many documents have a journal title that starts with 'IEEE Transactions on ...'. Consequently, the user expects the section paragraph to be ranked higher than the journal title element. However, conventional approaches to information retrieval derive term weights for the collection as a whole and may therefore rank the journal title higher than the paragraph. ◊

Our current work at ETH Zurich aims at addressing the problem of inconsistent rankings for flexible retrieval from XML. We are currently building PowerDB-XML, an XML engine that supports both data-centric and document-centric processing of XML in an effective and efficient way on a scalable platform implemented on top of a cluster of databases. On the one hand, our approach relies on extending state-of-the-art XML query languages such as W3C XPath with document-centric functionality. Section 2 reports on

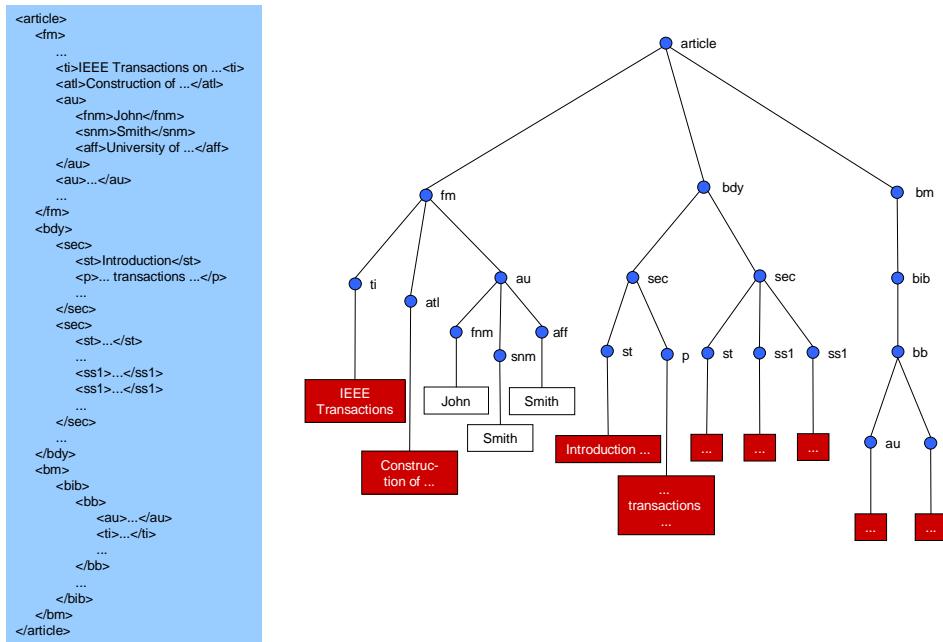


Figure 1: Sketch of an XML document from the INEX collection

these current efforts. On the other hand, relevance ranking with PowerDB-XML derives term weights for retrieval from XML at a much finer granularity than conventional retrieval. This prevents from inconsistent rankings that would occur with conventional IR term weighting, as Example 1 has illustrated. We discuss our approach that we currently evaluate within the INEX initiative in Section 3. Section 4 explains our implementation of IR functionality with PowerDB-XML. Section 5 discusses related work, and Section 6 concludes.

## 2. EXTENDING XML QUERY LANGUAGES WITH IR FUNCTIONALITY

Previous efforts to come up with query languages for XML were mainly driven by the database community. There, the focus has been on functionality for data-centric processing. This has led to the development of query languages such as XPath and XQuery [11, 12]. Recently, extensions of these languages have been proposed in order to cover document-centric processing as well. XIRQL for instance extends XPath with functionality for ranked retrieval, relevance-oriented search, vague predicates and semantic relativism [4, 7]. PowerDB-XML takes over much of these ideas. We have also decided in favor of XPath because it has found wide acceptance in particular in practical systems after it became a recommendation of the W3C in 1999. A further reason is that XPath is part of other ongoing standardization efforts of the W3C such as XQuery – the prospective standard query language for XML. Furthermore, XPath comes with an intuitive and easy-to-understand syntax.

However, XPath lacks of the functionality to pose IR-queries to search for relevant content which is needed with document-centric processing. The only XPath functionality available in this respect is the function *contains()*. It allows to check for occurrences of a given character string in XML content. Clearly, this does not suffice to cover the requirements for meaningful and flexible retrieval from XML documents in the sense of information retrieval. For instance, weighting and relevance ranking is not available with

XPath. Hence, our approach is to extend XPath with information retrieval primitives.

XPath already provides data-centric constructs for selection and projection by structure constraints. With XPath, structure constraints are formulated as path expressions that select those nodes of the graph representation of a document that match the expression. Path expressions have the syntax */step/step/.../step*. Starting at the root node, each *step* moves the current context through the XML element hierarchy. Each *step* has the form *AxisSpec::NodeTest[Predicate]* and its evaluation depends on the current context. Different axis specifications *AxisSpec* allow to navigate through the document. For instance, the *child* axis and the *parent* axis denote the children nodes and the parent node of the current context, respectively. With a *NodeTest* in turn, only those nodes qualify for a step that are of a given type. For instance, the XPath step *descendant::firstname* returns only those descendants of the context node that are *firstname* elements. The joker sign *\** serves as a wildcard for node tests: *descendant::\** yields all descendants of the context node. *Predicates* can pose further constraints on the content of nodes. The usual comparison operators *<*, *≤*, *=*, *...* and Boolean operators *AND* and *OR* are available with predicates. Take the XPath expression *//descendant::auction[price < 20]* as an example. It returns all auctions whose price is less than 20.

As the previous example illustrates, XPath already covers important requirements for data-centric processing, namely projection and selection. Therefore, XPath has been adopted widely as a query language for data-centric processing. However, XPath does not cover document-centric processing since it is not possible to formulate IR-style queries. Our approach thus is to take over the data-centric functionality of XPath and to extend it with the functionality that is required for document-centric processing, namely flexible and meaningful ranked retrieval on XML content.

To do so, our path expression matching language called *XPathIR*

overloads the XPath function *contains(.)* to introduce information retrieval functionality. With XPathIR, the following signatures are available:

- The signature *contains(expr, string) → boolean* corresponds to the standard one from the original XPath recommendation. The function returns *true* if the textual content of the match to *expr* contains the string given by the second parameter.
- *contains(expr, query, irmodel, rsv, hits) → boolean* is an XPathIR-specific extension of the XPath Recommendation. It returns *true* for an element or attribute that matches *expr* only if its content has a retrieval status value of at least *rsv* and is among the top *hits* hits under the query text *query* when using the information retrieval model *irmodel*.

**Example 2:** Consider again the XML document in Figure 1 and the XPathIR-query `/article[contains(/bdy/sec, 'database transaction processing', TFIDF, 0.3, 10)]`. The query searches for articles where a `sec` element has an *rsv* of at least 0.3 and is among the top 10 hits under the query text 'database transaction processing' using TFIDF vector space retrieval. ◊

With the INEX initiative, retrieval functionality for XML has to cover both so-called *content-only queries* (CO queries for short) and *content-and-structure queries* (CAS queries for short) [2]. Content-and-structure queries refer to the document structure in order to restrict the context of IR search to those nodes that match a structural pattern provided with the query. The result of such a query is a ranking of XML elements that match the structural constraints of the query. Elements are ranked higher the more relevant they probably are to the query text. Content-only queries in turn do not have constraints with respect to document structure. Similar to conventional IR, they only comprise off a query text or a set of keywords. However, the result of such a query is a ranking of XML elements such that the elements are ranked higher the more specific and the more relevant they are. This is in contrast to conventional IR where the granularity of the resulting hits is the same for all hits returned.

The current workload of the INEX testbed consists of 30 CO topics and 30 CAS topics. Each topic comes with a topic title, a description, a narrative, and a set of keywords. With CAS topics, the topic title specifies the structural patterns. With both CO topics and CAS topics, the topic title also specifies the query text. We have taken the information from the topic title to transform the topics to XPathIR expressions. The following example illustrates this for a CO topic and two CAS topics taken from the INEX workload.

**Example 3:** INEX topic 31 is a content-only query with the query text 'computational biology'. We transform the topic to the XPathIR expression `//*[contains(., 'computational biology', TFIDF, 0.0, 100)]` that returns the top 100 XML elements that are most specific and most relevant to the query text using vector space TFIDF ranking. INEX topic 02 in turn is a content-and-structure query. Its topic title is '`<cw> research funded america </cw> <ce> ack </ce>`'. The contents of the `cw` element is the query text and the `ce` element specifies the structural pattern. We have mapped this topic to the XPathIR query `//ack[contains(., 'research funded america', TFIDF, 0.0, 100)]`. Using again TFIDF ranking, it returns those `ack` elements that are most relevant to the query text. Topic 01 with the title '`<cw>description logics</cw><ce>abs,`

`kwd</ce>`' in turn maps to the XPathIR expression `(//abs|//kwd)[contains(., 'description logics', TFIDF, 0.0, 100)]`. ◊

The previous example illustrates three basic retrieval operations that are needed for flexible retrieval from XML, namely single-category retrieval, multi-category retrieval, and nested retrieval. Topic 31 represents *nested retrieval* since the query is evaluated against all elements and their sub-elements. Topic 02 in turn is an example of *single-category retrieval*, since it only considers elements from the `ack` element type. Finally, topic 01 stands for a *multi-category query* since the context of interest of this query is composed from the union of the instances of the two element types 'abstract' and 'keywords' (`abs` and `kwd`). It is important to note, that with weighted retrieval models a multi-category query has different semantics than a sequence of single-category queries. For instance, the XPathIR expression for topic 01 given in Example 3 is different from expression `//abs[contains(., 'description logics', TFIDF, 0.0, 100)]|//kwd[contains(., 'description logics', TFIDF, 0.0, 100)]`. The following section explains this in more detail.

### 3. RELEVANCE RANKING FOR WEIGHTED RETRIEVAL FROM XML

Following the approach outlined in the previous section, we have mapped all INEX topics to XPathIR expressions. To implement query processing for these expressions, PowerDB-XML relies on our previous work on single-category retrieval, multi-category retrieval, and nested retrieval [6]. In the following, we briefly review the approach and explain how we have deployed it to the INEX framework.

Flexible retrieval for XML first requires to identify the basic element types of an XML collection that contain textual content. We denote them as *basic indexing nodes*. There are several alternatives how to derive the basic indexing nodes from an XML collection:

- The decision can be taken completely automatically such that each distinct element type at the leaf level with textual content is treated as a separate indexing node.
- An alternative is that the user or an administrator decides how to assign element types to basic indexing nodes.

These approaches can further rely on an ontology that, for instance, suggests to group element types 'title' and 'abstract' into the same basic indexing node. With the INEX framework, we have worked with two alternatives. The first alternative applies basic indexing nodes defined by an administrator. The second approach in turn relies on basic indexing nodes that have been derived automatically. With the latter approach, different basic indexing nodes have been generated for different XML element types. Figure 2 illustrates this for a part of the element type hierarchy of the INEX document collection (cf. Figure 1). IR pre-processing such as term extraction, Porter stemming, and stopword elimination on the textual content of the instances of the element type yields the information that the basic indexing node materializes. For our experiments with the INEX framework, we have generated basic indexing nodes with inverted lists (*IL*) and statistics (*STAT*) for vector space retrieval. Building on the notion of basic indexing nodes, we describe in the following how PowerDB-XML implements flexible and consistent retrieval on the INEX document collection using single-category retrieval, multi-category retrieval and nested retrieval.

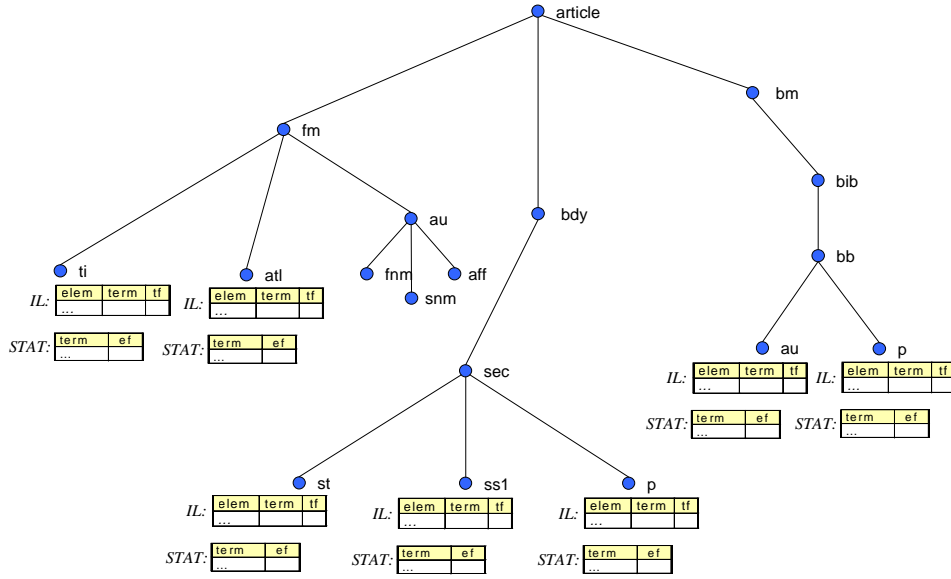


Figure 2: Example of basic indexing nodes for the INEX document collection

**Single-Category Retrieval.** Single-category retrieval with XML works on the element type that corresponds to a basic indexing node. The granularity of retrieval are all elements of that category. Topic 02 in Example 3 is an example of a single-category query. With single-category retrieval, we take over the usual definition of retrieval status value with the vector space retrieval model: As usual,  $t$  denotes a term, and  $tf(t, e)$  is its term frequency with an element  $e$ . Let  $N_{cat}$  and  $ef_{cat}(t)$  denote the number of elements at the single category  $cat$  and the element frequency of term  $t$  with the elements of  $cat$ , respectively. In analogy to the inverted document frequency for conventional vector space retrieval, we define *inverted element frequency* ( $ief$ ) as  $ief_{cat}(t) = \log \frac{N_{cat}}{ef_{cat}(t)}$ . The retrieval status value of an element  $e$  for a single-category query  $q$  is then

$$RSV(e, q) = \sum_{t \in terms(q)} tf(t, e) ief_{cat}(t)^2 tf(t, q) \quad (1)$$

**Multi-Category Retrieval.** In contrast to single-category retrieval, *multi-category retrieval* with XML works with *multi-categories*. Formally, a multi-category is given by a path expression that may contain choices. As with single-category retrieval, the granularity of retrieval with a multi-category are all elements that match the path expression. Topic 01 in Example 3 is an example of a multi-category query. When it comes to retrieval from a multi-category, statistics such as element frequencies for vector space retrieval and especially the  $rsv$  must reflect this. Otherwise, inconsistent rankings are possible. Our approach to guarantee consistent retrieval results is similar to integrating statistics for queries over different document categories with conventional retrieval [5]. We extend this notion now such that statistics for multi-category retrieval depend on the statistics of each single-category that occurs in the query. As the subsequent definitions show, our approach first computes the statistics for each single-category as defined in Definition 1 and then integrates them to the multi-category ones as follows. Let  $\mathcal{M}$  denote the set of basic indexing nodes of the multi-category.  $N_{mcat} = \sum_{cat \in \mathcal{M}} N_{cat}$  stands for the number of elements of the

multi-category. Thus,  $ief_{mcat}(t) = \log \frac{N_{mcat}}{\sum_{cat \in \mathcal{M}} ef_{cat}(t)}$ , where  $ef_{cat}(t)$  denotes the single-category element frequency of term  $t$  with category  $cat$ . The retrieval status value of an element  $e$  for a multi-category query  $q$  is then using again TFIDF ranking:

$$RSV(e, q) = \sum_{t \in terms(q)} tf(t, e) ief_{mcat}(t)^2 tf(t, q) \quad (2)$$

This definition integrates the frequencies of several single categories to a consistent global one. It equals Definition 1 in the trivial case with only one category in the multi-category.

**Nested Retrieval.** Another type of requests are those that operate on complete subtree of the XML documents. Topic 31 in Example 3 is an example of a nested-retrieval query. However, there are the three following difficulties with this retrieval type:

- A path expression may define a context of interest that comprises different categories in its XML subtree. Hence, retrieval over the complete subtree must differentiate between these element types to provide a consistent ranking.
- Terms that occur close to the root of the subtree are typically considered more significant for the root element than ones on deeper levels of the subtree. Intuitively: the larger the distance of a node from its ancestor is, the less it contributes to the relevance of its ancestor. Fuhr et al. [3, 4] tackle this issue by so-called *augmentation weights* which downweigh term weights when they are pushed upward in hierarchically structured documents such as XML.
- Element containment is at the instance level, and not at the type level. Consequently, element containment relations cannot be derived completely from the element type nesting.

More formally, let  $e$  denote an element that qualifies for the path expression of the nested-retrieval query. Let  $SE(e)$  denote the set

$$\begin{aligned}
RSV(e, q) &= \sum_{se \in SE(e)} \sum_{t \in terms(q)} tf(t, se) \left( \prod_{l \in path(e, se)} aw_l \right) ief_{cat(se)}(t)^2 tf(t, q) \\
&= \sum_{se \in SE(e)} \left( \prod_{l \in path(e, se)} aw_l \right) \sum_{t \in terms(q)} tf(t, se) ief_{cat(se)}(t)^2 tf(t, q)
\end{aligned}$$

**Figure 3: Retrieval status value with TFIDF ranking and nested retrieval**

of sub-elements of  $e$  including  $e$ , i.e., all elements contained by the sub-tree rooted by  $e$ . For each  $se \in SE(e)$ ,  $l \in path(e, se)$  stands for a label along the path from  $e$  to  $se$ , and  $aw_l \in [0.0; 1.0]$  is its augmentation weight.  $cat(se)$  denotes the category to which  $se$  belongs.  $ief_{cat(se)}(t)$  stands for the inverted element frequency of term  $t$  with the category  $cat(se)$ . The retrieval status value  $rsv$  of an element  $e$  under a nested-retrieval query  $q$  using the vector space retrieval model then yields the expression shown in Figure 3.

As the definitions in Figure 3 show, nested retrieval is a weighted sum of constrained single-category retrieval results. The constraint is such that an element  $se$  and its textual content only contribute to the retrieval status value of  $e$  if  $se$  is in the sub-tree rooted by  $e$ . Moreover, both definitions in the figure revert to the common TFIDF ranking for conventional retrieval on flat documents when all augmentation weights are equal to 1.0. In the trivial case where a nested query only comprises one single-category, the definitions in Figure 3 equal Definition 1.

#### 4. IMPLEMENTING FLEXIBLE RETRIEVAL FROM XML

In the following paragraphs, we explain how to implement multi-category retrieval and nested retrieval using the data of the basic indexing nodes.

**Multi-Category Retrieval.** Using the statistics of the basic indexing nodes directly for multi-category retrieval is not feasible since statistics are per element type. Hence, query processing must dynamically integrate the statistics if the query encompasses several categories. Using single-category statistics directly may lead to wrong rankings with multi-category queries. Multi-category queries compute the correct multi-category statistics during query processing. Algorithm **MULTICATEGORY** shown in Figure 4 reflects this. First, it determines the basic indexing nodes contained in the path expression of the multi-category query. Its second step is to retrieve the statistics for each such basic indexing node and to use them to compute the integrated ones. The third step executes the lookup in parallel at the inverted lists. The inverted list lookup takes the integrated multi-category statistics as input parameter and computes the partial ranking. The fourth step of **MULTICATEGORY** integrates the partial results from the third step and returns the overall ranking.

**Nested Retrieval.** As with the previous retrieval type, nested retrieval requires integrating statistics and processing queries over different indexes. In addition, it must also reflect element containment and augmentation weights properly. This makes processing of this query type more complex than with the other types. Our algorithm to process nested queries is called **NESTEDRETRIEVAL**, and it comprises four steps, as shown in Figure 5. The first step computes the categories that qualify for the path expression defining the scope of the nested query. The second step then iterates over the categories, their underlying basic indexing nodes, and dy-

#### Algorithm MULTICATEGORY

Parameters: Query  $q$ , path expression  $p$

**var** hits :=  $\emptyset$ ;  $\mathcal{M}$  :=  $\emptyset$ ;

**begin**

// Step 1: Determine the single-categories and

$\mathcal{M} = \text{LookUp}(p)$

// Step 2: Collect and integrate statistics

**for each** single-category  $cat \in \mathcal{M}$  **do in parallel**

Get per-category statistics ( $ef_{cat}(t)$ ,  $N_{cat}$ ); **end**;

Compute multi-category statistics  $stat_{mcat}$

( $ief_{mcat}$  and  $N_{mcat}$  for Def. 2);

// Step 3: Execute query for each category

**for each** category  $cat \in \mathcal{M}$  **do in parallel**

// process the query with the integrated statistics

hits := hits  $\cup$  Query $_{mcat}(cat, q, stat_{cat})$ ; **end**;

// Step 4: Post-processing and output of results

Sort hits by RSV; Return the ranking (element id and RSV);

**end**;

**Figure 4: Algorithm MULTICATEGORY**

namically generates the statistics for the appropriate vector space of the scope of the query. Note that the dynamically generated statistics  $STAT_{temp}$  comprise different inverted element frequencies ( $ief$ ) for the same term depending on the category where the term occurs and the weight of the category. The weighting function  $\mathcal{W}$  augments each term  $t \in q$  from the statistics  $STAT_{cat}$  with its proper augmentation weights regarding the context node of the query. This ensures that the properly augmented  $iefs$  are used to compute the  $rsv$ . The last step of the algorithm then computes the overall ranking.

#### 5. RELATED WORK

As a first measure to enhance functionality for document-centric processing of XML, Florescu et al. realize searching for keywords in textual content of XML elements [1]. However, the mere capability to search for keywords does not suffice to address the requirements for document-centric processing: support for state-of-the-art retrieval models with relevance ranking is needed. To tackle this issue, Theobald et al. propose the query language XXL and its implementation with the XXL Search Engine [9]. Similar to our approach with XPathIR, Fuhr and Großjohann et al. extend the W3C XPath Recommendation with operators needed for document-centric processing of XML [4, 7].

Regarding IR statistics such as inverted document frequencies ( $idf$ ), Fuhr et al. have already argued in [3, 4] that treating documents as flat structures comes too short for XML. They propose to down-weight terms by so-called augmentation weights when the terms are propagated upwards in the document hierarchy. However, [4] derive IR statistics such as  $idf$  for the collection as a whole. But, retrieval in different contexts requires a more dynamic treatment of

### Algorithm NESTEDRETRIEVAL

```
Parameters: Query  $q$ , path expression  $p$ 
var hits :=  $\emptyset$ ;  $\mathcal{N}$  :=  $\emptyset$ ;
begin
  // Step 1: Determine the single-categories
   $\mathcal{N}$  = LookUp( $p$ )

  // Step 2: Compute integrated statistics with augmented weights
  //  $\mathcal{W}(STAT_{cat}, \prod_{l \in path(base(p), cat)} aw_l)$  denotes the
  // weighted projection of the per-category statistics
  //  $base(p)$  denotes the element type of the query root
  for each category  $cat \in \mathcal{N}$  do in parallel
     $STAT_{temp} := STAT_{temp}$ 
     $\cup \mathcal{W}(STAT_{cat}, \prod_{l \in path(base(p), cat)} aw_l)$  end;

  // Step 3: Process the query on each category
  // with the augmented statistics
  for each category  $cat \in \mathcal{N}$  do in parallel
    hits := hits  $\cup$  Query $_{ncat}(q, STAT_{temp})$ ; end;

  // Step 4: Post-processing and output of results
  Sort hits by RSV; Return the ranking (element id and RSV);
end;
```

Figure 5: Algorithm NESTEDRETRIEVAL

term weights. Hiemstra comes to a similar conclusion for query term weights used in different query contexts [8]. Therefore, our approach proposed in [6] keeps different IR statistics for each basic indexing node. This allows for consistent retrieval with arbitrary query granularities, i.e., arbitrary combinations of element types.

## 6. CONCLUSIONS

Flexible retrieval is an important requirement with document-centric processing of XML. Flexible retrieval means that users define the scope of their queries dynamically, i.e., at query time. The different topics developed within the INEX framework reflect this requirement, defining both content-and-structure and content-only queries. To cover this requirement, the XML engine PowerDB-XML currently being developed at ETH Zurich extends the W3C XPath path expression language to XPathIR, a path expression language that allows for flexible retrieval from XML documents.

The difficulty with flexible retrieval on XML is to treat statistics such as document frequencies properly in the context of hierarchically structured data with possibly heterogeneous contents: the common assumption to derive IR statistics such as document frequencies for the collection as a whole does not necessarily hold with XML. To tackle this issue, PowerDB-XML integrates vector spaces on-the-fly, i.e., during query processing, to a consistent view of the statistics that properly reflects the scope of the query. Our implementation is based on the three basic retrieval operations *single-category retrieval*, *multi-category retrieval*, and *nested retrieval* that form the building blocks for processing information retrieval queries on XML content. PowerDB-XML currently deploys vector-space TFIDF ranking. Proper treatment of statistics with flexible retrieval from structured documents however is an issue that similarly arises for all weighted retrieval models. With these retrieval models as well, integration of statistics according to single-category, multi-category, and nested retrieval is necessary to guarantee consistent ranking. The collection of XML documents as well as the set of topics provided with the INEX testbed serves as our framework to further evaluate PowerDB-XML regarding both retrieval quality and retrieval efficiency.

## 7. REFERENCES

- [1] D. Florescu, D. Kossmann, and I. Manolescu. Integrating Keyword Search into XML Query Processing. In *Proc. of the Intern. WWW Conference, Amsterdam, May 2000*. Elsevier, 2000.
- [2] N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas. INEX: Initiative for the Evaluation of XML Retrieval. In *Proceedings of the ACM SIGIR Workshop on XML and Information Retrieval, Tampere, Finland*, pages 62–70. ACM Press, 2002.
- [3] N. Fuhr, N. Gövert, and T. Rölleke. 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.
- [4] N. Fuhr and K. Großjohann. 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.
- [5] T. Grabs, K. Böhm, and H.-J. Schek. PowerDB-IR – Information Retrieval on Top of a Database Cluster. In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM2001), November 5-10, 2001 Atlanta, GA, USA*, pages 411–418. ACM Press, 2001.
- [6] T. Grabs and H.-J. Schek. Generating Vector Spaces On-the-fly for Flexible XML Retrieval. In *Proceedings of the ACM SIGIR Workshop on XML and Information Retrieval, Tampere, Finland*, pages 4–13. ACM Press, 2002.
- [7] K. Großjohann, N. Fuhr, D. Effing, and S. Kriewel. Query Formulation and Result Visualization for XML Retrieval. In *Proceedings of the ACM SIGIR Workshop on XML and Information Retrieval, Tampere, Finland*, pages 26–32. ACM Press, 2002.
- [8] D. Hiemstra. Term-specific smoothing for the language modeling approach to information retrieval: the importance of a query term. In *Proceedings of the 25th Annual ACM SIGIR Conference on Research and Development in Information Retrieval 2002, Tampere, Finland*, pages 35–41. ACM Press, 2002.
- [9] A. Theobald and G. Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In *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.
- [10] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. <http://www.w3.org/TR/1998/REC-xml-19980210>, Feb. 1998.
- [11] World Wide Web Consortium. XML Path Language (XPath) Version 1.0. <http://www.w3.org/TR/xpath>, Nov. 1999.
- [12] World Wide Web Consortium. XQuery 1.0: An XML Query Language. <http://www.w3.org/TR/xquery>, Feb. 2001.

# Applying the IRstream Retrieval Engine for Structured Documents to INEX

Andreas Henrich  
University of Bayreuth  
D-95440 Bayreuth, Germany  
andreas.henrich@uni-bayreuth.de

Günter Robbert  
University of Bayreuth  
D-95440 Bayreuth, Germany  
guenter.robber@uni-bayreuth.de

## ABSTRACT

For a long period of time the research activities in information retrieval have mainly addressed flat text files. Although there have been approaches towards multimedia data and structured data in the past, these topics gain increasing interest today in the context of XML data. To address structured multimedia data, an efficient combination of content-based retrieval for multimedia data, retrieval in meta data and mechanisms which allow to exploit the document structure is needed.

To this end, we propose *IRstream* as a general purpose retrieval service for structured multimedia documents. IR-Stream is intended as a powerful framework to search for components of arbitrary granularity – ranging from single media objects to complete documents. At this, 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. We describe the significant features of this approach and point out the system architecture. Furthermore, we present the application of IRstream as a retrieval system for XML documents in the context of INEX.

## 1. MOTIVATION

Today, electronic documents are more than flat text, rather they form a complex structure of different parts. Besides text data, we can find other media types like audio, image, and video. Furthermore, documents can contain meta data concerning the contained media objects, the internal document structure, and the document itself.

To deal with such documents, we need an efficient combination of (1) content based retrieval techniques for text and multimedia data, (2) search mechanisms which can address and exploit the structure of the documents, (3) retrieval in meta data, and (4) traditional retrieval facilities such as fact retrieval or pattern matching. Finally – according to

the experiences in the information retrieval community – the retrieval system should yield a ranking based on some type of similarity conditions. In the context of structured multimedia data, the system has to allow for a flexible and precise definition of these similarity conditions.

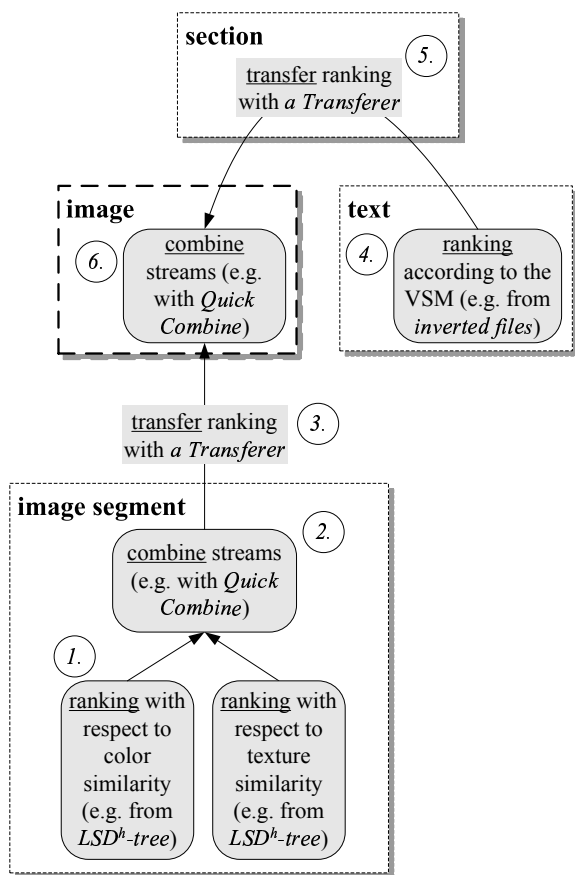
In the present paper, we propose a stream-oriented approach to process such complex similarity-based queries. The basic idea is to deploy access structures efficiently supporting similarity queries wherever possible. These access structures produce initial streams which can be combined and transferred afterwards. To this end, we use components which combine multiple rankings (usually derived for different ranking criteria) and transfer rankings derived for objects of a certain type to objects of a related type. An important feature of the approach is that it is pull-based, i.e. each stream extracts elements from its input streams only on demand. This can be seen as a lazy evaluation approach, where each input stream is produced only to the extent needed to produce the desired number of elements in the final output stream presented to the user.

Obviously, this approach is not only applicable with structured multimedia documents, but also in the area of structured text documents. Especially the increasing use of XML in digital libraries, product catalogues, scientific data repositories and across the Web encouraged the development of appropriate searching and browsing methods. For this reason, the Initiative for the Evaluation of XML retrieval (INEX) [5] initiated an international, coordinated effort to promote evaluation procedures for content-based XML retrieval. INEX provides an opportunity for participants to evaluate their retrieval methods using uniform scoring procedures and a forum for participating organizations to compare their results. As a participating organization, we applied IRstream to the collection of XML documents provided by INEX. Hereby, we investigated the usability of IRstream for structured text documents.

The rest of the paper is organized as follows: In section 2 we will give a first rough description of our approach. Thereafter we will go into the details of the main components of IRstream in section 3. The concrete architecture of our IRstream implementation is presented in section 4. Section 5 shows how IRstream can be used as a retrieval engine for XML documents in the context of INEX and presents the experiences gained. Finally, section 6 concludes the paper.

## 2. A FIRST VIEW

A first impression of our approach can be given best by an example. Such an example for a query dominated by



**Figure 1: Stream-oriented processing of our example query**

ranking conditions might arise when the user is searching for images maintained in structured multimedia documents. Here, the user might be interested in images containing a given logo – i.e. images which contain a segment similar to the given logo – where the text nearby the image is dealing with skiing or winter sports in general. This query contains two ranking conditions: (1) There is a ranking condition for the text in the vicinity of the desired images and (2) a ranking condition for image segments which are required to be similar to a given logo.

Now we assume that the multimedia documents consist of a set of sections. Each section contains images and/or text blocks. Furthermore, each image is associated with several image segments. In this case, our example query searching for images containing a given logo where the text nearby the image is dealing with skiing or winter sports in general can be processed as depicted in figure 1.

First, two different rankings are generated for the image segments delivering these image segments sorted according to their color and texture similarity, respectively, compared to the given logo. To this end, feature vectors representing the color and texture characteristics of each image segment are applied. Comparing these vectors with the given logo, two *retrieval status values* are calculated for each image segment defining the rankings for the color and texture similarity. For the efficient stepwise calculation of these rankings various access structures have been proposed, such as the M-

tree, the X-tree or the  $LSD^h$ -tree [15, 1, 8]. In figure 1 this part of the query evaluation process is indicated as step 1.

Then the rankings derived for the two criteria have to be combined into a single weighted ranking (step 2). To this end, algorithms such as Fagin’s algorithm [2, 3], Nosferatu [14] or Quick-Combine [6] can be deployed.

Now we have derived a combined ranking for the image segments. However, what is needed is a ranking for the images themselves. To derive this ranking, we transfer the ranking for the image segments to the images. To this end, we exploit that each image segment is associated with some type of retrieval status value determining the ranking of the image segments. As a consequence, we can transfer the ranking for the image segments to the images based on these retrieval status values. For example, we can associate the maximum retrieval status value of a related image segment with each image. To implement this transfer of the ranking, we consider the ranking for the image segments one element after another, determine the associated image and calculate the corresponding ranking of the images (step 3). More details of this algorithm will be presented in section 3.3.

Now we have to derive a second ranking for the images with respect to the requirement that the text nearby the image – i.e. in the same section – is dealing with skiing or winter sports in general. To this end, a ranking for the text blocks can, for example, be created via an implementation of the vector space model using inverted files (step 4). Then this ranking has to be transferred from the text blocks to the images in the same section (step 5). Now we have got two rankings for the images: one concerning the “logo criterion” and one concerning the “text in the vicinity criterion”. Finally these rankings have to be combined to yield a common ranking for the images (step 6).

### 3. 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.

#### 3.1 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.



## 3.2 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 [4]. Other approaches are based on the ranks of the objects with respect to the single criterion [12, 9]. To calculate such a combined ranking efficient algorithms, such as Fagin’s algorithm [2, 3], Nosferatu [14], Quick Combine [6] and  $J^*$  [13] can be deployed.

## 3.3 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. This transfer of a ranking onto related objects seems to be worth a more in-depth consideration.

Before we can explain the algorithm for the transfer of a ranking, we have to clarify the semantics of this transfer. To this end, we consider a simplified example query where the user is searching for images containing an image segment similar to a given logo. Here the situation is as follows: We have a retrieval status value for the image segments. This value allows to derive a ranking for the image segments. However, we are not interested in a ranking of the image segments but in a ranking of the images. Therefore it is necessary to derive a retrieval status value for each image.

Let  $RSV_r(ro)$  be the retrieval status value of object  $ro$  ( $ro$  for “related object” and  $RSV_r$  for the  $RSV$  values of “related” objects). In our example  $ro$  would be an image segment. Further let  $\{ro_{i,1}, ro_{i,2}, \dots, ro_{i,n_i}\}$  be the set of related objects associated with the “desired object”  $do_i$ . In our example this set would contain the image segments associated with the image  $do_i$ . Finally let us assume that high  $RSV$  values stand for well fitting objects. Then we need a function  $\mathcal{F}$  deriving the retrieval status value  $RSV_d(do_i)$  from the objects associated with  $do_i$  and their  $RSV$  values:

$$RSV_d(do_i) \stackrel{\text{def}}{=} \mathcal{F} \left( \begin{array}{l} \langle ro_{i,1}, RSV_r(ro_{i,1}) \rangle, \\ \langle ro_{i,2}, RSV_r(ro_{i,2}) \rangle, \\ \vdots \\ \langle ro_{i,n_i}, RSV_r(ro_{i,n_i}) \rangle \end{array} \right)$$

Examples for meaningful choices for  $\mathcal{F}$  are the maximum  $RSV_r$  value, the average  $RSV_r$  value, a weighted average  $RSV_r$  value, or even the minimum  $RSV_r$  value.

Now the problem which has to be solved by a transferer can be described as follows: 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  $ot_r$  (*image segments* for example).

We assume that the relationship between these objects is well-defined and can be traversed in both directions. For our example, this means that we can determine the concerned image for an image segment and that we can determine the related image segments for an image. In this situation there will be only one concerned image for each image segment but situations are conceivable where a related object is shared by multiple desired objects. In this case, we get multiple objects of type  $ot_d$ .

In addition, we assume there is an input stream yielding a ranking for the objects of type  $ot_r$ .

Based on these assumptions, the “transfer algorithm” can proceed as follows. It uses the stream with the ranked objects of type  $ot_r$  as input. For the elements from this stream, the concerned object – or objects – of type  $ot_d$  are computed traversing the respective relationships. Then the  $RSV_d$  values are calculated for these objects of type  $ot_d$  according to the desired semantics and the object of type  $ot_d$  under consideration is inserted into an auxiliary list maintaining the objects considered so far. In this list, each object is annotated with its  $RSV_d$  value. Now the next object of type  $ot_r$  from the input stream is considered. If the  $RSV_r$  value of this object is smaller than the  $RSV_d$  value of the first element in the auxiliary list which has not yet been delivered in the output stream, this first element in the auxiliary list can be delivered in the output stream of the transfer component.

For a more detailed consideration, we have to define the characteristics of the auxiliary list  $AL$ .  $AL$  maintains pairs  $\langle do_i; RSV_d(do_i) \rangle$  with  $\text{type}(do_i) = ot_d$ . These pairs are sorted in descending order with respect to their  $RSV_d$  values. For  $AL$  the following operations are needed:  $\text{createAL}()$  creates an empty auxiliary list.  $\text{getObj}(AL, i)$  yields the object with the  $i^{\text{th}}$  highest  $RSV_d$  value stored in  $AL$ .  $\text{getRSV}(AL, i)$  returns the  $RSV_d$  value for the object with the  $i^{\text{th}}$  highest  $RSV_d$  value stored in  $AL$ .  $\text{contains}(AL, do_j)$  checks whether there is an entry for object  $do_j$  in  $AL$ .  $\text{insert}(AL, \langle do_i; RSV_d(do_i) \rangle)$  inserts the entry for  $do_i$  into  $AL$  preserving the sorting with respect to  $RSV_d$  – moreover, if other objects with the same  $RSV_d$  value are already present in  $AL$ , the new object is placed behind these objects in  $AL$ .  $\text{size}(AL)$  returns the number of entries in  $AL$ .

Based on these definitions, we can state a class *Transferer* which provides a constructor and a  $\text{getNext}$  method. This class is given in pseudo-code in figure 2. The attributes which have to be maintained for a transferer comprise the input stream, a definition of the desired relationship between the objects of type  $ot_r$  and  $ot_d$ , the auxiliary list, a variable  $o_r$  which stores the next object of the input stream, and the number of delivered objects.

It has to be mentioned that the *maximum* semantics allows for some simplifications of the presented algorithm. With this semantics, there is no need to calculate  $RSV_d$  values in the **foreach** loop, because if there is no entry for  $o_d$  in  $AL$ ,  $o_r$  is surely the related object with the highest  $RSV_r$  value for  $o_d$ . Consequently,  $RSV_d(o_d) = RSV_r(o_r)$  holds, and the operation  $\text{insert}(AL, \langle o_d; RSV_d(o_d) \rangle)$  in the  $\text{getNext}$  method can be replaced by the more efficient operation  $\text{insert}(AL, \langle o_d; RSV_r(o_r) \rangle)$ .

## 3.4 Filters

Of course, it must be possible to define filter conditions for all types of objects. With our stream-oriented approach

```

Class Transferer {
  Stream : inputStream;
  RelationshipDef : rel_d; /* desired relationship */
  AuxiliaryList : AL;
  InputObject : o_r; /* next object to be considered */
  Integer : n; /* no. of next object to be delivered */
  constructor(Stream : input, RelationshipDef : rel) {
    inputStream := input;
    rel_d := rel;
    AL := createAL();
    o_r := streamGetNext(inputStream);
    if o_r = ⊥ then exception("empty input stream");
    n := 1;
  }
  getNext() : OutputObject {
    while o_r ≠ ⊥ ∧ (size(AL) < n
      ∨ RSV_r(o_r) ≥ getRSV(AL, n)) do
      /* consider the next input object o_r */
      SDO := {o_d | ∃rel_d(o_r → o_d)};
      /* all objects which can be reached via
        the desired relationship */
      foreach o_d ∈ SDO do
        if ¬contains(AL, o_d) then
          insert(AL, ⟨o_d; RSV_d(o_d)⟩);
        end /* foreach */;
      o_r := streamGetNext(inputStream);
    end /* while */;
    if o_r = ⊥ ∧ size(AL) < n then
      return ⊥; /* stream exhausted */
    else
      n++;
      return getObj(AL, n - 1);
    end /* if */;
  }
}

```

Figure 2: Class *Transferer* in pseudo code

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.

#### 4. 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 3 depicts this architecture.

The core IRstream system — shaded grey in figure 3 — comprises four main parts: (1) Implementations for rankers, combiners, transferers, and filters. (2) Implementations of various methods for the extraction of feature “values” as

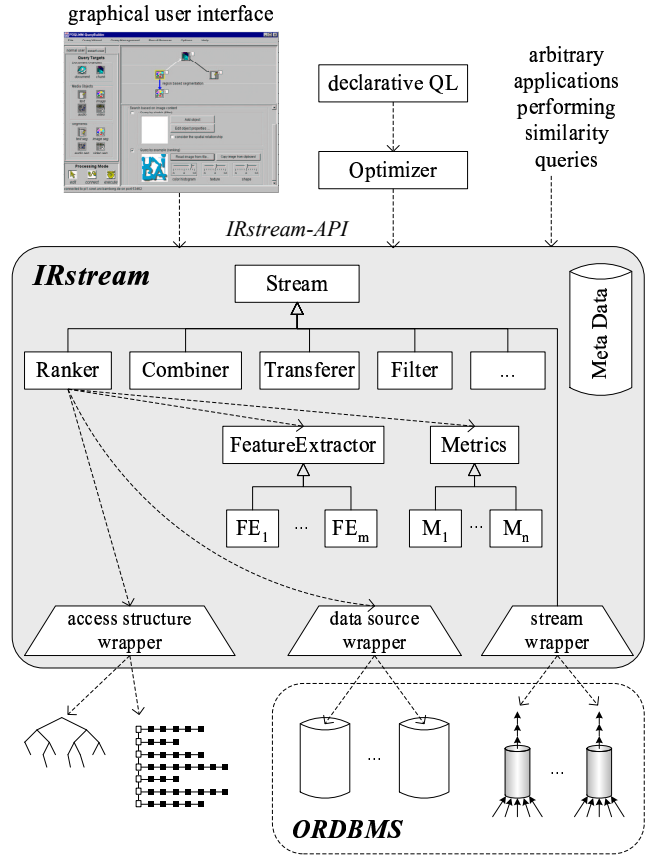


Figure 3: Architecture of the IRstream system

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 algo-

rithm 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 of our system maintains meta 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. Here the meta data can e.g. be used to control the query construction in a graphical user interface.

### Wrapper

*Data source wrappers* are needed to attach systems maintaining the objects themselves to our retrieval system. At present, ORDBMSs can be attached via JDBC.

*Access structure wrappers* can be used to deploy access structures originally not written for our system. For example, we incorporated an LSD<sup>b</sup>-tree implementation written in C++ via a corresponding wrapper. In general, this interface should be used to attach access structures which can maintain collections of feature values and perform similarity queries on these values.

Finally, *stream wrappers* can be used to incorporate external stream producers. At present, the text module of the underlying ORDBMS is integrated via a stream wrapper. In contrast to an access structure, such an external stream producer provides not only a ranking but also access to the maintained objects themselves. This means that an external stream producer is aware of the objects themselves, whereas an external access structure does only maintain feature values and associated object references.

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 [10]. 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<sup>MM</sup> query language [7, 11].

## 5. IRSTREAM IN THE CONTEXT OF INEX

To assess the applicability of our IRstream approach as a retrieval engine for XML documents, we performed up to sixty retrieval runs on the INEX test collection containing more than ten thousand documents. All these documents were inserted into the ORDBMS underlying our system.

document part	cardinality
journal	124
article	11,993
author	21,902
frontmatter	11,993
body	11,993
backmatter	9,954
section/subsection/...	140,417
paragraph	1,398,494

**Table 1: Addressable document parts and their cardinality**

To this end, we parsed all documents and decomposed them hierarchically into several parts. Table 1 depicts all document parts and their cardinality. By these means, we can address different granules of the documents in order to support a search concerning the document structure.

Furthermore we implemented a specialized ranker for XML data which internally uses the text retrieval functionality provided by the underlying ORDBMS, and incorporated this ranker into our IRstream retrieval engine. Using this approach, we were able to deal with all sixty topics.

In the following, we point out how the query processing in IRstream is done by means of a typical example topic. To this end, we consider topic 3, which is a so-called *content and structure topic* (CAS):

<b>Title:</b> information data visualization
<b>Context:</b> <i>Keyword:</i> information data visualization <i>Document:</i> large information hierarchies spaces multidimensional data databases
<b>Description:</b> I am looking for techniques for visualizing large information hierarchies or information spaces.
<b>Narrative:</b> For a document or document element to be considered relevant, the document (element) has to deal with visualization techniques for data mining or visualization techniques for large textual information spaces or hierarchies. Document/document components describing visualization of any multidimensional data (be it hierarchical or otherwise) are relevant. Documents describing rendering techniques and algorithms are not relevant.

To process topic 3 we used three rankers, three transfers and one combiner. Figure 4 shows the involved components and their interaction for the stream-oriented processing of topic 3 with IRstream.

First we used one ranker to determine a ranking for the document parts of type *frontmatter*, where the attribute *keyword* (tag <keyword>) contains terms like "*information data visualization*". In parallel, we employed two rankers to acquire a ranking for the document parts of type *body* (tag <body>) concerning the terms "*information hierarchies*" and "*information techniques*". The original query text and the addressed document granule are depicted in the boxes of figure 4 named *XML ranker*.

In order to get whole articles as result elements, we used three transferers applying the maximum semantics to map the results of the different streams onto the document type

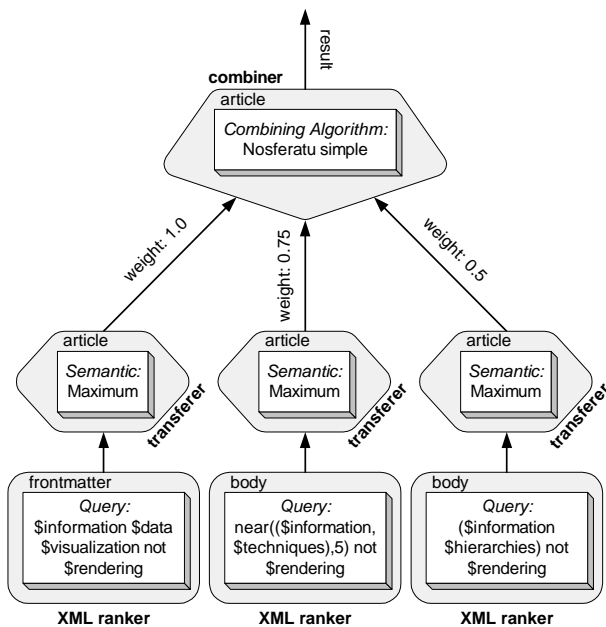


Figure 4: Stream-oriented processing of topic 3

article.

Last but not least, to achieve the final result we used a combiner to merge the ranking of the three incoming streams using the algorithm Nosferatu simple [14]. For the merging of the different input streams, a weight was assigned to each stream in order to control the influence of the different document parts. The weights are noted at the arrows leading from the transferers to the combiner in figure 4.

For all topics the average response time of the IRstream retrieval engine was about one second. It has to be noted that all query processing has been performed with a first IRstream prototype. This prototype implemented in Java is by no means optimized.

## 6. CONCLUSION AND FUTURE WORK

In this paper, we have presented an approach for the stream-oriented processing of complex similarity queries. The approach is intended to complement traditional query processing techniques for queries dominated by similarity conditions. The approach has been implemented as a prototype in Java on top of an ORDBMS and first experimental results achieved with this prototype are promising.

In the near future, we will address the optimization of the prototype implementation and perform experiments with larger test collections. Furthermore, we will develop a query language for this approach and consider optimization issues regarding 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 — into the query execution in order to improve the precision of the system.

## 7. REFERENCES

[1] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An index structure for high-dimensional data. In *VLDB'96, Proc. 22th Intl. Conf. on Very Large Data Bases*, pages 28–39, Mumbai, India, 1996.

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

[3] 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.

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

[5] N. Fuhr, M. Lalmas, G. Kazai, and N. Gvert. *Initiative for the Evaluation of XML retrieval (INEX)*. Online available: url: <http://qmir.dcs.qmul.ac.uk/inex/>, 2002.

[6] U. Guntzer, W.-T. Balke, and W. Kiefling. 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.

[7] 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.

[8] A. Henrich. The LSD<sup>b</sup>-tree: An access structure for feature vectors. In *Proc. 14th Intl. Conf. on Data Engineering, Orlando, USA*, pages 362–369, 1998.

[9] 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.

[10] 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.

[11] A. Henrich and G. Robbert. POQL<sup>MM</sup>: 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.

[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.

[15] P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with M-trees. *VLDB Journal*, 7(4):275–293, 1998.

# CWI at INEX2002

Johan List and Arjen P. de Vries

Center for Mathematics and Computer Science (CWI)  
Data Mining and Knowledge Discovery (INS1)  
P.O.Box 94079, 1090GB Amsterdam, The Netherlands  
{j.a.list, a.p.de.vries}@cwi.nl

## ABSTRACT

This paper describes our participation in INEX 2002 (the XML Retrieval Initiative) and discusses several aspects of our XML retrieval system: the retrieval model, the document indexing and manipulation scheme and our preliminary evaluation results of the submitted three runs.

In our system, we have used a probabilistic retrieval model where we map (structural) properties of documents to *dimensions of relevance* and use these dimensions of relevance for retrieval purposes. The study concentrates on *coverage*, defined as the amount of relevant information present in a document component. We also discuss an efficient and database-independent indexing scheme for XML documents, based on text regions and discuss region operators for selection and manipulation of XML document regions.

Initial evaluation of our results, with a rather adhoc approach, made clear that evaluation measures for structured document retrieval needs more discussion and research.

## 1. INTRODUCTION

This paper describes our participation in INEX (XML Retrieval Initiative). We participated with our XML retrieval system, using a research database kernel, MonetDB.

The primary goals for participation in the XML Retrieval Initiative were 1) to gain experience in information retrieval of documents possessing various degrees of semantic structure, 2) to look for possibilities to introduce structural properties of documents into probabilistic retrieval models and 3) to examine whether the use of structure information can improve retrieval performance.

The construction of any information retrieval system (and as such an XML retrieval system) can be thought of to address three components: document representation, the retrieval model and query formulation. Document representation defines the logical and physical representation of documents in a retrieval system.

'Flat' documents are mostly represented with techniques such as inverted lists, but in the case of structured documents we need to represent the structural aspects of documents as well.

The use of structure plays a possible role as well in addressing the second component, the definition of the retrieval model. The basis for our model is a probabilistic retrieval model, the statistical language model discussed in [10].

The third component deals with query formulation. The extra dimension of structure in XML documents plays a role here as well: how is structural information integrated in the query possibilities and in what sense do query formulation possibilities depend on user knowledge of the structure(s) present in the collection?

The main contributions of this paper are twofold. We present an efficient and database-independent indexing scheme for XML documents based on *XML document regions*. We then describe a probabilistic retrieval model where we map (structural) properties of documents to dimensions of relevance and use these dimensions of relevance for retrieval purposes. The study concentrates on *coverage*, defined as the amount of relevant information present in a document component.

## 2. THE RETRIEVAL MODEL

We consider approaches based on using a query language capable of expressing structural constraints only on top of a database system as *data retrieval approaches*. These approaches assume users have detailed knowledge of the structure of the documents present in the collection and are only practically useful in homogeneous document collections. Since it is difficult to give an a-priori estimation for the importance of document structure in the retrieval process, we hypothesise that a more accurate picture of the influence of structure on document relevance is most likely to be a question of increasing insight obtained by the user during the query process.

In our system we use a probabilistic retrieval model and generally, additional knowledge is encoded with the prior probabilities of such models. For example, Westerveld et al. [17] used this strategy successfully to increase the likelihood of finding entry pages in a Web retrieval task. Also, a prior on document length improved retrieval performance at TREC-style experiments [10], based on the assumption that longer documents have a higher probability of containing relevant information.

Research in the user modeling and concept of relevance areas (see ao. [2], [3], [4], [1]) suggests that relevance is a multi-

dimensional concept of which *topicality* (i.e. content-based relevance) is only a single one. Mizarro [15] names other, possible non-topical dimensions *abstract characteristics of documents* constructed independently from the particulars of the database or collection at hand. In other words: other, non-topical dimensions are constructed independently from the language models present in the documents of a collection, suggesting orthogonality between the topicality dimension and any additional dimensions. Examples of other, non-topical dimensions include comprehensibility (style or difficulty of the text) and quantity (how much information does the user want; this is measured in a.o. the size of documents and the number of documents returned to the user).

Furthermore, encoding additional knowledge in prior probabilities makes it more difficult to reliably (re-)estimate dimension models, due to the possible noise non-dimension related prior probabilities introduce.

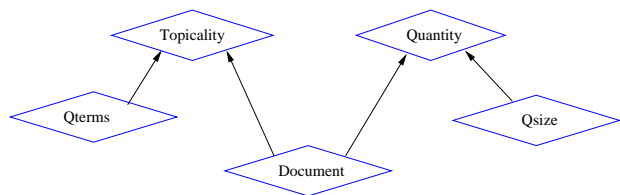
We believe that, because of the process of increasing insight obtained by the user during the query process and the orthogonality of additional dimensions of relevance, these dimensions can be estimated separately and these dimensions can be modeled with a set of independent probabilities (independent given a document instantiation) in a probabilistic retrieval model. This approach is visualized in Figure 1. The main idea behind this model is to determine whether we can effectively map additional dimensions to document properties (structural or otherwise) that in turn can be represented by (probabilistic) entities in the retrieval model.

In our experiments, we were able to experiment with ‘coverage’, as used in the INEX XML Retrieval initiative. Coverage is defined as how much of the document component is relevant to the topic of request. Estimating the right amount of coverage for a search request plays a significant role in the case of structured document retrieval where the desirable retrieval unit is not known a-priori. Effective determination of the retrieval unit is a key issue which distinguishes structured document retrieval from traditional retrieval (where the retrieval unit is fixed a-priori).

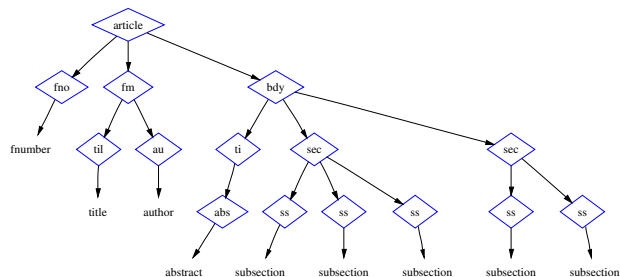
To further illustrate the model, consider a short motivating example. Let us assume we have the example document in Figure 2. Now, the system that estimates topicality identifies one relevant subsection in the first section and one relevant subsection in the second section. The open question is then whether to return the two separate subsections, or the separate sections or single body containing these as well as the remaining (possibly irrelevant) subsections. The additional context provided by the full sections or body may be more desirable for a user than the individual two subsections in isolation.

Assume a user is trying to solve the retrieval unit question and decides to use coverage as an additional relevance dimension. For modeling coverage, the user decides to regard coverage as a function of both topicality of document components and the size of document components (size being an aspect of the quantity dimension). The user reasons that:

- the shorter the document component is, the more likely it will not contain enough information to fulfill the information need;
- the longer the document component is, the more likely it is that distilling the topically relevant information will take substantial more reader effort.



**Figure 1: Encoding of additional relevance dimensions. Note that  $Q_{terms}$  and  $Q_{size}$  denote information given by the query (query terms and preferred component size).**



**Figure 2: Running example XML syntax tree.**

Now, when a user is ranking a document collection with regard to coverage, a ranking is performed against a combination of both topicality relevance and quantity relevance (where the user uses document component size as a representation of quantity). In probabilistic terms we are calculating the probability of complete relevance of a document component, given topicality relevance and quantity relevance.

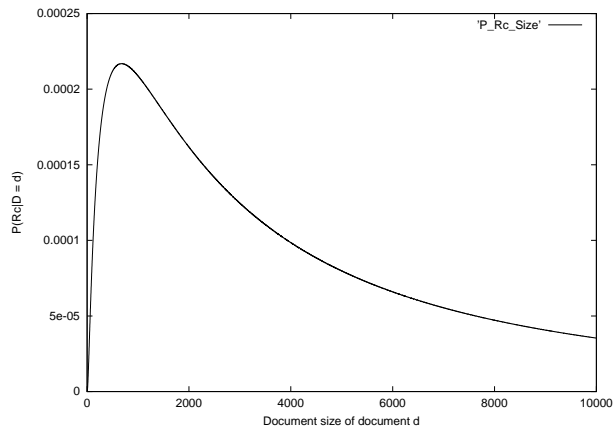
## 2.1 Modeling Relevance Dimensions

Firstly, for modeling additional relevance dimensions, we prefer a probabilistic description. The model in Figure 1 leads to the following. When  $P(R_t|D_d)$  is the probability of topical relevance given document  $d$  and  $P(R_q|D_d)$  is the probability of quantity relevance given document  $d$ , then we can calculate a joint probability of ‘complete’ relevance or user satisfaction as:

$$P(D_d, R_t, R_q, Q_{terms}, Q_{size}) = P(R_t|D_d, Q_{terms})P(R_q|D_d, Q_{size})P(D_d)$$

Looking at the motivating example in Section 2 and especially the user reasoning for modeling the quantity dimension, we decided to use a log-normal distribution. It is a distribution characterized by both a steep slope at the start and a long tail (as can be seen from Figure 2.1). The steep slope at the start reflects the ‘punishing’ behavior we want to model for (extremely) short document components. The long tail reflects that we do want to punish long document components, but not as harshly as extremely short ones (since these still might be useful, even while taking more reader effort to distill the relevant information). Secondly, we need the modeling parameter for the distribution itself. We have chosen for component size, but other possibilities include:

- the depth of the document component in the tree structure, where we want to penalize components present deep in



**Figure 3: The log-normal distribution used for modeling the quantity dimension**

the trees (generally small components and too specific) or components present high in the trees (generally large components and too broad);

- the number of children of a document component. A short document component containing a large amount of children highly likely contains a diversified mix of information and a could be less desirable for a user than a more homogeneous component.

## 2.2 Modeling Topicality

The model used for describing topicality of documents is a probabilistic model, the statistical language model described by Hiemstra [10]. The main idea of this model is to extract and to compare document and query models and determine the probability that the document generated the query. In other words, the statistical language model extracts linguistic information and is suited for modeling of the topicality dimension of the information need.

In deriving document models for all of the documents in the collection, we regarded every subtree present in the collection as a separate document. The probability of topical relevance  $P(R_t|D_d, Q_{terms})$  where  $Q_{terms}$  consists of the set of query terms  $\{T_1, \dots, T_n\}$  is calculated with:

$$P(R_t|D_d, Q_{terms}) = P(R_t|D_d, T_1, \dots, T_n) = P(D_d) \prod_{i=1}^n P(I_i)P(T_i|I_i, D_d)$$

where  $P(I_i)$  is the probability that a term is important (the event  $I$  has a sample space of  $\{0, 1\}$ ).

We follow the reasoning of Hiemstra [10] to relate the model to a weighting scheme (tf.idf-based). After some manipulation of the model we get:

$$P(D_d, T_1, \dots, T_n) \propto P(D_d) \prod_{i=1}^n \left(1 + \frac{\lambda P(T_i|D_d)}{(1-\lambda)P(T_i)}\right)$$

As estimators for  $P(D_d)$ ,  $P(T_i|D_d)$  and  $P(T_i)$  we used:

$$P(D_d) = \frac{1}{n} \quad (1)$$

$$P(T_i|D_d) = \frac{tf_{i,d}}{\sum_i tf_{i,d}} \quad (2)$$

where  $n$  is the number of documents,  $tf_{i,d}$  is the term frequency of term  $i$  in document  $d$  and  $\sum_i tf_{i,d}$  is the length of document  $d$ .

For  $P(T_i)$  we used:

$$P(T_i) = \frac{df_i}{\sum_i df_i} \quad (3)$$

where  $df_i$  is the document frequency of term  $i$ .

Filling in the likelihood estimators gives us the following model for topicality (with a constant  $\lambda$  for all terms):

$$P(R_t|D_d, Q_{terms}) = P(R_t|D_d, T_1, \dots, T_n) \propto \sum_{i=1}^n \log\left(1 + \frac{\lambda}{1-\lambda} \frac{tf_{i,d}}{\sum_i df_{i,d}} \frac{\sum_i df_i}{df_i}\right)$$

We used a very simple query model resulting in query term weights represented with  $tf_{i,q}$ , the term frequency of term  $i$  in query  $q$ .

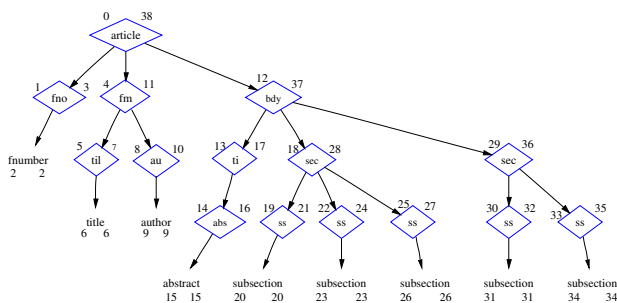
## 3. XML DOCUMENT INDEXING AND MANIPULATION

### 3.1 Document Model

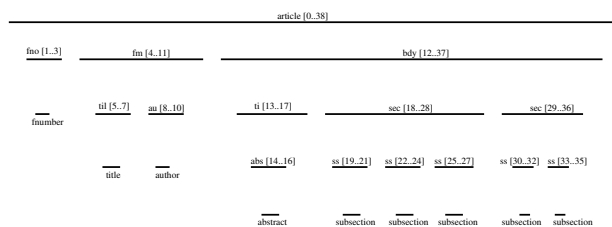
Generally, XML documents are represented as rooted (syntax) trees and indexing schemes focus on storage of the edges present in the syntax tree, combined with storage of the text present. One of these approaches is described by Schmidt [16], which we used as a starting point for our own indexing scheme. In Schmidt's approach, each unique path is stored in a set of binary relations where each binary relation represents an edge present in the path. Furthermore, multiple instances of the same path (even if they are present in different syntax trees) are stored in the identical set of relations. The system also maintains a schema of the paths present and their corresponding relations: the *path summary*.

The advantage of Schmidt's approach is that the execution of pure path queries can be performed efficiently; selecting the nodes belonging to a certain path prevents a forced scan of (large) amounts of irrelevant data, requiring only a fast lookup in the path summary to get to the relation required. The disadvantage is that the generation of the transitive closure of a node is an expensive operation. In database terms: the transitive closure is the union of the separate paths present in the component. The reconstruction of each path is performed with join operations, where the number of join operations depends on the number of steps present in the path.

Since we need fast access to the component text for determining statistics, we pursued another approach. Instead of seeing an XML document instance as a syntax tree, we see each XML document instance as a linearized string or a set of *tokens* (including the document text itself). Each component is then a text region or a contiguous subset of the entire linearized string. The linearized string of the example document in Figure 2 is shown below:



**Figure 4: Startpoint and endpoint assignment of the running example XML document.**



**Figure 5: Region representation of the example document in Figure 4.**

```
<article><fno>fno</fno><fm><til>Til</til>
<au>Author</au></fm><bdy><abs>Abs</abs>
<sec>Sec</sec></bdy></article>
```

A text region  $a$  can be identified by its starting point  $s_a$  and ending point  $e_a$  within the entire linearized string. Figure 4 visualizes the start point and end point numbering for the example XML document and we can see, for example, that the *bdy*-region can be identified with the closed interval  $[12..37]$ . We have visualized the complete region set of the example XML document in Figure 5.

The index terms present in the content text of the XML document are encoded as text regions with a length of 1 position and stored in a separate relation, the word index  $\mathcal{W}$ .

For completeness, we give the formal definition for an XML data region as used in our system below.

**Definition 3.1.** An XML data region  $\mathbf{r}$  is defined as a five-tuple  $(o_r, s_r, e_r, t_r, p_r)$ , where:

- $o_r \in \mathbf{oid}$  denotes a unique node identifier for region  $r$ ;
- $s_r$  and  $e_r$  represent the start and end positions of the text region  $\mathbf{r}$  respectively;
- $t_r \in \mathbf{string}$  is the nodename of region  $\mathbf{r}$ ;
- $p_r \in \mathbf{oid}$  is the identifier of the parent region of region  $\mathbf{r}$ .

We also define the node index  $\mathcal{N}$  as the projection of  $o_r$  over the set of all indexed regions.

Operator	Definition
$contains(a, b, NP)$	$true \iff s_b \geq s_a \wedge e_b \leq e_a$
$contains(a, b, P)$	$true \iff s_b > s_a \wedge e_b < e_a$
$contained(a, b)$	$contains(b, a)$
$length(a)$	$e_a - s_a + 1$
$textlength(a)$	$ contains(a, \mathcal{W})  - 1$
$contains(A, B, NP)$	$\{(o_a, o_b) \mid a \leftarrow A, b \leftarrow B, contains(a, b, NP)\}$
$contains(A, B, P)$	$\{(o_a, o_b) \mid a \leftarrow A, b \leftarrow B, contains(a, b, P)\}$
$contained(A, B, NP)$	$\{(o_a, o_b) \mid a \leftarrow A, b \leftarrow B, contains(b, a, NP)\}$
$contained(A, B, P)$	$\{(o_a, o_b) \mid a \leftarrow A, b \leftarrow B, contains(b, a, P)\}$
$length(A)$	$\{(o_a, length(a)) \mid a \leftarrow A\}$
$textlength(A)$	$\{(o_a, textlength(a)) \mid a \leftarrow A\}$

**Table 1: Region and region set operators (the set operators are given in comprehension syntax [5]). Note that  $(s_a, e_a)$  and  $(s_b, e_b)$  denote the starting and ending positions of regions  $a$  and region  $b$  respectively. The markers  $P$  and  $NP$  stand for proper and non-proper respectively.**

## 3.2 Document Manipulation

The linearized string view enabled us to use theory and practice from the area of text region algebras [6, 7, 8, 12, 14, 13] for selection and manipulation of (sets of) text regions.

Table 1 summarizes the operators we implemented for our system. The *containment* operation  $contains(a, b)$  determines if the region  $a$  contains some other region  $b$ , and its inverse is the *contained*-operation. Finally,  $length(a)$  gives the length of a text region (including markup tags) and  $textlength(a)$  gives the length of the content text present in region  $a$ . The operators are also defined for sets ( $A$  and  $B$  in Table 1).

The use of text regions shows us efficient implementation possibilities. Generating the transitive closure of a region  $a$  requires a contains-operation, a selection on the word index  $\mathcal{W}$  with lower and upper bounds  $s_a$  and  $e_a$ . Generating the original XML structure of a (sub)document  $d$  encompasses:

- a containment operation on the node index  $\mathcal{N}$  to retrieve all descendant nodes of  $d$ :  $desc := contains(d, \mathcal{N}, NP)$ . The containment is non-proper since we want the root element  $d$  in the set as well;
- a (proper) containment operation on the word index  $\mathcal{W}$  to retrieve all context text:  $text := contains(d, \mathcal{W}, P)$ ;
- a union of *desc* and *text*, followed by sorting and some string manipulation for finalization.

Note that the approach outlined in this subsection is similar to the preordering and postordering approach for acceleration of XPath queries, proposed by Grust [9], although we consider Grust's approach a specific instance of general text region algebras, as is ours.

## 4. EXPERIMENTS

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



**Table 2: Experimentation scenarios**

Scenario	Retr. Unit	Dimension(s)
$V_1$	$tr(\text{'article'})$	topicality
$V_2$	$tr(\text{'*'})$	topicality
$V_3$	$tr(\text{'*'})$	topicality, quantity

all subtrees or transitive closures 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 500 (representing a user with a preference for components of 500 words). We summarized our experimentation scenarios in Table 2.

At the time of finishing this paper, we did not have complete relevance assessments or evaluation results of the experimentation scenarios described. We created some scripts to convert INEX submissions and (the available) INEX relevance judgements to TREC format. We then used the *trec\_eval* evaluation software.

Processing of INEX relevance judgements with *trec\_eval* required a conversion of the four-graded relevance scale (0 for irrelevant to 3 for highly relevant) to a binary relevance scale. This conversion was done in two ways:

**Binary Relevance Assignment  $B_1$**  All components judged with 1, 2 or 3 were regarded as relevant, all components judged with 0 were regarded as irrelevant.

**Binary Relevance Assignment  $B_2$**  All components judged with 2 or 3 were regarded as relevant, all components judged with 0 and 1 were regarded as irrelevant.

We will refer to relevance assignments as  $B_1$  and  $B_2$  in the rest of this discussion. Also note that we only focus on the content-only topics, and those content-only topics that have been assessed<sup>1</sup>.

An initial evaluation with *trec\_eval* showed us poor results, for all three runs and both relevance assignments, with mean average precisions below 0.1. A more detailed analysis of the evaluation results for all three runs showed an observation that triggered our curiosity: for many topics, far more relevant components exist than the result set size could fit.

Traditional retrieval collections constructed in the Cranfield tradition contain a small amount of relevant documents in the collection (at least, the amount of relevant documents per query is much smaller than the resultset size). This small amount of relevant documents enables a ‘perfect’ retrieval system to retrieve all relevant documents in the resultset, which in turn enables the calculation of system (and run) comparable recall-precision graphs.

However, with a large discrepancy between number of relevant documents and the result set size, higher percentages of recall

<sup>1</sup>At the time of writing the notebook paper, the assessed content-only topics were: 31, 34, 36, 37, 38, 41, 42, 43, 45, 46, 47, 51, 52 and 58.

**Table 3: Evaluation results; mean average precision (MAP) of the 14 assessed content-only topics. For scenario  $V_1$  we evaluated against relevant article components only instead of all relevant components.**

Scenario	Relevance Assignment	MAP
$V_1$	$B_1$	0.2979
$V_1$	$B_2$	0.3363
$V_2$	$B_1$	0.0279
$V_2$	$B_2$	0.0174
$V_3$	$B_1$	0.0613
$V_3$	$B_2$	0.0627

could never be reached, causing meaningless recall-precision curves. To illustrate this effect further, consider the following example. Let us assume we have a query that has 1000 relevant documents in the collection. The result set size is set at 100 documents. When we determine a precision-recall graph for this query, we will see that after 0.1 recall we get precision values which say nothing meaningful about the performance of a system. Even if all results in the result set are relevant (we will reach maximum precision at 0.1 recall), the precision values at higher levels of recall will always decrease, simply because no more documents have been retrieved (resulting in an average precision of 33% instead of 100%).

For fair evaluation, we can follow two possible paths. Firstly, we can use a measure that is invariant wrt the difference between 1) the number of relevant documents in the collection (for a given topic) and 2) the resultset size. A possibility would be to use precision at various document cutoff levels, instead of precision at various levels of recall [11]. In any case, the numbers for runs 2 and 3 are not very meaningful but we include these for completeness.

Secondly, we can look for ways to transform the runs to comparable formats. For example, to get fair evaluation results for runs 2 and 3 we would need to return a resultset of a larger size, at least as large as the number of relevant documents. However, this conversion renders the evaluations for runs 2 and 3 incomparable to the evaluation for run 1. We can also convert the result set of the article run 1 to a result set that contains all descendants of the returned original articles. In the context of all document components being independent documents, returning an article is the same as marking all the article descendants as possibly relevant.

We re-evaluated our first run by using only relevant articles from the assessments instead of using all the relevant components. Since our first run focused on article retrieval and we fixed the retrieval unit at the article-component, a fair evaluation of this run can be made only when the set of relevant documents contains article components only. This re-evaluation gave us results which we would expect from the statistical language model based on reported behavior on other collections. For runs 2 and 3 however, we used all relevant document components for evaluation since we did not fix the retrieval unit.

The last evaluation we performed was an initial evaluation of coverage. We used *trec\_eval* for this evaluation as well. For creating TREC formatted judgements results, we marked each exact element as ‘relevant’ and each small, large and non-coverage el-

**Table 4: Evaluation results; mean average EC-precision of the 14 assessed content-only topics. For scenario  $V_1$  we evaluated against exact coverage article components only instead of all exact coverage components.**

Scenario	Mean Average EC-Precision
$V_1$	0.1975
$V_2$	0.0163
$V_3$	0.0426

ement as ‘irrelevant’. The measurement `trec_eval` gives us then can be considered as an Exact-component precision (abbreviated EC-precision in the rest of the discussion). Again, evaluation showed us poor results, which are summarized in Table 4.

We had expected many article components would have been deemed too large. Examination of some of the assessed content-only topics showed us otherwise; in some cases the percentage of articles being assessed as exact was nearly 50% of the total number of exact components. We also examined some of the assessments of these ‘suspicious’ topics and discovered a probable judgement disagreement in two of the topics. Since we have no numeric details on inter-assessor (dis)agreement, for the simple reason that the topics have been assessed by a single person for the first time, we cannot further extend our discussion on coverage at this time.

## 5. 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. In hindsight, we have not looked deeply into the possibilities for integrating structure, apart from describing a simple model with which structural properties of documents can be injected into the retrieval process.

Our evaluation of the retrieval results is a rather ad-hoc approach and generic conclusions cannot be drawn. However, the evaluation exercise showed us that evaluation measures for structured document collections needs further defining and discussion.

Future work includes more extensive experimentation with the model described in this paper, especially in the area of relevance feedback.

## 6. REFERENCES

- [1] C.L. Barry. User-defined Relevance Criteria: An Exploratory Study. *Journal of the American Society for Information Science*, 45(3):149–159, 1994.
- [2] N.J. Belkin, R.N. Oddy, and H.M. Brooks. ASK for Information Retrieval: Part 1. Background and Theory. *Journal of Documentation*, 38(2):61–71, 1982.
- [3] N.J. Belkin, R.N. Oddy, and H.M. Brooks. ASK for Information Retrieval: Part 2. Results of a Design Study. *Journal of Documentation*, 38(3):145–164, 1982.
- [4] H.W. Bruce. A Cognitive View of the Situational Dynamism of User-centered Relevance Estimation. *Journal of the American Society for Information Science*, 45(3):142–148, 1994.
- [5] P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. Comprehension Syntax. In *SIGMOD Record*, 1994.
- [6] 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.
- [7] 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.
- [8] 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.
- [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] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, Twente, The Netherlands, 2000.
- [11] D. Hull. Using Statistical Testing in Evaluation of Retrieval Experiments. In *Proceedings of the 16th ACM SIGIR Conference on Research and Development in Information Retrieval*, 1993.
- [12] J. Jaakkola and P. Kilpelainen. Nested Text-Region Algebra. Technical Report C-1999-2, Department of Computer Science, University of Helsinki, 1999.
- [13] P. Kilpelainen and H. Mannila. Retrieval from Hierarchical Texts by Partial Patterns. In *Proceedings of the 16th ACM SIGIR International Conference on Research and Development in Information Retrieval*, 1993.
- [14] R.C. Miller. *Light-Weight Structured Text Processing*. PhD thesis, Computer Science Department, Carnegie-Mellon University, 2002.
- [15] S. Mizarro. How Many Relevances in Information Retrieval? *Interacting With Computers*, 10(3):305–322, 1998.
- [16] A.R. Schmidt, M.L. Kersten, M.A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *International Workshop on the Web and Databases (in conjunction with ACM SIGMOD)*, pages 47–52, 2000.
- [17] T. Westerveld, W. Kraaij, and D. Hiemstra. Retrieving Web Pages using Content, Links, URLs and Anchors. In *NIST Special Publication 500-250, The 10th TREC Retrieval Conference (TREC 2001)*, 2001.

# A database approach to INEX

– draft –

Djoerd Hiemstra  
University of Twente,  
Centre for Telematics and Information Technology  
P.O. Box 217, 7500 AE Enschede, The Netherlands  
hiemstra@cs.utwente.nl

## ABSTRACT

This paper describes a first prototype system for content-based retrieval from XML data. The system's design supports both XPath queries and complex information retrieval queries.

**Keywords:** Databases, XML, Information Retrieval, Language Models

## 1. INTRODUCTION

This paper describes a number of fundamental ideas and starting points for building a system that seamlessly integrates data retrieval and information retrieval (IR) functionality into a database system. We describe a first prototype system that is developed according to these ideas and starting points and report on experimental results of the system on the INEX collection. The current prototype system only contains a very small part of the functionality that we envision for future systems, but in the upcoming years we will build a number of such prototype systems in the CIRQUID project (Complex Information Retrieval Queries in a Database) project that is funded by the Netherlands Organisation for Scientific Research (NWO).

The CIRQUID project bridges the gap between structured query capabilities of XML query languages and relevance-oriented querying. Current techniques for XML querying, originating from the database field, do not support relevance-oriented querying. On the other hand, techniques for ranking documents, originating from the information retrieval field, typically do not take document structure into account. Ranking is of the utmost importance if large collections are queried, to assist the user in finding the most relevant documents in a retrieved set.

The paper is organised as follows: Section 2 describes our database approach to relevance-oriented querying from XML

documents. Section 3 reports the experimental results of our first prototype system.

## 2. A MULTI-MODEL DATABASE APPROACH

A three level design of DBMSs – distinguishing a conceptual, a logical, and a physical level – provides the best opportunity for balancing flexibility and efficiency. In our approach, we take the three level architecture to its extreme. Not only do we guarantee logical and physical data independence between the three levels, we also map the conceptual data model used by the end users to a physical implementation *using different data models at different levels of the database architecture*: the so-called “multi-model” database approach [25].

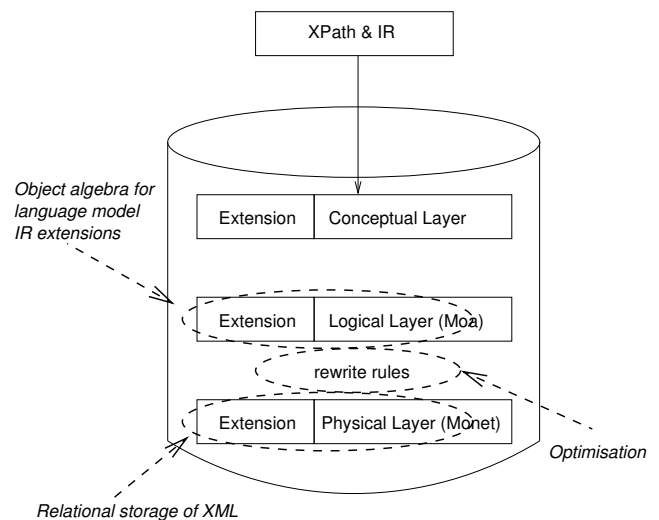


Figure 1: Database internals

Figure 1 shows a graphical representation of the approach. At the logical level, language models will be used to develop information retrieval primitives as a logical algebra. The physical level provides a relational storage of the XML data, including fast index structures. A new approach to query optimisation deals with the complex queries that combine structure and content at the logical level. In the following three subsections we will present some of the ideas and starting points for developing the three levels of the multi-model database approach.

## 2.1 All of XPath and modern IR queries

The conceptual level should support XML and IR queries. Our objective is to build a system that supports “all of XML and all of IR”.

For XML, standards are currently emerging, and it seems reasonable to support the XPath standard for our “traditional database queries”. Practically, this means that our system should contain a complete representation of the XML data, and that the system is able to reproduce (parts of) the data as the result of the query. For XPath we refer to [2].

Unlike the database and XML communities, which have developed some well-accepted standards in the past 30 years, the information retrieval community does not have any comparable standard query language or retrieval model. If we look at some practical systems however, e.g. internet search engines like Google and AltaVista, or online search services as provided by e.g. Dialog and LexisNexis, it turns out that there is much overlap in the kind of functionality they provide.

- |  |
|--|
| <ol style="list-style-type: none"><li>1. IT magazines</li><li>2. +IT magazine* -MSDOS</li><li>3. "IT magazines"</li><li>4. IT NEAR magazines</li><li>5. (IT OR computer) (books OR magazines OR journals)</li><li>6. XML[0.9] IR[0.1] title:INEX site:utwente.nl</li></ol> |
|--|

Figure 2: Examples of complex IR queries

Figure 2 gives some example queries from these systems. The first query is a simple “query by example”: retrieve a ranked list of documents about IT magazines. The second query shows the use of a mandatory term operator ‘+’, stating that the retrieved document *must* contain the word IT,<sup>1</sup> a wild card operator ‘\*’ stating that the document might match “magazine”, but also “magazines” or “magazined” and a the ‘-’ operator stating that we do not prefer IT magazines about MSDOS. The third and fourth query searches for documents in which “IT” and “magazines” occur respectively adjacent or near to each other. The fifth query shows the use of the ‘OR’ operator, stating that the system might retrieve documents about “IT magazines”, “computer magazines”, “IT journals”, “IT books”, etc. The sixth and last query shows the use of structural information, very much like the kind of functionality that is provided by XPath; so “title:INEX” means that the title of the document should contain the word INEX. The last query also shows additional term weighting, stating that the user finds XML much more important than IR.

Practically, these example suggest that at the logical level, our system should support algebraic constructs for proximity of terms, mandatory terms, a logical OR, term weighting, etc. To support proximity operators the system should at least store term position information somehow at the physical level.

<sup>1</sup>Note that most retrieval systems do not distinguish upper case from lower case, and confuse the acronym “IT” with the very common word “it”.

## 2.2 Moa and Language Models

Parts of a prototype multi-model database system have already been developed with the extensible object algebra Moa [15] as the logical layer. An open question in this setup is how Moa, which provides a highly structured nested object model with sets and tuples, can be adapted to managing semi-structured data. In this paper we will not get into Moa, but direct our attention to the language modelling approach to information retrieval as proposed in [10, 18] to guide the definition of the logical layer of our system.

The basic idea behind the language modelling approach to information retrieval is that we assign to each XML element  $X$  the probability that the element is relevant, given the query  $Q = q_1, \dots, q_n$ . Using Bayes’ rule we can rewrite that as follows.

$$P(X|q_1, q_2, \dots, q_n) = \frac{P(q_1, q_2, \dots, q_n|X)P(X)}{P(q_1, q_2, \dots, q_n)} \quad (1)$$

Note that the denominator on the right hand side does not depend on the XML element  $X$ . It might therefore be ignored when a ranking is needed. The prior  $P(X)$  however, should only be ignored if we assume a uniform prior, that is, if we assume that all elements are equally likely to be relevant in absence of a query. Some non-content information, e.g. the number of accesses by other users to an XML element, or e.g. the length of an XML element, might be used to determine  $P(X)$ .

Let’s turn our attention to  $P(q_1, q_2, \dots, q_n|X)$ . The use of probability theory might here be justified by modelling the process of generating a query  $Q$  given an XML element as a random process. If we assume that this page in the INEX proceedings is an XML element in the data, one might imagine picking a word at random from the page by pointing at the page with closed eyes. Such a process would define a probability  $P(q|X)$  for each term  $q$ , which might simply be calculated by the number of times a word occurs on this page, divided by the total number of words on the page. Similar generative probabilistic models have been used successfully in speech recognition systems [20], for which they are called “language models”.

The mechanism above suggests that terms that do not occur in an XML element are assigned zero probability. However the fact that a term is never observed does not mean that this term is never entered in a query for which the XML element is relevant. The problem that events which are not observed in the data might still be reasonable in a new setting, is called the sparse data problem in the world of language models [16]. Zero probabilities should therefore be avoided. A standard solution to the sparse data problem is to interpolate the model  $P(q|X)$  with a background model  $P(q)$  which assigns a non-zero probability to each query term. If we additionally assume that query terms are independent given  $X$ , then:

$$P(q_1, q_2, \dots, q_n|X) = \prod_{i=1}^n \left( (1-\lambda)P(q_i) + \lambda P(q_i|X) \right) \quad (2)$$

Equation 2 defines our basic language model if we assume that each term is generated independently from previous terms given the relevant document. Here,  $\lambda$  is an unknown mixture parameter, which might be set using e.g. relevance feedback of the user. Ideally, we would like to train the probability of an unimportant term  $P(q_i)$  on a large corpus of queries. In practice however, we will use the document collection to define these probabilities. By some simple rewriting, it can be shown that Equation 2 can be implemented as a vector space weighting algorithm [11].

Why would we prefer the use of language models over the use of e.g. a vector space model with some *tf.idf* weighting algorithm as in [21]? The reason is the following: our generative query language model gives a nice intuitive explanation of *tf.idf* weighting algorithms by means of calculating the probability of picking at random, one at a time, the query terms from an XML element. We might extend this simply by any other generating process to model complex information retrieval queries in a theoretically sound way that is not provided by a vector space approach. For instance, we might calculate the probability of sampling either “magazines” or “books” or “journals” from the XML document by summing the probabilities  $P(\text{magazines}|X)$ ,  $P(\text{journals}|X)$ , and  $P(\text{books}|X)$ . So, Query 5 from Figure 2 would assign the following probability to each XML element (ignoring for a moment the prior  $P(X)$  and the linear interpolation with the background model  $P(q_i)$  for simplification of the example).

$$P(\text{Query 5}) = (P(\text{IT}|X) + P(\text{computer}|X)) \cdot (P(\text{books}|X) + P(\text{journals}|X) + P(\text{magazines}|X))$$

Interestingly, a similar approach was proposed in 1960 by Maron and Kuhns [17]. In a time when manual indexing was still guiding the field, they suggested that an indexer, which runs through the various possible index terms  $q$  that possibly apply to a document, might assign a probability  $P(q|X)$  to a term given a document instead of making a yes/no decision. The language modelling equivalent of ‘disjunction’ and ‘conjunction’ (i.e. ‘AND’ and ‘OR’ operators) is motivated by adding a so-called translation model to the basic model [1, 14, 26].

In CIRQUID we will explore language modelling approaches that model all structured queries in Figure 2. The interested reader is referred to [18, 24] for so-called bigram models for proximity queries, and [13] for mandatory terms.

### 2.3 Relational storage

At the physical level, we will use the ‘good-old’ relational model for storage of the data. In order to combine XPath and information retrieval functionality, we somehow have to combine relational data representations of XML as described in e.g. [5, 23], and relational representations of information retrieval indexing structures as described by e.g. [3, 8, 25]. Our starting point for the relational storage of the XML data is that it should not critically depend on the existence of a schema or DTD, and that it should be possible to reconstruct the XML data completely. Our starting point for the storage of information retrieval indexing structures is that it should provide the ‘traditional information retrieval’ functionality as well as term position information to support proximity queries.

### Related work on XML storage

A standard approach to storing hierarchical or nested data, with or without a schema, is to store each “instance node” separately in a relational table. This is illustrated in Figure 3, 4 and 5. Figure 4 shows a tree representation of the XML instance of Figure 3. Each node in the tree is assigned a node identifier “id”.

```
<article>
  <au><fnm>Boudewijn</fnm><snm>Büch</snm></au>
  <atl>Kleine blonde dood</atl>
  <bdy>
    <p>Een schrijver ontmoet een oude bekende.</p>
    <p>Er ontstaat een liefdesrelatie.</p>
  </bdy>
</article>
```

Figure 3: Example XML data

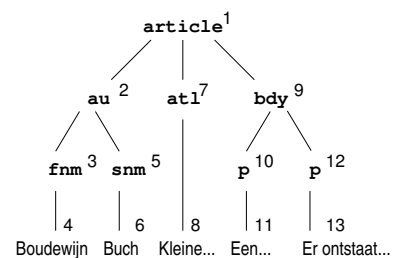


Figure 4: Tree representation of the data

Now for each node, we might store its id and the id of its parent as shown in Figure 5. One can think of numerous alternative ways to assign the ids to the instance nodes (in this example they were assigned in pre-order). Similarly, one can think of many relational schemas that support this basic idea, by fragmenting the tables of Figure 5 in various ways. In previous work, we used a full fragmentation in binary relational tables [15] which provides efficient support for XML querying [23].

tags			pcdata		
id	parent	tag_name	id	parent	string
1	0	article	4	3	Boudewijn
2	1	au	6	5	Büch
3	2	fnm	8	7	Kleine blonde ...
5	2	snm	11	10	Een schrijver ...
7	1	atl	13	12	Er ontstaat ...
9	1	bdy			
10	9	p			
12	9	p			

Figure 5: Example relational storage of XML data

### Related work on the storage of IR indexes

A standard approach to the relational storage of information retrieval indexes uses two tables. One table stores the document term statistics, i.e. for each document-term pair some statistics related to the number of times the term occurs in the document. A second table stores the global term statistics, i.e. for each term some statistics related to the total

number of times that a term occurs in the entire collection. In traditional systems that use a *tf.idf* term weighting algorithm, the first table contains the *tf*'s (term frequencies) and the second table contains the *df*'s (document frequencies). In the language modelling approach we might store  $P(q|X)$  in the first table and  $P(q)$  in the second.

local_stats			global_stats	
word	id	$P(\text{word} \text{id})$	word	$P(\text{word})$
aardvark	43	0.007	aardvark	0.00001
after	3	0.09	after	0.0345
after	42	0.11	affect	0.00055
after	78	0.015	ambient	0.0000001
after	980	0.067	an	0.107
affect	321	0.2	:	
ambient	761	0.0001	:	
:			:	
bekende	1	0.031	:	
blonde	1	0.031	:	
boudewijn	1	0.031	:	
:			:	

Figure 6: Example relational storage of an IR index

In [4, 8, 25], *id* refers to a document identifier. For XML data it should refer to the node id of the XML element as shown in Figure 4 and 5. A fundamental problem with this approach is the following. If we include all word-id pairs in the table *local\_stats* of Figure 6, then each word in the data will occur as often as the average depth of the XML data. For INEX, the average depth is about 7, so our information retrieval index would be 7 times as big as the “regular” index that only indexes traditional documents (e.g. web pages).

A solution to this problem is to let the database administrator choose the nodes that need to be indexed, the so-called “indexing nodes” [6, 27], however, this will restrict the functionality such that queries like `//*[computational biology]` (pseudo “XPath+IR” for any element about “computational biology”) would be impossible, or only possible by inefficient linear scans over all string fields in the PCDDATA table of Figure 5.

An alternative solution to this problem is to only store all leaf nodes of the XML data in *local\_stats* as suggested in [7]. This means that queries like `//article[computational biology]` (any *article* element about “computational biology”) would need a number of repeated joins with the table *tags* of Figure 5 in order to determine the id of the article node that contains the query terms

Instead storing the tag name, one could store the complete path in Figure 5. This would solve only part of the problem, because it would require a special purpose implementation of regular path matches on attributes.

Figure 7 shows the typical information retrieval ranking algorithm expressed in SQL to give the reader a flavour of how the system uses the tables of Figure 6 at run time. In practice, we will not use SQL at the physical level. The function *f* in the algorithm might be any *tf.idf* formula. In case of the language modelling approach, *f* might be defined as  $\log(1 + P(q|X)/P(q))$  [11].

```
SELECT id, SUM(f(local_stats.p, global_stats.p)) AS s
FROM local_stats, global_stats
WHERE local_stats.word = global_stats.word
AND (local_stats.word = 'computational'
OR local_stats.word = 'biology')
GROUP BY id
ORDER BY s DESC
```

Figure 7: Traditional IR query in pseudo SQL

### A first prototype

For our first prototype we implemented the XML storage scheme proposed by Grust [9]. Grust suggests to assign two identifiers to each instance node: one id is assigned in pre-order, and the other in post-order. These ids replace the explicit parent-child relations as described in the previous paragraphs.<sup>2</sup> The pre and post assignment of XML element ids provides elegant support for processing XPath queries [7].

```
<article>1
<au>2<fnm>3Boudewijn4</fnm>5<snm>6Büch7</snm>8</au>9
<atl>10Kleine11 blonde12 dood13</atl>14
<bdy>15
<p>16Een17 schrijver ontmoet een oude bekende.</p>
<p>Er ontstaat een liefdesrelatie.</p>
</bdy>
</article>
```

Figure 8: Example XML document: assigning ids

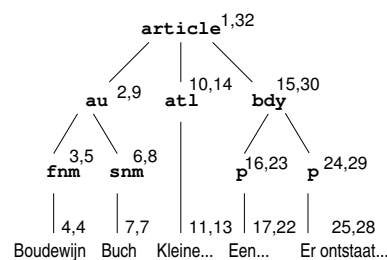


Figure 9: Tree representation: assigning ids

Note that pre and post assignment can be done almost trivially in XML by keeping track of the order of respectively the opening and closing tags as shown in Figure 8 and 9. Both figures also show that position information is assigned to each word in the document. These positions will be used in our term position index. This leads to the relational storage

<sup>2</sup>Actually, [9] store the id of the parent as well. Similarly, in [23] a field is added to keep track of the order of XML elements, here we emphasise different view points.

of XML data as shown in Figure 10 and the relational storage of the information retrieval positional index as shown in Figure 11.

tags2			pcdata2		
pre	post	tag_name	id	parent	string
1	32	article	4	4	Boudewijn
2	9	au	7	7	Büch
3	5	fnm	11	13	Kleine blonde...
6	8	snm	17	22	Een schrijver ...
10	14	atl	25	28	Er ontstaat ...
15	30	bdy			
16	23	p			
24	29	p			

Figure 10: Relational storage of XML data

position_index		global_stats	
word	position	word	$P(\text{word})$
bekende	22	bekende	0.00321
blonde	12	blonde	0.00013
boudewijn	4	boudewijn	0.00004
büch	7	büch	0.00001
een	17	een	0.0991
een	20	er	0.0145
een	27	:	
er	25		
kleine	11		
:			

Figure 11: Relational storage of the IR positional index

Note that exactly one ‘join’ (on the condition: `position > pre` and `position < post`, counting the positions) will give us a table that is similar to `local_stats` in Figure 6. Figure 12 expresses this in SQL.

```
CREATE VIEW local_stats2 AS
SELECT word, pre, post,
CAST(COUNT(position) AS float) / (post - pre) AS p
FROM position_index, tags2
WHERE position > pre
AND position < post
GROUP BY word, pre, post
```

Figure 12: Combining term information and the structured information in pseudo SQL

Also note that, unlike the approaches in [27, 7], we are not interested in the total number of times a term occurs in a certain XML element type (that is, the so-called ‘document frequency’ of the term). The language modelling approach suggests that  $P(q)$  is the probability of a term in “general query English”: It should be the same for all queries. Furthermore, to avoid the sparse data problem, it should be estimated on as much data as possible. In our case,  $P(q)$  is defined by the total number of occurrences of  $q$  in the entire INEX collection, divided by the total number of term occurrences in INEX (i.e. the “collection length” measured in the number of words).

## 2.4 Optimisation

As an example of a logical optimisation step, let’s have a look at the fifth query of Figure 2 again. For the second part of Query 5,  $P(\text{books OR journals OR magazines}|X)$  is defined in Section 2.2 as:

$$P(\text{books}|X) + P(\text{journals}|X) + P(\text{magazines}|X)$$

Remember that each  $P(q|X)$  is defined by the ‘join’ of Figure 12. This suggests that we have to do the ‘join’ for each of the words *books*, *journals* and *magazines*, and then group them by the XML element id, adding the probabilities. In [12] it is shown that a more efficient approach would be to first determine the number of occurrences of either (*books OR journals OR magazines*) and then compute the probability by dividing by the length of the XML element. So, we could first do a selection of (*books OR journals OR magazines*) on the position index, and then do the ‘join’ with the tags table. This way we avoid two of the three joins. A similar optimisation step is in general not possible in extended Boolean models [22] and fuzzy set models [19].

To understand what is happening here, note that each occurrence of (*books OR journals OR magazines*) actually has its own position. At any place in the XML data where either *books*, or *journals*, or *magazines* occurs, we actually know its position. We cannot do a similar optimisation for ‘AND’ queries (Note that all queries of Figure 2, except for Query 5, are implicit ‘AND’ queries), that is, the words *books*, *journals*, and *magazines* occur nowhere in the data on exactly the same position, for the simple reason that each position contains exactly one word.

The above example shows a simple, almost trivial, optimisation step. A modern database query optimiser should be able to reason over queries that contain clauses over data structures that are typically implemented in different extensions of the DBMS. Current, state-of-the-art optimiser technology can deal with extensions in isolation. In future work, we will design an inter-object optimiser layer that is able to bridge the typical orthogonality of database extensions. At the logical level, the query optimiser will be extended to handle interacting extensions, including e.g. extensions for other media.

## 3. EVALUATION RESULTS

The evaluation results were not available at the time when this paper was written. They are presented at the workshop and they will be included in the final version of this paper.

## Acknowledgements

The research presented in this paper was funded in part by the Netherlands Organisation for Scientific Research (NWO).

## 4. REFERENCES

- [1] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *Proceedings of the 22nd ACM Conference on Research and Development in Information Retrieval (SIGIR’99)*, pages 222–229, 1999.
- [2] A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Simeon. XML

- Path language (XPath) 2.0. Technical report, World Wide Web Consortium, 2002.  
<http://www.w3.org/TR/xpath20/>
- [3] H.E. Blok. *Database Optimization Aspects for Information Retrieval*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, 2002.
  - [4] H.E. Blok, D. Hiemstra, S. Choenni, F.M.G. de Jong, H.M. Blanken, and P.M.G. Apers. Predicting the cost-quality trade-off for information retrieval queries: Facilitating database design and query optimisation. In *Proceedings of the 10th International Conference on Information and Knowledge Management, CIKM'01*, pages 207–214, 2001.
  - [5] D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. In *Proceedings of the VLDB'99*, pages 105–110, 2001.
  - [6] N. Fuhr and K. Grossjohann. XIRQL: A query language for information retrieval in XML. In *Proceedings of the 24th ACM Conference on Research and Development in Information Retrieval (SIGIR'01)*, pages 172–180, 2001.
  - [7] T. Grabs. Generating vector spaces on-the-fly for flexible XML retrieval. In *Proceedings of the SIGIR workshop on XML and Information Retrieval*, pages 4–13, 2002.
  - [8] D.A. Grossman, O. Frieder, D.O. Holmes, and D.C. Roberts. Integrating Structured Data and Text: A Relational Approach. *Journal of the American Society of Information Science*, 48(2):122–132, 1997.
  - [9] T. Grust, Accelerating XPath location steps *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 109–120, 2002.
  - [10] D. Hiemstra. A linguistically motivated probabilistic model of information retrieval. In *Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 569–584, 1998.
  - [11] D. Hiemstra. A probabilistic justification for using tf.idf term weighting in information retrieval. *International Journal on Digital Libraries*, 3(2):131–139, 2000.
  - [12] D. Hiemstra. *Using language models for information retrieval*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, 2001.
  - [13] D. Hiemstra. Term-specific smoothing for the language modeling approach to information retrieval: The importance of a query term. In *Proceedings of the 25th ACM Conference on Research and Development in Information Retrieval (SIGIR'02)*, pages 35–41, 2002.
  - [14] D. Hiemstra and F.M.G. de Jong. Disambiguation strategies for cross-language information retrieval. In *Proceedings of the third European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 274–293, 1999.
  - [15] M. van Keulen, J. Vonk, A.P. de Vries, J. Flokstra, and H.E. Blok. Moa: extensibility and efficiency in querying nested data. Technical report 02-19, Centre for Telematics and Information Technology, 2002.
  - [16] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
  - [17] M.E. Maron and J.L. Kuhns. On relevance, probabilistic indexing and information retrieval. *Journal of the Association for Computing Machinery*, 7:216–244, 1960.
  - [18] D.R.H. Miller, T. Leek, and R.M. Schwartz. A hidden Markov model information retrieval system. In *Proceedings of the 22nd ACM Conference on Research and Development in Information Retrieval (SIGIR'99)*, pages 214–221, 1999.
  - [19] C.P. Paice. Soft evaluation of Boolean search queries in information retrieval systems. *Information Technology: Research and Development*, 3(1):33–42, 1984.
  - [20] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In A. Waibel and K.F. Lee, editors, *Readings in speech recognition*, pages 267–296. Morgan Kaufmann, 1990.
  - [21] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.
  - [22] G. Salton, E.A. Fox, and H. Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.
  - [23] A. R. Schmidt, M. L. Kersten, M. A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents (Extended Version). In *The World Wide Web and Databases - Selected Papers of WebDB 2000*, Springer-Verlag, pages 137–150, 2000.
  - [24] F. Song and W.B. Croft. A general language model for information retrieval. In *Proceedings of the 8th International Conference on Information and Knowledge Management, CIKM'99*, pages 316–321, 1999.
  - [25] A.P. de Vries. *Content and Multimedia Database Management Systems*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, 1999.
  - [26] J. Xu, R. Weischedel, and C. Nguyen. Evaluating a probabilistic model for cross-lingual information retrieval. In *Proceedings of the 24th ACM Conference on Research and Development in Information Retrieval (SIGIR'01)*, pages 105–110, 2001.
  - [27] R. van Zwol. *Modelling and searching web-based document collections*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, 2002.



# The Xircus Search Engine\*

Holger Meyer<sup>†</sup>  
University of Rostock  
Computer Science Dept  
Database Research Group  
18051 Rostock, Germany  
www.xircus.de

## ABSTRACT

Nowadays, XML is the document model in favor for both document- and data-centric web applications. Its influence in other, more traditional projects grows as the web and associated techniques become the de-facto standard in user interfaces in such systems.

We present an XML-sensitive search engine (Xircus) suited for processing semi-structured queries over large collections of XML documents. Xircus is based on state of the art information retrieval techniques. It is a testbed for research in query processing for XML and semi-structured data in general.

## Categories and Subject Descriptors

I.7.3 [Document and Text Processing]: Index Generation; H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing; H.3.7 [Digital Libraries]: Systems issues

## General Terms

XML-sensitive Information Retrieval

## 1. INTRODUCTION

Traditional search engines are build upon classical information retrieval methods. Even though they are enhanced by evaluating the hyper-link structure of web sites, there is little effort made in exploiting the document structure itself. Newly build XML-sensitive search engines should rely more on the XML structure and facilitate path expression and structured queries based on a type system.

---

\*Xircus is an acronym for XML-based Indexing, Ranking, and Classification Techniques for Customized Search Engines

<sup>†</sup>hm@ieee.org

The application of such XML-sensitive search engines is manifold: digital libraries, (web) content management, XML-enabled databases, and many other web-based software projects.

Beside that, there are two reasons why we started the Xircus project.

- In the first place, we wanted an XML search engine which implements state of the art techniques for full-text search, XML indexing and query processing.
- Secondly, Xircus should offer a research framework for experimenting with information extraction from XML document collections, path indexing and processing and semi-structured query processing in general, that combines information retrieval with structured, SQL-like queries.

The software architecture should allow for plug in different methods like language specific stemmers, domain specific stopword lists, ontologies and thesauri.

The search engine builds upon several basic data structures. The meta database describes attributes common to XML document collections and properties of documents within these collections. Per collection, there might be different index structures for accelerating the access to documents and their fulltext, XML structure, and often queried fragments.

The Xircus search engine should be easily deployed in a distributed, heterogeneous environment and adopted to different settings.

The paper is organized as follows. Primarily, we give an overview of the objectives of the project and line out the system architecture. Than, a brief discussion of steps involved in index creation and query processing follows. The paper closes with a look at the first prototype and its user interface. Last but not least, some related work and future tasks are discussed.

# An XML Retrieval Model based on Structural Proximities

[Extended Abstract]

Shinjae Yoo  
Department of Digital Contents  
Sejong Cyber University, Korea  
jolly74@korea.com

## ABSTRACT

XML documents differ from general documents in that there are structures. Conventional IR models and structured document retrieval models have not fully exploited the structure information in ranking. Our retrieval model utilizes structural information, esp. proximities, in ranking. In addition, because the complex document structures perplex a novice composing a structured query, we simplify query language but gracefully overcome the problems of lack of expression power.

## 1. INTRODUCTION

XML is a markup language for describing documents and for interchanging data among different systems. Web pages, e-catalogs, e-books or exchanging data may employ XML of which application domains are gradually increasing. In particular, many XML systems have been or are being developed for storing, maintaining, and retrieving XML data or documents in companies or organizations. Although the popularity of XML is increasing, no retrieval model has adapted to XML document. Therefore, we propose new XML document retrieval models which reflect the properties of XML documents.

From an information retrieval point of view, an XML document retains following two properties compared with conventional document and the other structured document :

1. **An XML document keeps structures.**
2. **The structure may be complex.**

When not conforming to DTD(Document Type Definition) or aggregated from several source, XML documents might maintain irregular structure. Conventional retrieval models did not utilize document structures and most of structured document retrieval models made use of document structure not in ranking but only in filtering. Moreover, End-users, in the heterogeneous document collections, may encounter problems in querying irregular structured documents because they do not know exact structure or they are easy to compose bungled queries.

To resolve these problems, new XML document retrieval models may consider following two ideas.

- XML retrieval models can make the most of structured information of XML documents in ranking.
- An end user is able to require easy and short but descriptive structured query language .

In this paper, we assume that we regard the structure of an XML document as not a graph but a tree, which is similar to the DOM (Document Object Model) [1] tree that regard a link as just an attribute.

We propose a model for XML document retrieval evaluated in a bottom-up way, as in [15, 16, 21]. Before a structural query condition is evaluated, content conditions in the query are evaluated in order to reduce the document search space and then we evaluating structural query conditions. In addition, the weight of a node may be affected by the proximity of result nodes and unique query paths.

## 2. PRELIMINARIES

In this section, we describe our models of documents and queries and define proximities in the structured documents.

### 2.1 XML Document Model

Like Figure 1, We model XML documents as a ordered tree. So we can easily build an index of XML documents, but cannot reflect all the information contained in the XML documents; however, we can exploit all the information in XML documents if we are able to analyze and process link information of the XML document during query evaluation.

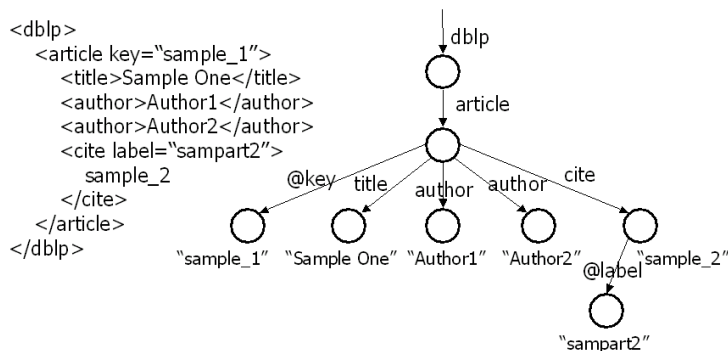


Figure 1: XML document and its document tree

For each node, the label of the incoming edge denotes the name of the element or attribute. For leaf nodes, the value of the node is the corresponding PCDATA value.

## 2.2 XML Querying Model

Because a document modeled as a tree, a query may be modeled as a tree or an SRP (set of regular paths). “Retrieve documents which include ‘contents’ in the title” may be translated into the XPath query `//title = ‘contents’`. If a user asks `//paper[title=‘contents’]/journal=‘journalA’`, this query can be modeled as a tree (see Figure 2-(b)), which contains structural constraints. The information need expressed in Figure 2-(b) is more specific than that of 2-(a). However, it is impossible to formulate complex structural queries when a user is not familiar with the structure of the documents in the collection. The user may prefer simple queries like e.g. in Figure 2-(a), but his real information need would be represented better by Figure 2-(b). For a naïve user, the formulation `//title = ‘contents’ //journal = ‘journalA’` is easier than that of Figure 2-(b). Based on query conditions such as that Figure 2-(a), we developed a query model which assumes independence among query paths. We call this model the SRP (Set of Regular Paths) query model. In this paper, we use the SRP query model to develop our retrieval models. The syntax of SRP is given in to Appendix A. Since the SRP model assumes independence among paths, it does not allow for the formulation of constraints among paths. This problem can be overcome by the model proposed in Section 3.3.

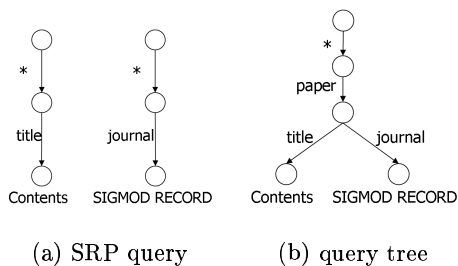


Figure 2: querying models

## 2.3 Proximity

Many retrieval approaches assume that retrieval effectiveness can be improved by considering the proximity of query terms occurring in a document. [5–7, 12, 18, 23, 24] defined proximity operators for considering proximity in retrieval, or incorporated proximity in their weighting formulas. Among these, it was passage retrieval that betrayed the most possibility in ranking documents. Passage retrievals find relevant passages in documents or documents of which weights result from the weight of passages or both passages and document. However, the existing passage retrievals retained a defect combining weights of passages when they were retrieving document which may include fixed size passages or retrieving several passages such as `<section>`, `<chapter>` or `<book>`. This defect, in the XML document retrieval, is more serious than no structured document since an user may retrieve not only a leaf node but also any ancestor node of a leaf node. To improve this defect in the structured documents, we make use of proximities in combining weights

using structural relations between nodes such as vertical ancestor-descendant or horizontal sibling.

In order to define proximity, we need the concepts of distances. In the structured documents, word distances between terms in the leaf node or node distance between nodes may be defined. Node distances can also be classified by horizontal and vertical distances. Horizontal distance (H-distance) is the number of sibling nodes between nodes. From a document point of view, an XML document is an ordered tree. An ordered list among child nodes of a node has a meaning. For instance, two `<paragraph>` nodes which are adjacent are closer semantically than the nodes which are more distant.

A set or list of logical units can be grouped by a different logical unit. The degree of grouping can be measured by vertical distance (V-distance). For example, a series of sentences can be grouped by a paragraph, a series of paragraphs can be grouped by a section and a series of section can be grouped by a chapter. In this case, V-distance between a paragraph and a chapter which includes the paragraph is two. From the data-centric view on XML, H-distance is meaningless but V-distance is meaningful. For instance, when data extracted from a relational database is translated into an XML document, the order among attributes from a table is meaningless. Therefore, in order to cover both the document-centric as well as the data-centric view of XML, a retrieval model should consider both V- and H-distance.

For defining our distance measures, we use the following notations:

- $t_{ij}$  :  $j^{th}$  word in the document  $i$
- $Node(t_{ij})$  : returns the leaf node containing  $t_{ij}$
- $N_{ik}$  :  $k^{th}$  BFS(Breadth First Search) numbered node in document  $i$
- $level(N_{ik})$  : return the level of  $N_{ik}$
- $Parent(N_{ik})$  : return the parent node of  $N_{ik}$
- $maxH(i)$  : maximum number of children of a node in document  $i$

For trees representing XML documents, we define the distance measures shown below (in all of these definitions, if the specified condition is not fulfilled, the distance is considered to be  $\infty$ ).

$$T-dist(t_{ij}, t_{ik}) = |j - k| \text{ if } Node(t_{ij}) = Node(t_{ik}) \quad (1)$$

$$V-dist(N_{ij}, N_{ik}) = |level(N_{ij}) - level(N_{ik})| \quad (2)$$

if  $N_{ij}$  is a descendant  
or ancestor of  $N_{ik}$

$$H-dist(N_{ij}, N_{ik}) = |j - k| \quad (3)$$

if  $Parent(N_{ij}) = Parent(N_{ik})$

$$\mathbb{H}-dist(N_{ij}, N_{ik}) = \frac{maxH(i) - H-dist(N_{ij}, N_{ik})}{maxH(i)} \quad (4)$$

Basically,  $T-dist$  is the same as word distance between words in an unstructured document, but here we apply it to the leaf node of XML documents. For example,  $T-dist(Sample_{i6}, One_{i7})$  is 1 between ‘Sample’ and ‘One’ in a leaf node of the path `“/dblp/article/title”` on Figure 3.  $H-dist$  compares sibling nodes and computes the of BFS numbers. For instance,  $H-dist(N_{i3}, N_{i6})$  between `“/dblp/article/@key”` and `“/dblp/article/author”` is 3 in Figure 3.  $\mathbb{H}-dist$  is a normalized version of  $H-dist$ . The reason for the normalization

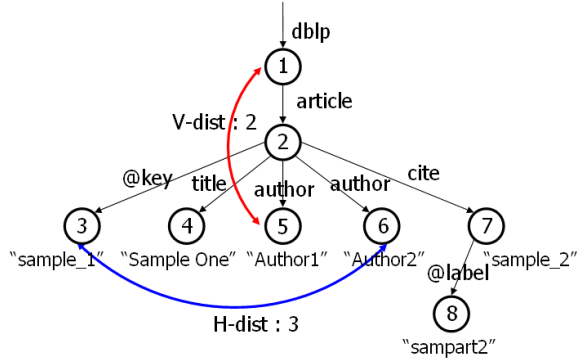


Figure 3: Distances

of  $H-dist$  is to overcome the differences due to document length, especially in horizontal distance.

$V-dist$  is the difference between levels of nodes in a path.  $V-dist(N_{i1}, N_{i5})$  between  $'/dblp'$  and  $'/dblp/article/title'$  is 2. Both  $V-dist$  and  $H-dist$  are new distance measures for XML documents.

A proximity describes the closeness of two nodes, yielding a weight of 1 in case two nodes are identical, and smaller weights for distant nodes. Here, we consider two kinds of proximities,  $H-prox$  (Horizontal proximity) and  $V-prox$  (Vertical proximity), which relates to the corresponding distance measure.

With the following notations

$$\begin{aligned}
 D &: H-dist(N_{ij}, N_{ik}) \\
 C &: \frac{level(N_{ik})}{maxV(i)} \\
 p, v &: \text{constant } (0 \leq p, v \leq 1) \\
 maxV(i) &: \text{the depth of document } i,
 \end{aligned}$$

we define  $H-prox$  as

$$H-prox(N_{ij}, N_{ik}) = (p(v + (1 - v)C))^D \quad (5)$$

$H-prox$  exponentially decreases with growing  $H-dist$   $D$ . This definition is based on the assumption that  $H-prox$  should be close to 0 as the two nodes are far apart. We also considered the product and the sum of  $p$  and  $H-dist$ . However, experiment showed that these functions do not work well, whereas  $p^D$  yielded great improvements in terms of precision. The level factor  $C$  is used to differentiate  $H-dist$  according to the level in the tree.  $v$  is the degree of the effect of  $C$  on  $p$ .

Using the notations

$$\begin{aligned}
 t &: V-prox \text{ factor } (0 < t \leq 1) \\
 V &: V-dist(N_{ij}, N_{ik}) \\
 N_{ij} &: \text{an ancestor or descendant of } N_{ik},
 \end{aligned}$$

we define

$$V-prox(N_{ij}, N_{ik}) = t^V \quad (6)$$

$$V-prox(N_{ij}, N_{ik}) = \min\left(\frac{level(N_{ij})}{level(N_{ik})}, \frac{level(N_{ik})}{level(N_{ij})}\right) \quad (7)$$

Like  $H-prox$ ,  $V-prox$  is a decreasing function according to  $V-dist$ .  $V-prox$  is decreased by  $t$  ratio according to  $V-dist$   $V$ . The normalized version  $V-prox$  also considers the level of the nodes.  $V-prox$  has the same effect as in

terms of assigning proximity to nodes but the proximity is normalized through the level.

### 3. XML DOCUMENT RETRIEVAL MODEL

In this section, we propose a new model for XML document retrieval, based on XML document model, querying model and proximity. Retrieval approach is bottom-up which is similar to [15, 16, 21]. Firstly, content based queries are performed to reduce search space. Then, the nodes which content based queries satisfy are verified by structure based query.

#### 3.1 A leaf node's weight

We defined a leaf node's weight using TF\*IDF weight. More specifically, using the notations

$$\begin{aligned}
 idf_j &: idf \text{ of query term on the } j^{th} \text{ query path} \\
 tf_{N_{ik},j} &: tf \text{ of query term on the } j^{th} \text{ query path} \\
 &\quad \text{in the } N_{ik} \\
 W_{ik} &: \text{the weight of } N_{ik}
 \end{aligned}$$

we compute

$$\begin{aligned}
 W_{ik} &= \sum_{j=1}^{|SRP|} idf_j \cdot tf_{N_{ik},j} \cdot e \\
 \text{where } e &= \begin{cases} 1 & \text{if query path is exactly matched} \\ 0 & \text{otherwise} \end{cases}
 \end{aligned} \quad (8)$$

The Equation 8 represents the weight of a leaf node  $N_{ik}$  computed by the normal TF-IDF weights when a query path matches its tree path from the root to the leaf node. However, other weighting functions could be used as well [17, 22].

#### 3.2 An internal node's weight

The previous subsection's weighting is equal to logical unit passage retrieval. To compute internal node's weight, [8, 23] accumulated a document weight according to the following general weighting formula :  $W_i = \sum_{j=1}^n 0.5^{j-1} \cdot j^{th}$  weight in document  $i$ . However, these weighting schemes produce no great improvement when we retrieve document because they could not utilize structural proximities. For instance, if a user wishes to find "a paper, book or related materials whose title contains 'contents' and published in 1996", the query may be  $//title = 'contents' //year = '1996'$ . In this case, nodes matched to the title or year are more close, the document has more important meaning. On the other hand, if the matched nodes are much apart, the document is less important. But [8, 23] can not differentiate these two case. However, we might acquire better results when we calculate relative proximity using V and H-proximity.

Formally, we define  $H-sum$  operator ( $\Xi$ ) between two sibling nodes based on  $H-prox$ ,  $V-sum$  operator ( $\Upsilon$ ) between ancestor-descendant nodes based on  $V-prox$  and the weight

of an internal node using  $\Xi$  and  $\Upsilon$ .

$$\begin{aligned}
W_{i_q} &= W_{ij} \Xi W_{ik} \text{ defined as} \\
W_{i_q} &= \max(W_{ij}, W_{ik}) \\
&+ H\text{-prox}(N_{ij}, N_{ik}) \cdot \min(W_{ij}, W_{ik}) \quad (9) \\
\text{where } q &= \frac{j * W_{ij}}{W_{ij} + W_{ik}} + \frac{k * W_{ik}}{W_{ij} + W_{ik}}
\end{aligned}$$

$$\begin{aligned}
W_{ii} &= W_{ii} \Upsilon W_{ik} \text{ defined as} \\
W_{ii} &= W_{ii} + V\text{-prox}(N_{ii}, N_{ik}) \cdot W_{ik} \quad (10)
\end{aligned}$$

$$\begin{aligned}
W_{ii} &= W_{ii} \Upsilon \max\{W_{ij} \Xi W_{ik} \Xi \dots \Xi W_{in}\} \quad (11) \\
\text{where } N_{ii} &\text{ is parent of } N_{ij}, N_{ik}, \dots, N_{in}
\end{aligned}$$

We design  $\Xi$  to preserve the weight of better one but decrease the weight of the other based on  $H\text{-prox}$ . When the weights of two nodes reside in a node,  $H\text{-sum}$  of these two nodes produce the sum of these two nodes with no loss; however, when the one is far apart from the other,  $H\text{-sum}$  of these two nodes is nearly equal to the weight of the one whose weight is greater than the other. The position of a horizontally summed node ( $q$  of Equation 9) for the further  $H\text{-sum}$  is the centroid of these two nodes. On the contrary with  $\Xi$ ,  $\Upsilon$  use  $V\text{-prox}$  directly applied to only descendant nodes. Because associative law for  $\Xi$  is invalid, we employ  $\max\{\dots\}$ .

### 3.3 Heterogeneity

In the Section 2.2, we proposed the SRP querying model. Although the SRP querying model is easy to use, the expression power of an SRP query is worse than that of an XPath query because we assume the independence among query paths. To overcome this disadvantage, we adopt structural proximity among query paths in the ranking. For instance, formulating an SRP query, a user prefers a document which encompasses all kinds of query paths which are structurally adjacent. More specifically, if a user asks `//article/title='system' //article/year='2000'` which may be translated from an XPath query `//article[title='system']/year='2000'`, we should give higher weight to a document fulfilling both conditions than documents satisfying only one of them. Moreover, more paths with more structural proximities, more weight we assign to the document, which result in best matching policy of an XPath query. We define the degree of structural matches and proximities of unique query paths as a heterogeneity.

We devise the heterogeneity of a node like the weight of an internal node. Firstly, we define the heterogeneity value ( $H_{ik}$ ) of a leaf node ( $N_{ik}$ ), which requires two attributes for a leaf node –  $H_{ik,j}^S$ , the level of a leaf node to  $j^{\text{th}}$  query path for the later computation of  $V\text{-prox}$ ;  $H_{ik,j}^T$ , the heterogeneity value to  $j^{\text{th}}$  query path.  $H_{ik}^T$  is a vector for these two attributes of all kinds of query path. Our formal definition is as follows :

$$H_{ik,j}^T = \begin{cases} 1 & \text{if } j^{\text{th}} \text{ query path is exactly matched} \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

$$H_{ik,j}^S = \text{level}(N_{ik}) \quad (13)$$

$$H_{ik} = \sum_{j=1}^{|SRP|} H_{ik,j}^T \quad (14)$$

$$H_{ik}^T = \left( \begin{array}{c} H_{ik,1}^T \ H_{ik,2}^T \ \dots \ H_{ik,|SRP|}^T \\ H_{ik,1}^S \ H_{ik,2}^S \ \dots \ H_{ik,|SRP|}^S \end{array} \right) \quad (15)$$

The heterogeneity value of a leaf node for  $j^{\text{th}}$  query path represents the existence of exactly matched  $j^{\text{th}}$  query path. If  $j^{\text{th}}$  query path is exactly matched,  $H_{ik,j}^T$  is 1. When no query term of the  $j^{\text{th}}$  query path matches in node  $N_{ik}$ ,  $H_{ik,j}^T$  is 0.

For the heterogeneity of an internal node, we define the heterogeneity sum operator ( $\Phi$ ) between two sibling nodes  $N_{ik}$  and  $N_{iq}$  and define  $H_{ii}^T$ , the heterogeneity of an internal node ( $N_{ii}$ ).

$$H_{ii}^T = H_{ik}^T \Phi H_{iq}^T \quad \text{w.l.o.g. Let } H_{ik} \geq H_{iq} \quad (16)$$

for each  $j$

$$H_{ii,j}^T = \begin{cases} \tau & \text{if } H_{ik,j}^T < \tau \\ H_{ik,j}^T & \text{otherwise} \end{cases} \quad (17)$$

$$H_{ii,j}^S = H_{iq,j}^S$$

$N_{it}$  : may be a sibling node or the parent node of  $N_{ik}$  and  $N_{iq}$

$$\tau : \left( \frac{\text{level}(N_{ik})}{H_{iq,j}^S} \right)^h$$

$$H_{ii}^T = \max\{H_{ij}^T \Phi H_{ij}^T \Phi \dots \Phi H_{in}^T\} \quad (18)$$

where  $N_{ii}$  is parent of  $N_{ij}, N_{ik}, \dots, N_{in}$

In Equation 16,  $\tau$  adopt  $V\text{-prox}$ . If  $h$  is 0, then we count only the number of unique paths; however, if  $h$  is greater than 1 or equal to 1,  $V\text{-dist}$  will affect heterogeneity. When  $h > 1$  and two query paths are joined in a root node,  $\tau$  will be  $\frac{1}{\text{the level of a leaf node}}$ . But if  $h > 1$  and two query paths are merged in a node whose level is adjacent to leaf node,  $\tau$  will be nearly 1. Equation 18 computes the heterogeneity of an internal node. Since  $\Phi$  is not associative, we also employ  $\max\{\dots\}$ . Like internal node's weight, we may compute the heterogeneities of internal nodes from leaf nodes up to the root.

The weight of a node  $N_{ij}$  ( $W_{ij}$ ) reflecting  $H_{ij}$  is as follows :

$$W_{ij} = \frac{|SRP| + k \cdot H_{ij}}{|SRP|} W_{ij} \quad (19)$$

When  $k = 0$ ,  $W_{ij}$  yields the same results of  $W_{ij}$ , which means that only reflects proximities among query terms. But if  $k > 0$ , the heterogeneity will affect a node's weight.

For the structured query, we can measure the degree of the structural matching by heterogeneity. On the contrary, for the unstructured query, the heterogeneity of a node represents the degree of best matching boolean 'and' query among terms.

**THEOREM 3.1.** *When some XPath query is translated into SRP query, the heterogeneity of the more exact match is greater than that of less exact match if there are no duplicate tags from root to leaf in the document  $i$ .*

**Proof :** The proof is reserved for the reader.

In accordance with Theorem 3.1, the heterogeneity can approximate most XPath queries. Therefore, we may say that an SRP query can represent most of the expression power of XPath.

### 3.4 Document length normalization

## 4. EXPERIMENTS

## 4.1 Query Set

## 4.2 Query Result

## 4.3 Result Analysis

## 5. CONCLUSION AND FUTURE WORK

The characteristics of our XML retrieval approach can be summarized as follows: (1) Tags describe the structure of a document and (2) this structure of the documents in a collection may be complex. (3) These two properties cause that users have difficulties in query formulation. Current approaches for XML retrieval do not provide appropriate solutions for this problem. For this reason, we propose a new XML retrieval model which consider proximities of query terms as well as heterogeneity of query paths, and we define appropriate weighting schemes. For the problem of irregular document structures, we proposed SRP querying model, which facilitates the formulation of structural queries for end users.

Another line of further research is the extension of our query model. So far, the logical query structure is linear. In the future we will also consider Boolean connectives, e.g. by applying the p-norm model.

## 6. REFERENCES

- [1] <http://www.w3.org/DOM/>.
- [2] <http://www.w3.org/TR/xpath>.
- [3] Ricardo A. Baeza-Yates and Gonzalo Navarro. Integrating Contents and Structure in Text Retrieval. *SIGMOD Record*, 25(1):67–79, 1996.
- [4] Neil Bradley. *The XML Companion, 2nd Edition*. Addison-Wesley, 1999.
- [5] J.P. Callan. Passage-Level Evidence in Document Retrieval. In W. Bruce Croft and C.J. van Rijsbergen, editors, *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 302–310, Dublin, Ireland, July 1994. Springer-Verlag.
- [6] C. Clarke, G. Cormack, and F. Burkowski. Shortest substring ranking (MultiText experiments for TREC-4). In D. K. Harman, editor, *Proceedings of the 4th Text Retrieval Conference(TREC-4, Washington, D.C., Nov.)*, pages 295–304, 1995.
- [7] Charles L. A. Clarke and Gordon V. Cormack. Shortest-substring retrieval and ranking. *TOIS*, 18(1):44–78, January 2000.
- [8] G.V. Cormack, C.L.A. Clarke, C.R. Palmer, and S.S.L. To. Passage-based refinement (MultiText experiments for TREC-6). In E. M. Voorhees and D. K. Harman, editors, *Proceedings of the 6th Text Retrieval Conference(TREC-6, Gaithersburg, Maryland, Nov.)*, pages 303–320, 1997.
- [9] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing Semistructured Data with STORED. In *SIGMOD*, pages 431–442, 1999.
- [10] Daniela Florescu and Donald Kossmann. Storing and Querying XML Data Using an RDBMS. *Data Engineering Bulletin*, 22(3), 1999.
- [11] Roy Goldman, Jason McHugh, and Jennifer Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *WebDB*, pages 25–30, 1999.
- [12] D. Hawking and P. Thistlewaite. Proximity operators - so near and yet so far. In D. K. Harman, editor, *Proceedings of the 4th Text Retrieval Conference(TREC-4, Washington, D.C., Nov.)*, pages 131–143, 1995.
- [13] M. Kaszkiel and J. Zobel. Passage retrieval revisited. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '97, Philadelphia, PA, USA, July 27-31)*, pages 178–185, 1997.
- [14] V.I. Levvenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.*, pages 707–710, 1966.
- [15] S. H. Myaeng and et. al. A Flexible Model for Retrieval of SGML Documents. In *SIGIR*, pages 138–145, 1998.
- [16] Gonzalo Navarro and Ricardo Baeza-Yates. Proximal nodes: a model to query document databases by content and structure. *TOIS*, 15(4):400–435, 1997.
- [17] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *SIGIR*, pages 232–241, 1994.
- [18] G. Salton and C. Buckley. Automatic text structuring and retrieval: Experiments in automatic encyclopedia searching. In *ACM/SIGIR Conference*, pages 21–31, 1991.
- [19] T. Schlieder and M. Meuss. Result ranking for structured queries against XML documents. In *First DELOS workshop on Information Seeking, Searching and Querying in Digital Libraries*, 2000.
- [20] Takeyuki Shimura, Masatoshi Yoshikawa, and Shunsuke Uemura. Storage and Retrieval of XML Documents using Object-Relational Databases. In *DEXA*, pages 206–217, 1999.
- [21] D. Shin, H. Jang, and H. Jin. BUS: An Effective Indexing and Retrieval Scheme in Structured Documents. In *Digital Libraries*, pages 235–243, 1998.
- [22] Howard R. Turtle and W. Bruce Croft. Evaluation of an Inference Network-Based Retrieval Model. *TOIS*, 9(3):187–222, 1991.
- [23] Ross Wilkinson. Effective retrieval of structured documents. In *SIGIR '94*, pages 311–317, 1994.
- [24] Justin Zobel, Alistair Moffat, Ross Wilkinson, and Ron Sacks-Davis. Efficient retrieval of partial documents. *Information Processing and Management*, 31(3):361–377, 1995.

## APPENDIX

### A. SRP QUERY SYNTAX

```
term := id
tag_name := term
| '?'
path_elem := / tag_name filter
| // tag_name filter
path := path_elem path
| //
| ε
query_path := path '=' term
| term
query := query_path query
| query_path
filter := [query] filter
| ε
```

# An Appropriate Unit of Retrieval Results for XML Document Retrieval \*

Kenji Hatano  
Nara Institute of Science and  
Technology  
8916-5 Takayama, Ikoma  
Nara 630-0192, Japan  
hatano@is.aist-nara.ac.jp

Hiroko Kinutani  
Nara Institute of Science and  
Technology  
8916-5 Takayama, Ikoma  
Nara 630-0192, Japan  
hiroko-k@is.aist-nara.ac.jp

Masahiro Watanabe  
National Institute of Special  
Education  
5-1-1 Nobi, Yokosuka  
Kanagawa 239-0841, Japan  
masahiro@nise.go.jp

## ABSTRACT

In the research field of document retrieval using a few keywords as a query, retrieval results returned by information retrieval systems are whole documents or document fragments. However, they are not suitable for document retrieval since they are not congruent with the information which users are searching for. Therefore, we believe that retrieval results should be portions of the documents, such as chapter, section, or subsection in the documents. That is, the most important concern of document retrieval is to define the retrieval unit of XML documents, which is meaningful for users. In this paper, we propose a method to define an appropriate unit of XML document by analyzing both contents and structure of XML documents in order to realize a keyword-based XML document retrieval system. In our method, we utilize three information extracted from XML documents, and decide the appropriate size of partial XML documents as the unit.

## Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: Search process; H.3.4 [Systems and Software]: Performance evaluation (efficiency and effectiveness)

## General Terms

Algorithm, Performance, Measurement, Experimentation

## Keywords

XML document retrieval, Appropriate unit of XML document, Keyword-based query

## 1. INTRODUCTION

XML (Extensible Markup Language) [3] is becoming widely used as a standard document format in many application domains. In the near future, various kinds of data and documents will be expressed in XML. Therefore, XML document retrieval systems will become very important tools for users to explore XML documents.

---

\*This work was partly supported by the Ministry of Education, Culture, Sports, Science and Technology, Japan, under grants #14019064 and #14780325, and by CREST Program "Advanced Media Technology for Everyday Living" of Japan Science and Technology Corporation.

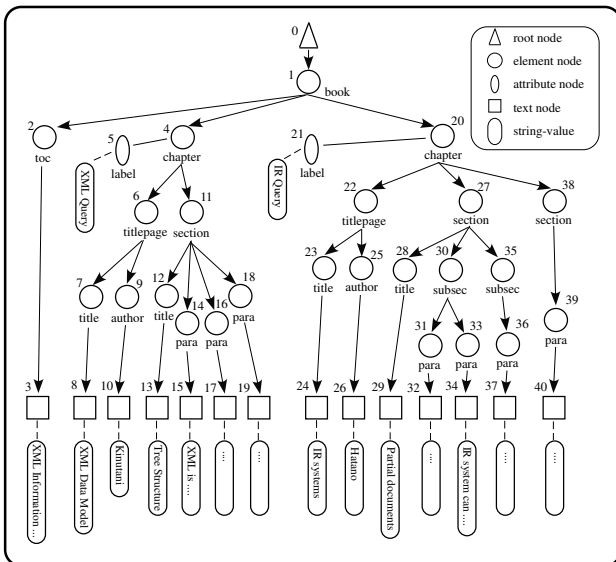
In spite of the big demand for XML document retrieval systems, they have not yet available. It is true that many XML query languages have been proposed [2]. However, XML documents have various kinds of document structure, so that specifying document structures which is adopted in XML query languages is not clearly suitable for a query of XML document retrieval systems. This is because it is hard for users to enter the document structure as a query into an XML document retrieval system for specifying the logical structures of heterogeneous set of XML documents. Therefore, we envision a much simpler form of query, which consists of a few keywords.

In order to develop a keyword-based XML document retrieval system, XML documents have to be divided into partial XML documents beforehand. Because we cannot retrieve partial XML documents using a keyword-based query if the keyword-based XML document retrieval system does not divide XML documents into partial XML documents. XML is a markup language, so that it is easy to automatically divide XML documents into partial XML documents using their markup [9]. However, if the XML documents are divided as much as possible, the number of the partial XML documents will become huge. That is, it takes much time to retrieve partial XML documents related to a users' query. For this reason, we have to decide the granularity of the partial XML documents as a unit of retrieval results, and have to reduce the number of the partial XML documents.

In this paper, we propose a method to define an appropriate unit of XML document by analyzing both contents and structure of XML documents, and adapt our method to a document retrieval system for retrieving partial XML documents as retrieval results. In concrete terms, we investigate three information such as structural information, content information, and statistics information of partial XML documents, and decide the appropriate size of partial XML documents as a unit of XML documents based on these information.

Some researches have been proposed a method to define a fine granule of the data. Especially, finding an appropriate unit of Web documents is famous in recent year [11, 13]. Moreover, they also have proposed a method to define an appropriate unit of semi-structured documents [4], so that we believe that our research topic will be important for XML





**Figure 1: A Tree Representation of an XML Document.**

document retrieval in the near future.

The remainder of this paper is organized as follows. We firstly describe our data model of XML document retrieval in Section 2. Then, we explain how to define an appropriate unit of XML document in Section 3, and report experimental results analyzing the three information extracted from XML documents in Section 4. Finally, we conclude our paper in last section.

## 2. XPATH DATA MODEL

In this section, we follow the notations and data model defined in XPath 1.0 [5].

In the XPath data model, an XML document is modelled as a hierarchical tree. Figure 1 shows the logical structure of a sample XML document. The order of nodes, which is numbered 1 through 40, is document order. Although there are seven types of nodes in the XPath data model, we limit our attention to the root node, element nodes, attribute nodes and text nodes for the sake of simplicity<sup>1</sup>. In an XML tree, leaf nodes are text nodes or attribute nodes, and intermediate nodes are element nodes. The child element node of the root node is called the *document node*. In the XPath data model, a somewhat strange parent/child relationship between the element nodes and attribute nodes is used. An element node is a parent of an attribute node, but the attribute node is not a child of the element node.

In our model, however, we regard the attribute node as a child of the element node. This is the only difference between the XPath data model and our data model. The *expanded-name* of an element node (or attribute node) is the element type name (or attribute name) of the node. The *string-value* of a text node is the text itself, the *string-value* of an attribute node is the value of the attribute, and

<sup>1</sup>The remaining three types of nodes are namespace nodes, processing instruction nodes and comment nodes.

the *string-value* of an element node is the concatenation of the string-values of all text-node descendants of the element node.

Until now, two kinds of XML document retrieval model based on the XPath data model have been studied [1]: one is the non-overlapping match [6] and the other is the proximal nodes [12]. Our model of partial XML documents is similar to the model based on the proximal nodes. That is, our logical model of partial XML documents is a subtree whose root node is an element node. Therefore, we can identify a partial XML document by the reference number,  $n$ , of the root node of the partial XML document tree. We refer to such a partial XML document as “partial document # $n$ .”

We think that retrieval results are also partial XML documents following the XPath data model if we adopt the INEX test collection<sup>2</sup> as target XML documents. For this reason, our XML document retrieval system can divide XML documents of the INEX test collection into partial XML documents, and identifies them by their reference number.

## 3. Coherent Partial Document

In order to retrieve partial XML documents using a keyword-based query, XML documents stored in an XML document retrieval system have to be divided into partial XML documents. However, if the XML documents are divided as much as possible, the number of divided partial XML documents is huge<sup>3</sup>. Therefore, we cannot perform an efficient retrieval of XML documents.

In this section, we propose a method to define an appropriate unit of XML document, and reduce the number of divided partial XML documents for an efficient retrieval.

### 3.1 What is an Appropriate Unit of XML Document?

In our approach, we have to find partial XML documents which are coherent and meaningful portions. We mean that the partial XML documents must have an appropriate unit of XML document.

For example, let us consider the case when a user issues a single keyword query “Hatano.” Which partial documents are relevant as an answer of this query? The minimum partial document containing a character string “Hatano” is the partial document #25. A text representation of this partial document is shown below:

```
<author>Hatano</author>
```

We consider, however, this partial document is not informative enough for the user, because the user cannot know what “Hatano” has authored. Obviously, the other end of

<sup>2</sup>The INEX test collection is constructed by INEX Project organized by the DELOS Network of Excellence for Digital Libraries. The INEX project was launched in order to solve the problems with which the researchers of XML document retrieval are faced.

<sup>3</sup>The number of divided partial document is as same as the number of intermediate nodes of XML documents.

the line, i.e. returning the whole document, is not adequate either. Because the document in Figure 1 has two chapters, and “Hatano” is the author of the second chapter. For this reason, we believe that partial document #20 will be the most relevant as an answer of the query. That is, we regard partial document #20 as a semantically consolidated granule of documents.

In XML document retrieval, we believe that such semantically consolidated granule of XML documents should be retrieved as retrieval results. In our research, we call this type of partial XML documents as *Coherent Partial Document (CPD)*. If we adopt the CPD as a unit of retrieval results, the number of retrieval results are reduced. This is because the CPD does not equal to partial documents divided from XML documents; in short, the size of the partial XML documents might be too small or too large as that of the CPD.

*Context search* is used for representing a retrieval method which returns CPDs as an answer set of a keyword-based user’s query. It can automatically identify the CPDs without DTD (Document Type Definitions) of XML documents. The reason for not using DTD is that XML documents on the Net may have no DTD or have a great variety of DTDs. Under such circumstances, the most important concern is to define an appropriate size of CPDs of XML documents.

In order to define the size of CPDs, we analyze both contents and structures of XML documents and utilize the following information obtained by the analyses:

- **Structural information**

If we construct DOM trees of XML documents using an XML parser, we can grasp an element information of XML documents, such as element name, path expression of the element, and relationship between other elements. It should be appreciated that we can reconstruct the XML documents using only element information.

- **Content information**

The content information expresses the content of documents. That is, it is accounted for counting word frequencies of XML documents. In this research, the content information are feature vectors of partial XML documents based on the tf-idf schemes.

- **Statistics information**

In our previous approach [7], we utilized the first two information to define the CPDs. However, the number of CPDs of the INEX test collection did not decrease dramatically. Therefore, we also utilize the statistics information of XML documents in this paper. Especially, the number of words and element nodes in partial XML documents are important for deciding the appropriate size of CPDs, we believe.

These information analyzed by a structure analyzer and a content analyzer are utilized to define the size of CPDs of XML documents.

In the following subsections, we explain functions of two

analyzers.

## 3.2 Structure Analyzer

The structure analyzer generates an index file of document structures of XML documents to utilize both structural and statistics information.

Table 1 shows the result of analyzing the XML document shown in Figure 1 using the structure analyzer. From this figure, we can appreciate many kinds of information such as the number of elements in the XML document, path expressions of the elements, root node of each partial document, the number of the partial documents in the XML document, and the number of words in the partial documents. Using these analyses, it becomes possible to decide the appropriate size of CPDs of the XML document statistically.

For example, we can get 24 partial documents<sup>4</sup> from the XML document shown in Figure 1. However, the sizes of partial document #9 and #25 are one word, so that they are too small as the size of CPDs. In contrast, we could get partial document #4, #20, #27, #30, #35, and #38 as CPDs using our previous method. However, we also should extract partial document #11 as a CPD, because other partial documents whose root node is as same as that of the partial document are extracted as CPDs. Thus, we believe it is useful to use not only the structural information, but also the statistics information for the definition of the appropriate size of a CPD.

## 3.3 Content Analyzer

The content analyzer generates another index file based on the word frequencies of partial XML documents. The reason for analyzing partial XML documents is to retrieve partial documents using a keyword-based query.

Table 2 shows the result of analyzing the XML document shown in Figure 1 using the content analyzer. The weights are calculated by a keyword weighting strategy based on word frequencies of partial XML documents. Using this result, we generate an index file for searching appropriate partial XML documents relevant to a user’s query. This function is as same as the one which the current document retrieval systems have; however, we need to apply a keyword weighting strategy of having specialized in partial document retrieval when we calculate the weights of each word. Because words appeared in an XML document also appear in its partial documents<sup>5</sup>.

Finally, we integrate these index files generated by two analyzers into a compound index file for efficient retrieval of partial XML documents using a keyword-based query. The compound index file is a kind of an inverted file which consists of not only the content information, but also the structural information. Needless to say, a unit of retrieval results is a CPD defined by analyzing three information described in Section 3.1.

<sup>4</sup>Because the number of intermediate nodes is 24.

<sup>5</sup>In our previous research [8], we proposed a keyword weighting strategy for partial XML documents in order to solve the problem.

Table 1: Structure Analyzer.

element	path expression	partial doc. #	# of words
author	/book[1]/chapter[1]/titlepage[1]/author[1]	9	1
author	/book[1]/chapter[2]/titlepage[1]/author[1]	25	1
book	/book[1]	1	324
chapter	/book[1]/chapter[1]	4	92
chapter	/book[1]/chapter[2]	20	185
para	/book[1]/chapter[1]/section[1]/para[1]	14	20
para	/book[1]/chapter[1]/section[1]/para[2]	16	18
para	/book[1]/chapter[1]/section[1]/para[3]	18	36
para	/book[1]/chapter[1]/section[1]/subsec[1]/para[1]	31	60
para	/book[1]/chapter[1]/section[1]/subsec[1]/para[2]	33	25
para	/book[1]/chapter[1]/section[1]/subsec[1]/para[3]	36	70
para	/book[1]/chapter[1]/section[2]/para[1]	39	18
section	/book[1]/chapter[1]/section[1]	11	78
section	/book[1]/chapter[2]/section[1]	27	159
section	/book[1]/chapter[2]/section[2]	38	18
subsec	/book[1]/chapter[2]/section[1]/subsec[1]	30	85
subsec	/book[1]/chapter[2]/section[1]/subsec[2]	35	70
title	/book[1]/chapter[1]/titlepage[1]/title[1]	7	8
title	/book[1]/chapter[1]/section[1]/title[1]	12	4
title	/book[1]/chapter[2]/titlepage[1]/title[1]	23	3
title	/book[1]/chapter[2]/section[1]/title[1]	28	4
titlepage	/book[1]/chapter[1]/titlepage[1]	6	9
titlepage	/book[1]/chapter[2]/titlepage[1]	22	4
toc	/book[1]/toc[1]	2	47

Table 2: Content Analyzer.

word	Partial Doc. #	weight
xml	1	0.943
information	1	0.449
query	1	0.143
	...	
xml	2	0.833
information	2	0.231
	...	
xml	4	0.632
query	4	0.486
data	4	0.435
model	4	0.297
	...	

#### 4. PRELIMINARY EXPERIMENT

As we described in Section 3, the most important concern of XML document retrieval is to define the retrieval unit of XML documents. The size of partial XML documents are differ, so that we cannot define an appropriate size of CPDs if we use only structure and content information. Therefore, we utilize not only structural and content information of XML documents, but also the statistics information. In this section, we report the results of preliminary experiments for investigating the statistics information.

In this experiments, we used the INEX test collection which consists of a set of journals of IEEE Computer Society. Firstly, we divided XML documents of the INEX test collection into partial XML documents as much as possible. The number of divided partial XML documents is about one mil-

lion.

When we extract the content information from the divided partial documents, it is important to carry out stemming and elimination of stopwords. However, it takes a long time to perform these processes, so that we did not carry out these processes. Instead, we defined the following indicator concerning the kinds of word, and worked up some statistics about the content information.

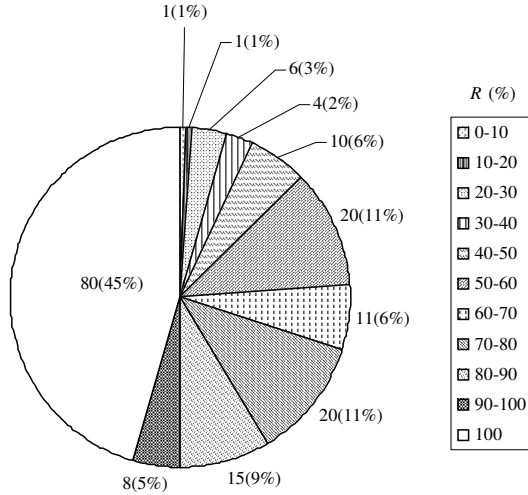
**DEFINITION 1** (THE RATIO OF KINDS OF WORD:  $R$ ). *A partial XML document  $d$  contains some words in itself. We assume that the number of the words is  $N$  and the kinds of word is  $n$  in the partial document. At this time, the ratio of kinds of word  $R$  is defined as follows:*

$$R = \frac{n}{N}$$

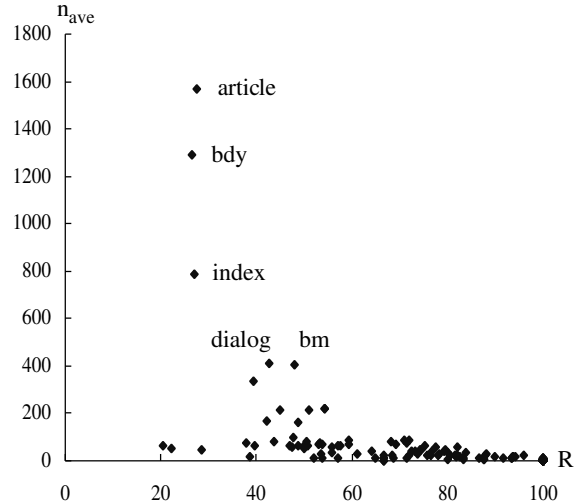
Table 3 shows the number of partial documents, the number of words  $N$ , and the kinds of word  $n$  in the partial documents. The elements in the table are sorted with the average number of words  $N_{ave}$ . As in Table 3, the elements located on higher level of document structure, such as **books**, **journal**, **article**, and so on, were ranked as higher, because the size of the partial documents whose root node is a higher-level-element of XML documents are larger. Moreover, we also found that the ratio of kinds of word  $R$  of the partial XML documents which contain many words in themselves is smaller than those of others. In short, we can forecast that the partial documents whose the ratio of kinds of word  $R$  is large have an aspect of data. Because the ratio of kinds of word  $R$  must be small if they have an aspect of document. We think the partial XML documents with an aspect of data

**Table 3: A Result of Content Analysis (Top 20).**

element	# of partial document	# of words: $N$			kinds of word: $n$			$R$ (%)
		Ave. ( $N_{ave}$ )	Max ( $N_{max}$ )	Min ( $N_{min}$ )	Ave. ( $n_{ave}$ )	Max ( $n_{max}$ )	Min ( $n_{min}$ )	
books	116	528,924	1,530,446	73,483	45,512	93,216	10,930	8.60
journal	760	80,730	223,661	28,896	12,455	24,655	7,179	15.43
article	10,792	5,682	38,269	43	1,563	8,130	39	27.52
bdy	10,792	4,866	33,850	15	1,289	7,197	15	26.50
index	111	2,923	13,354	467	787	1,998	300	26.92
dialog	194	969	5,661	33	413	1,698	32	42.66
sec	62,238	849	17,619	1	336	4,700	1	39.57
bm	8,881	842	18,347	2	405	5,175	2	48.18
ss1	53,292	477	14,142	2	215	3,798	2	45.04
app	5,533	418	11,810	2	213	1,712	2	51.09
bib	7,432	406	5,693	9	220	2,103	9	54.41
bibl	7,438	405	5,693	9	220	2,103	9	54.41
ss3	105	397	2,471	22	167	511	20	42.13
ss2	14,278	326	5,271	2	159	1,579	2	48.86
tgroup	4,995	321	3,961	2	66	409	2	20.60
tbody	4,993	238	3,715	2	53	399	2	22.39
proof	3,424	210	6,555	3	100	1,186	3	47.83
l4	98	198	1,769	9	75	431	9	38.08
dl	317	181	2,777	18	79	1,226	12	43.78
lb	53	171	1,172	31	67	157	16	39.52



**Figure 2: Ratio of Kinds of Word  $R$ .**



**Figure 3: Relationship between  $R$  and  $n_{ave}$ .**

are not informative, so that they cannot be CPDs of XML documents.

On the other hand, when we focused the ratio of kinds of word  $R$  (see Figure 2), we found that 45 percent of 183 kinds of partial document has 100 percent as the ratio  $R$ . As we described above, the partial XML documents whose ratio of kinds of word is 100 percent are not informative; consequently, they cannot be CPDs.

Furthermore, when we investigated the kinds of word  $n$  in all partial documents, we observed that the partial documents whose root node is a certain element, such as **article**, **body**, **index**, and so on, have large number of the kinds of word although the ratio of kinds of word is small (see Figure 3).

Therefore, we believe that there is a relationship between the number and the ratio of kinds of word.

In addition, it was thought that if partial XML documents with same element as a root node are appeared repeatedly in XML documents, they are structurally meaningful; that is, they are suitable for CPDs, we believe.

From the above-mentioned points, we believe that we can decide a unit of CPDs of XML documents. We may summarize about the definition of CPDs of XML documents as follows:

- The partial XML documents whose the ratio of kinds of word  $R$  is 100 percent are not suitable for CPDs, because they have an aspect of data, not that of document.
- Most partial XML documents whose ratio of kinds of word  $R$  is less than 100 percent contain less than 200 kinds of word. Therefore, we believe the size of a CPD should be less than 200 words.
- Meaningful partial XML documents are appeared repeatedly in XML documents. Consequently, the partial XML documents whose frequency is large and whose ratio of kinds of word is small are suitable for CPDs.

## 5. CONCLUSION

In this paper, we proposed a method to define an appropriate granule of XML documents, a CPD, by analyzing both contents and structure of XML documents in order to realize a keyword-based XML document retrieval system. In our system, XML documents are divided into CPDs of XML documents beforehand, so that we can retrieve partial XML documents relevant to a user's query which consists of some keywords efficiently. In order to define an appropriate size of a CPD of XML documents, we investigated both contents and structure of XML documents using two analyzers, and utilized a heuristic approach based on results of the analysis.

However, we cannot carry out experiments for verification of our method using the INEX test collection in this paper. Therefore, we have to verify the effectiveness of our method as soon as possible. Moreover, if we can define an appropriate size of a CPD, we have another problem about a similarity calculation method of between CPDs and a users' query. The current document retrieval systems calculate the similarities using only contents of whole documents; by contrast, the XML document retrieval system should calculate the similarities using both contents and structure of XML documents, we believe. Lalmas and we have already studied solving this problem for semi-structured documents such as SGML and XML documents [8, 10], so that we will adopt these approaches to our XML document retrieval system.

## 6. ADDITIONAL AUTHORS

Masatoshi Yoshikawa (Nagoya University; Furoh, Chikusa, Nagoya, Aichi 464-8601, Japan, E-mail: [yosikawa@itc.nagoya-u.ac.jp](mailto:yosikawa@itc.nagoya-u.ac.jp)) and Shunsuke Uemura (Nara Institute of Science and Technology; 8916-5 Takayama, Ikoma, Nara 630-0192, Japan, E-mail: [uemura@is.aist-nara.ac.jp](mailto:uemura@is.aist-nara.ac.jp)).

## 7. REFERENCES

- [1] R. Baeza-Yates and B. Ribeiro-Neto, editors. *Modern Information Retrieval*. ACM Press, 1999.
- [2] A. Bonifati and S. Ceri. Comparative Analysis of Five XML Query Languages. *ACM SIGMOD Record*, 29(1):68–79, Mar. 2000.
- [3] T. Bray, J. Paoli, C. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). <http://www.w3.org/TR/2000/REC-xml-20001006>, Oct. 2000. W3C Recommendation 6 October 2000.
- [4] S. Chakrabarti. Text Search for Fine-grained Semi-structured Data. Tutorial Notes of the 28th International Conference on Very Large Data Bases, Aug. 2002.
- [5] 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.
- [6] R. Daniel, S. DeRose, and S. Maler. XML Pointer language (XPointer) Version 1.0. <http://www.w3.org/TR/xptr>, June 2000. W3C Candidate Recommendation 7 June 2000.
- [7] K. Hatano, H. Kinutani, M. Yoshikawa, and S. Uemura. Extraction of Partial XML Documents Using IR-based Structure and Contents Analysis. In *Conceptual Modeling for New Information Systems Technologies*, volume 2465 of *LNCS*, pages 334–347. Springer-Verlag, 2002.
- [8] K. Hatano, H. Kinutani, M. Yoshikawa, and S. Uemura. Information Retrieval System for XML Documents. In *Proc. of the 13th International Conference on Database and Expert Systems Applications*, volume 2453 of *LNCS*, pages 758–767. Springer-Verlag, Sep. 2002.
- [9] 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, July 1997.
- [10] M. Lalmas. Dempster-Shafer's Theory of Evidence applied to Structured Documents: modelling Uncertainty. In *Proc. of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 110–118. ACM, July 1997.
- [11] W.-S. Li, K. Candan, Q. Vu, and D. Agrawal. Retrieving and Organizing Web Pages by "Information Unit". In *Proc. of the 10th International World Wide Web Conference*, pages 230–244, May 2001.
- [12] 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, 1997.
- [13] K. Tajima, K. Hatano, T. Matsukura, R. Sano, and K. Tanaka. Discovery and Retrieval of Logical Information Units in Web. In *Proc. of the 1999 ACM Digital Library Workshop on Organizing Web Space*, pages 13–23, Aug. 1999.

# CSIRO INEX experiments: XML search using PADRE

*Anne-Marie Vercoustre*<sup>1</sup>     *James A. Thom*<sup>2</sup>     *Alexander Krumpholz*<sup>1</sup>  
*Ian Mathieson*<sup>1</sup>     *Peter Wilkins*<sup>1</sup>     *Mingfang Wu*<sup>1</sup>     *Nick Craswell*<sup>3</sup>  
*David Hawking*<sup>3</sup>

<sup>1</sup>CSIRO Mathematical and Information Sciences  
Private Bag 10, South Clayton MDC, VIC 3169, Australia

<sup>2</sup>School of Computer Science and Information Technology, RMIT University  
GPO Box 2476V, Melbourne 3001, Australia

<sup>3</sup>CSIRO Mathematical and Information Sciences  
GPO Box 664, Canberra, ACT 2601, Australia

Email for correspondence: *Anne-Marie.Vercoustre@csiro.au*

## Abstract

This paper reports on the CSIRO group's participation in INEX. We indexed documents and document fragments using PADRE the core of CSIRO's Panoptic Enterprise Search Engine. A query translator converts the INEX topics into queries containing selection and projection constraints for the results. Answers are extracted from ranked documents and document fragments based on the projection constraints in the query.

## 1 Introduction

Broadly speaking there are two main approaches to XML retrieval: a database approach as exemplified by query languages such as XQuery and a text retrieval approach as exemplified by search engines ranking documents or document fragments. The database and information retrieval communities have different approaches to query evaluation. The database community focuses on the expressive power of query languages that retrieve exact answers. The information retrieval community focuses on the effectiveness of ranked retrieval. Our approach at CSIRO to the INEX experiment was to add database techniques to an underlying text retrieval technology. Thus we combine selection and ranking of candidate documents and document fragments using information retrieval with a database style projection to extract the final answers. Further discussion of the motivation for our approach is described elsewhere [1].

We discuss issues with topic formulation in Section 2. In Section 3 we describe the overall archi-

**INEX workshop, Schloss Dagstuhl. December 9-11, 2002**

ture of our approach using PADRE, the core of CSIRO's Panoptic Enterprise Search Engine [2]. In Section 4 we outline the INEX runs we made and present some initial results.

## 2 Topics

Figure 1 shows topic 14, which is based on one of the topics proposed by our group to find figures that describe the Corba architecture and the paragraphs that refer to those figures. We are using this query in the rest of the paper as an example to describe our system.

As well as an obvious typographic error in the keywords, the topic finally used in INEX has several limitations. First, we did not correctly formulate the topic due to inadvertently overlooking some aspects of the complex DTD; there are other elements such as `<figw>` that should have logically been included in the topic. This raises a question for semi-structured retrieval — how much information about the structure is it reasonable to expect the average user to know? Second, due to the INEX requirement that answers could only be a single element it was not possible to capture the semantics as described in the narrative, that is an answer “would ideally contain both the figure and the paragraph referring to it”. This could only happen in section elements which would have larger coverage than the specific information need. In defining the syntax and semantics for INEX topics it would have been desirable for different semantics to be given to

```
<te>fig,p</te>
```

meaning an answer would both be a `<fig>` element and a `<p>` element, whereas

```

<?xml version="1.0"
  encoding="ISO-8859-1"?>
<!DOCTYPE INEX-Topic SYSTEM "inex-topics.dtd">
<INEX-Topic topic-id="14"
  query-type="CAS" ct-no="075">
<Title>
  <te>fig,p,ip1</te>
  <cw>Corba architecture</cw>
  <ce>fgc</ce>
  <cw>Figure Corba Architecture</cw>
  <ce>p, ip1</ce>
</Title>
<Description>
  Find figures that describe the Corba architecture
  and the paragraphs that refer to those figures.
</Description>
<Narrative>
  To be relevant a figure must describe the
  standard Corba architecture or a system
  architecture that relies heavily on Corba.
  A figure describing a particular aspect of a
  system will not be regarded as relevant even
  though the system may rely on Corba otherwise.
  Retrieved components would ideally contain both
  the figure and the paragraph referring to it.
</Narrative>
<Keywords>
  CORBA ORB Object Request Broker Architecture
  interface invocation interoperability
  communication protocols IDL
</Keywords>
</INEX-Topic>

```

Figure 1: INEX topic 14

```
<te>fig|p</te>
```

would mean an answer is either element. It is the former that the narrative of this topic implies.

### 3 System overview

#### 3.1 System Architecture

Figure 2 shows the overall architecture of our system. We translate INEX topics into queries comprising a selection component and a projection component; a simplified query is shown in the architecture diagram. The selection component of the query is sent to our search engine, PADRE, which ranks the more similar matching documents and document fragments meeting the selection criteria. The projection component, that is mostly based on the target element component of the topic, is sent to an extractor that extracts the desired answers from the ranked documents and document fragments returned by PADRE.

#### 3.2 PADRE indexing

We extended CSIRO’s document indexing and retrieval system, PADRE [3], to handle XML documents. PADRE is the indexing core of the Panoptic

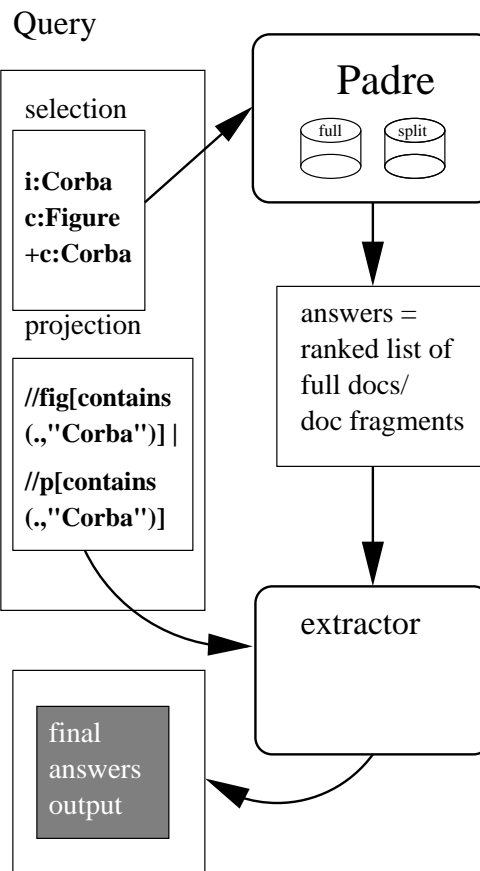


Figure 2: System architecture

Enterprise Search Engine [2] and combines full-text and metadata indexing and retrieval. PADRE enables us to rank documents primarily on how many of the query terms appear in each document or document fragment and secondarily on the relevance score, using a slightly modified form of the Okapi BM25 function [4].

We were able to adapt PADRE’s capability for indexing metadata fields to enable us to index selected XML elements. For example, given the mapping rule

```
//figc → i
```

the index terms for the element

```
<figc>Corba Architecture</figc>
```

would be mapped to the field “i” as i:Corba and i:Architecture.

As each element is processed, the first matching rule determines what metadata field is used to index the content of the element. In processing the content of sub-elements the rules are reapplied. Thus given the mapping rules

```
//p → c
//figc → i
```

```
<figc><p>Corba Architecture</p></figc>
```

would be mapped to c:Corba and c:Architecture.

---

a	- article author
b	- bibliography entry
c	- paragraph text (but not within abstract, keywords, acknowledgements etc)
d	- publication date
f	- figure text
i	- figure caption
j	- journal title
l	- abstract, NOT including 's' keywords
n	- acknowledgements
p	- publisher
q	- affiliation
r	- table text
s	- keywords
t	- article title
u	- url
v	- fragment subset(s)
w	- title of a section
y	- ISSN
z	- volume, issue, pp

---

Figure 3: Fields

This illustrates a weakness in our approach that higher structural elements are ignored.

The mappings are also used in queries. For example, the query “give me documents containing figures with Corba architecture in the caption” can be expressed as `i:Corba i:Architecture`. This query will first return matching documents that contain both “Corba” and “Architecture” in a figure caption, followed by partial matching documents that contain either “Corba” or “Architecture” in a figure caption. Mandatory constraints are supported, so this query could be expressed as `+i:Corba i:Architecture` so all matching documents must contain “Corba” in a figure caption. Phrase querying is also supported, in which case this query could be expressed as `i:"Corba Architecture"` and only documents containing the phrase “Corba Architecture” in the caption of a figure would be returned as answers.

A complete list of the fields is shown in Figure 3 together with the actual mappings in Figure 4

We only defined mappings for concepts that we considered useful for querying the INEX collection. The “v” field is used to allow queries on particular types of documents fragments.

### 3.3 Splitting

As shown in Figure 2 the system uses PADRE to select and rank documents. We wanted to make good use of PADRE’s initial ranking and, since Wilkinson [5] shows that simply extracting elements from ranked documents is a poor strategy, we decided to investigate ranking

---

/books/journal/title	→ j
/books/journal/issue	→ z
/books/journal/publisher	→ p
/books/PANOPTIC-from	→ v
/books/PANOPTIC-genericXPath	→ v
/article/fm/hdr/hdr1/ti	→ j
/article/fm/hdr/hdr1/crt/issn	→ y
/article/fm/hdr/hdr2/obi	→ z
/article/fm/hdr/hdr2/pdt	→ d
/article/fm/hdr/hdr2/pp	→ z
/article/fm/tig/at1	→ t
/article/fm/tig/pn	→ z
/article/fm/au	→ a
/article/bdy/sec	→ c
/article/fm/abs	→ l
/article/fm/abs/p	→ l
/article/PANOPTIC-from	→ v
/article/PANOPTIC-genericXPath	→ v
//ack	→ n
//ack/p	→ n
//kwd	→ s
//kwd/p	→ s
//aff	→ q
//url	→ u
//st	→ w
//bb	→ b
//p	→ c
//p1	→ c
//p2	→ c
//p3	→ c
//ip1	→ c
//ip2	→ c
//ip3	→ c
//ip4	→ c
//ip5	→ c
//ilrj	→ c
//item-none	→ c
//fig	→ f
//figw	→ f
//fgc	→ i
//tbl	→ r

---

Figure 4: Actual mappings

document fragments as well as whole documents. Thus before indexing by PADRE we split the documents into various fragments and indexed the fragments as well as the whole documents. For the content only queries we expected that ranking document fragments as well as whole documents will improve performance by finding the relevant portions of documents, especially where the coverage of whole documents was too broad. For the content and structure queries we expected the splitting to improve the ranking but also envisaged that for queries involving a specific target element further extracting would be required. We describe this further in the next section.



---

```

/article/
/article/bdy//fig/
/article/bdy//figw/
/article/bdy//ilrj/
/article/bdy//ip1/
/article/bdy//ip2/
/article/bdy//ip3/
/article/bdy//ip4/
/article/bdy//ip5/
/article/bdy//item-none/
/article/bdy//p/
/article/bdy//p1/
/article/bdy//p2/
/article/bdy//p3/
/article/bdy//sec/
/article/bdy//tbl/
/article/fm/
/article/fm/abs/
/books/

```

---

Figure 5: Document fragments

We analysed the collection and identified elements to use as fragments based on:

- a reasonable granularity that is not too small, and
- the expected elements for results.

Thus we split document fragments based on the paths shown in Figure 5 We also included some additional context to the fragments such as the filename of the original document and the path within the document to the fragment. This context allows subsequent processing of the document fragment.

We were able to use our existing indexing and retrieval engine to index both the documents and the fragments as one collection although this increased the number “documents” by a factor of 100, and the size in bytes by a factor of 10.

If the query does not contain a projection, then the result of query is simply the ranked list produced by PADRE. Otherwise the extractor described in the next section is applied to the ranked list of documents and document fragments.

### 3.4 Extractor

Many of the content and structure queries contain a projection. We automatically generate the projection when there is a target element in the topic. Example of a projection in a query corresponding to topic 14 is shown in Figure 6. The projection is an XPath specifying the target element or elements to be extracted from the ranked list of documents and document fragments. The algorithm is as follows, for each returned fragment  $f$ :

1. load the fragment, get the name of the embedding article, load the full article  $A$ .

---

```

</query>
<query topic-id="14">
<selection>
  i:Corba
  i:architecture
  c:Figure
  c:Corba
  c:Architecture
  [CORBA ORB Object Request Broker
  Architecture interface invocation
  interoperability communication
  protocols IDL]
</selection>
<projection>
  //fig |
  //p[contains(., "Figure") or
  contains(., "figure") or
  contains(., "Corba") or
  contains(., "corba") or
  contains(., "Architecture") or
  contains(., "architecture")] |
  //ip1[contains(., "Figure") or
  contains(., "figure") or
  contains(., "Corba") or
  contains(., "corba") or
  contains(., "Architecture") or
  contains(., "architecture")]
</projection>
</query>

```

---

Figure 6: Query for topic 14

2. apply the XPath projection to the article  $A$ ; this returns  $e_1, e_2, \dots, e_n$  elements.
3.  $g = f$
4. while  $g! = nil$  do
  - if ( $g == e_i$  for any  $e_i$ )
  - then return the XPath of  $g$  and exit
  - else calculate  $g = parent(g)$
5. if (there are  $e_i$  that are descendants of  $f$ )
  - then return all of those and exit
  - else return the  $e_i$  (if any)

After our initial submission, we looked at improving the order of our final answers. We identified key terms in the projection, in the example of topic 14 “Corba”, “Figure”, and “Architecture”. By globally ranking the extracted fragments into tiers based on how many of the key terms appear in the projected elements, irrespective of how many times they appear and ignoring upper and lower-case differences.

### 3.5 Query Translator

The query translator constructed queries that we could process with our search engine and extractor. Figure 6 shows the query that was automatically generated for topic 14.

The following process was developed by analysing the structure of the topics in order to deduce the semantics of the various possible constructs in a topic, particularly the <Title> of a topic.

The <cw> and <ce> elements in the title of the topic are used to generate the selection component of the query. Mappings, similar to those described in Section 3.2 for the indexing, are used to map paths within <ce> elements to PADRE fields.

If there is more than one field specified by the paths within a <ce> element, then all possible combinations of the field mappings from the <ce> with terms from the <cw> must be generated in the query.

When content element involves dates, we use the metadata field “d” and convert the <cw> element into constraints on numerical dates. Similarly we attempt to identify phrases using location of commas in topic, so as to take advantage of the phrase feature of PADRE.

The <te> target element if present is translated into the projection component of the query. When the path in the projection maps to a field also used in the selection component additional constraints should be added to the projection.

## 4 Experiments and Results

We submitted three runs to INEX:

1. queries on *full* articles
2. queries on *split* articles
3. *manually* constructed queries on split articles

Subsequently we also explored:

4. queries on split articles with *post-projection fragment reranking*

Results for these four runs on topic 14 are shown in Figure 7, 8, 9 and 10. These graphs show relevance judgements for the 100 highest ranked answers for each run. Each answer corresponds to a vertical bar of about 2mm width. The highest ranked answer appears on the left. The height of the vertical bars represents the degree of relevance, and the greylevel the coverage.

Although in some cases the manually constructed queries performed better than the automatically generated queries using the query translator this did not occur for topic 14. There was a clear trend that using the collection containing documents and document fragments was more effective than using just the full documents. This is very clearly borne out with topic 14, but better performance is seen with other queries as well. It is unclear whether the post-projection reranking of fragments is effective as many unjudged elements were returned.

For comparison we have also included the optimal ranking in Figure 11 which shows there is still considerable room for further improvement in XML retrieval.

A key question that the INEX experiments has not addressed is do users want to get back documents fragments or are they more interested in pointers to relevant parts within actual documents. This raises questions about what constitutes an answer and how answers should be organised when presented to the user.

## References

- [1] N. Craswell, D. Hawking, A. Krumpholz, I. Mathieson, J. A. Thom, A.-M. Vercoustre, P. Wilkins and M. Wu. XML document retrieval with PADRE. In *Proceedings of the 7th Australasian Document Computing Symposium*, page To appear, Sydney, Australia, 16 December 2002.
- [2] CSIRO and Australian National University. Panoptic enterprise search engine. <http://www.panopticsearch.com/>.
- [3] David Hawking, Peter Bailey and Nick Craswell. Efficient and flexible search using text and metadata. Technical Report TR2000-83, CSIRO Mathematical and Information Sciences, 2000. <http://www.ted.cmis.csiro.au/~dave/TR2000-83.ps.gz>.
- [4] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu and M. Gatford. Okapi at TREC-3. In D. K. Harman (editor), *Proceedings of TREC-3*, Gaithersburg MD, November 1994. NIST special publication 500-225. <http://trec.nist.gov/pubs/trec3/papers/city.ps.gz>.
- [5] 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*, pages 311–317, Dublin, Ireland, July 3-6 1994. Springer-Verlag.



Figure 7: Results for Topic 14 — query on full articles

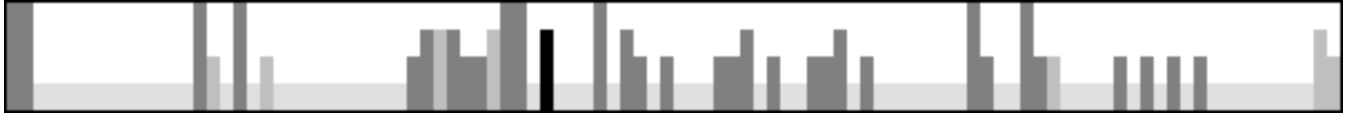


Figure 8: Results for Topic 14 — query on split articles



Figure 9: Results for Topic 14 — manual query on split articles



Figure 10: Results for Topic 14 - query on split articles with further ranking of final answers



Figure 11: Results for Topic 14 - optimal ranking (relevance only)

E	E - exact coverage
S	S - too small coverage
L	L - too large coverage
N	N - no coverage
not judged	not judged

# JuruXML – an XML retrieval system at INEX'02

Yosi Mass, Matan Mandelbrod, Einat Amitay, Yoelle Maarek, Aya Soffer

IBM Research Lab

Haifa 31905, Israel

+972-3-6401627

{yosimass, matan, einat, yoelle, ayas}@il.ibm.com

## ABSTRACT

XML documents represent a middle range between unstructured data such as textual documents and fully structured data encoded in databases. Typically, information retrieval techniques are used to support search on the “unstructured” end of this scale, while database techniques are used for the other end. To date, most of the work on XML query and search has stemmed from the structured side and is strongly inspired by database techniques. We describe here an approach that originates from the “unstructured” end and is based on augmentation of information retrieval techniques. It is specifically targeted to support the information needs of end-users, more specifically user friendliness via an intuitive querying mechanism, and ranking of results for approximate needs. We describe our query format and ranking mechanism and demonstrate how it was used to run the INEX topics.

## Keywords

XML, Information Retrieval.

## 1. INTRODUCTION

To date, most of the work on XML query and search has stemmed from the document management and database communities and from the information needs of business applications, as evidenced by existing XML query languages such as W3C's XQuery [1], which is strongly inspired by SQL. We propose here to extend the realm of XML by supporting the information needs of users wishing to query XML collections in a flexible way without knowing much about the documents structure. Rather than inventing a new query language, we suggest to query XML documents via pieces of XML documents or “XML fragments” of the same nature as the documents that are queried. We then present an extension of the vector space model for ranking XML results by relevance.

We have extended Juru[4], a full-text information retrieval system developed at the IBM Research Lab in Haifa, to handle XML documents. INEX provided a most useful framework to evaluate the capabilities of our query format and ranking methods. We show that despite its relative simplicity we were able to express 58

out of the 60 INEX topics and discuss limitations that prevented us from expressing the other two.

The rest of the paper is organized as follows. Section 2 introduces our query format and mechanism. Section 3 shows how the INEX topics were translated to this format. Section 4 proposes various ranking approaches and Section 5 provides some implementation details of our system. We conclude in Section 6 by describing our three INEX runs.

## 2. THE QUERY FORMAT

As stated above, we propose to tackle the XML search issue from an information retrieval (IR) perspective, and thus support the information needs of users wishing to query XML collections in a flexible way without knowing much about the documents structure. In a classical IR system, the document collection consists of “free-text” documents and the query is expressed in free text. We claim that the same can hold for XML collections and we suggest to query XML documents via pieces of XML documents or “XML fragments” of the same nature as the documents that are queried. Returned results should be not only perfect matches but also “close enough” ones ranked according to some measure of relevance.

One key element of this work is to avoid defining yet another complex XML query language but rather to allow users to express their needs as fragments of XML documents, or XML fragments for short. Users should not need to reformulate their queries as they may become too specific. The ranking mechanism should be responsible for giving priority to the closest form. This approach of using a very simple “fragment-based” language rather than SQL-like query languages (e.g., XQuery [1]) is somewhat analogous to using free-text rather than Boolean queries in IR: less control is given to the user, and most of the logic is put in the ranking mechanism so as to best match the user's needs.

### 2.1 Query syntax

XML fragments are portions of XML, possibly combined with free text, which can be viewed as a tree<sup>1</sup>. Documents that contain the query or part of it as a subtree are returned as results (see examples in Section 2.2). XML attributes are queried using the same syntax used in the XML documents<sup>2</sup>.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
*Conference '00*, Month 1-2, 2000, City, State.  
Copyright 2000 ACM 1-58113-000-0/00/0000...\$5.00.

<sup>1</sup> We add an artificial root node that encloses the whole query so as to make it a valid XML data

<sup>2</sup> As an alternative, attributes can be queried as if they were children node of their containing node.

The default semantic of a query is that a document/component is considered a valid result if it contains at least one path of the query tree from the root to a leaf (see examples below), or to follow the vector space model, if it has a non-null similarity with the query profile.

In order to allow for more control on the XML fragments and yet still keep their simple intuitive syntax, we augment the XML fragments with the following symbols:

- **+/-:** a +/- prefix can be added to elements, attributes and content. Prefixing an element with a “+” operator in the XML fragment means that the subtree below the node associated with this element should be fully contained in any retrieved document. Prefixing an element with “-” means that the sub tree below the node associated with the element, should not exist in any retrieved document. For example:
  - `<Book><Title>-Graph Theory</Title></Book>` as a query, will return all books whose title contains the word “theory” but not the word “graph”.
  - `<Book><-Abstract></Abstract></Book>` will return all books that do not contain abstracts.
- **“...” (phrase) :** Users can enclose any free text part of the XML fragment between quotes (“”) to support phrase match.
- **At least one:** An exception to the regular + operator behavior occurs when it is applied to two or more sibling elements of exactly the same type (i.e., having the same name). In this case, the semantics of + is that **at least one** of the subtrees below one of those sibling nodes must hold even if they have some internal + nodes (see example in Section 2.2.3)

## 2.2 Query examples

### 2.2.1 Task: Find books written by John.

Users with no knowledge of the documents DTD or schema, may simply issue a query in pure free text of the form “books written by John”. However, if they have some basic knowledge of the DTD, their query can become:

```
<book>
  <author>John</author>
</book>
```

One key contribution of our technique is that the structured query does not need to express a “perfect” need, rather we allow for approximate matching. Thus for the above query, the system would also assign a non-null score to documents containing a fragment of the form below.

```
<book>
  <fm><author><first>John</first></author></fm>
</book>
```

### 2.2.2 Task: Find books written by John Doe

```
<+book>
  <author>John Doe</author>
```

```
</book>
```

In this example, `<+book>` imposes the constraint that there be an instance of `<author>` that contains both John and Doe under the same `<author>` instance. Thus the + avoids results in which there are two different authors one with `<fm>`John and the second with `<snm>`Doe. The above syntax is similar to

```
<book><+author>John Doe</author></book>
and to
<book><author>+John +Doe</author></book>
```

### 2.2.3 Task: Retrieve all articles from the years 1999-2000 that deal with works on nonmonotonic reasoning. Do not retrieve articles that are calendar/call for papers

```
<article>
  <bdy> <sec>+"nonmonotonic reasoning"</sec> </bdy>
  <hdr>
    <yr>+1999</yr>
    <yr>+2000</yr>
  </hdr>
  <tig> <atl>-calendar</atl> </tig>
</article>
```

In this example, we have two sibling `<yr>` nodes labeled with +. This means that a valid result should contain at least one of the years 1999 or 2000.

## 3. INEX QUERY TRANSLATION

We describe how we translated the INEX topics into our query format. Note that the translation rules specified here are systematically applied to all queries. Their purpose is to capture the semantics of the INEX topics format (See its DTD in Figure 1) so as to best express it in our formalism.

```
<!ELEMENT INEX-Topic
<Title,Description,Narrative,Keywords)>
<!ATTLIST INEX-Topic
  topic-id          CDATA   #REQUIRED
  query-type        CDATA   #REQUIRED
  ct-no             CDATA   #REQUIRED
>
<!ELEMENT Title (te?, (cw, ce?)+)>
<!ELEMENT te    (#PCDATA)>
<!ELEMENT cw    (#PCDATA)>
<!ELEMENT ce    (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Narrative (#PCDATA)>
<!ELEMENT Keywords (#PCDATA)>
```

Figure 1: INEX topics format

We decided to consider only the `<Title>` and `<Keywords>` tags of the topic and ignore the `<Description>` and the `<Narrative>` ones.

### 3.1 CO topics translation

For CO topics we systematically applied the following translation rules:

- If there is only one word under the `<cw>` tag, we add it to the query with an implicit +, together with the words under the `<Keywords>` tag.
- If there are only two words under the `<cw>` tag, we add them to the query with an implicit phrase augmented with a + operator, together with the words under the `<Keywords>` tag.
- If there are more than 2 words under `<cw>` we simply add them to the query and ignore the `<Keywords>` part.

In the first two cases, we are guaranteed that result candidates will contain the words under <cw> (via the + operator) and adding the words under the <Keywords> part simply improves ranking. In the last case, we do not add the keywords, since the query is long enough to be expressive in itself and since we want to guarantee that the results contain at least some of the <cw> decorated words. The words under the <Keywords> tag may add noise, therefore we ignore them.

### 3.2 CAS topics translation

For CAS topics we applied similar rules as for the CO topics as follows:

- For each <cw><ce> pair:
  - If there is only one word under <cw>, we add it to the query with an implicit + under all nodes that appear in the <ce> tag
  - If there are only two words under <cw>, we add them to the query with an implicit phrase augmented with a + operator under all nodes that appear in the <ce> tag
  - If there are more than two words under <cw> we add them to the query under all nodes that appear in the <ce> tag
- For <cw> without a <ce> tag apply the CO rules described above.
- Add the words under the <Keywords> part to the query as free text

For example, lets consider the INEX **topic 5**, as expressed in Figure 2 below:

```
<Title>
  <te>tig</te>
  <cw>QBIC</cw><ce>bibl</ce>
  <cw>image retrieval</cw>
</Title>
<Keywords>
QBIC, IBM, image, video, content query, retrieval
system
</Keywords>
```

Figure 2: INEX topic 5

According to the above rules, it is translated into:

```
<article>
  <bibl>+QBIC</bibl>
  +"image retrieval"
  QBIC. IBM. image. video. "content query" . "retrieval system"
</article>
```

We assume some knowledge of the semantics of the INEX documents DTD and systematically apply the “at least one” rule for “years” and “authors” elements, as illustrated in topic 15 (see Figure 3).

```
<Title>
  <te>article/bm/bib/bibl/bb</te>
  <cw>
  hypercube, mesh, torus, toroidal,
  non-numerical, database
  </cw>
  <ce>article/bm/bib/bibl/bb</ce>
  <cw>1996 or 1997</cw>
  <ce>article/fm/hdr/hdr2/pdt</ce>
```

```
</Title>
<Keywords>
  1996 1997 hypercube mesh torus toridal
  non-numerical database
</Keywords>
```

Figure 3: INEX topic 15

This topic is translated into the following fragment form:

```
<article>
  <bm><bib><bibl><bb>
  hypercube. mesh. torus. toroidal. non-numerical.
  database.
  </bb></bibl></bib></bm>
  <fm><hdr><hdr2>
  <pdt>+1996</pdt>
  <pdt>+1997</pdt>
  </hdr2></hdr> </fm>
  1996 1997 hypercube mesh torus toridal non-numerical
  database
</article>
```

Note that according to our syntax, result candidates need to contain at least one of the years 1996 or 1997.

### 3.3 Target elements (<te> tag)

We distinguish between topics that contain a <te> tag and those that do not. In the first case, the elements under the <te> tag that match the query are returned as results. In the second case the search engine is left the freedom to decide whether it should return the entire document and/or the most relevant components. The decision is based on the ranking requirements and depends on the granularity level at which statistics (e.g. term frequency) are stored. We discuss our implementation in section 5.1.1 below

### 3.4 Limitations of our format

The proposed XML Fragments format is clearly not as expressive as a full-fledged SQL-like query language. However, our conjecture is that it covers most of users needs in querying XML collections and reduces significantly the complexity of the language. This is similar to free-text queries that provide less expressive power than complex Boolean queries, but provide sufficient expressiveness for most users’ needs. We verified this hypothesis in the INEX evaluation, as we could easily express 58 out of the total 60 INEX topics.

We could not express Topic 14, which states “Find figures that describe the Corba architecture and the paragraphs that refer to those figures”. This type of query requires a kind of “join” operation between two elements (or tables in database terms) “figures” and “paragraphs” which should be joined through a common “figure-id” field.

Another Topic that we could not express using our query format was Topic 28, which states “Retrieve the title of articles published in the Special Feature section of the journal ‘IEEE Micro’”. This topic depends on the order of sibling nodes (requires <articles> to appear after a specific <sec1> node). Our query format is expressed as an XML tree and thus cannot express relations that depend on node ordering. We could express topic 28 if the <journal> was organized such that <article> nodes are children of <sec1> nodes, as specified below:

```
<journal>
```

```

<title>...</title>
<sec1>
  <title>...</title>
  <article>...</article>
  <article>...</article>
</sec1>
</journal>

```

## 4. RANKING APPROACHES

In this section we discuss two possible approaches for combining the structured and unstructured portions of the query in terms of ranking. Let us remind here that a typical ranking model for IR is the vector space model where documents and queries are both represented as vectors in a space where each dimension represents a distinct indexing unit  $t_i$ . The coordinate of a given document  $D$  on dimension  $t_i$ , is denoted as  $w_D(t_i)$  and stands for the “weight” of  $t_i$  in document  $D$  within a given collection. It is typically computed using a score of the *tf x idf* family that takes into account both document and collection statistics. The relevance of the document  $D$  to the query  $Q$ , denoted below as  $\rho(Q, D)$ , is then usually evaluated by using a measure of similarity between vectors such as the cosine measure (Formula 1).

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

Formula (1)

We describe now two ranking methods for XML documents: one that weights each individual context and one that merges all contexts that match a query term. We have tested the two ranking methods in two different INEX runs and will use the INEX assessment results to verify which method is better.

### 4.1 Assigning weights to individual contexts

The first approach, which extends the vector space model, is described in details in [5]. The idea is to use as indexing units not single terms but pairs of terms of the form  $(t_i, c_i)$ , where  $t_i$  is the textual part or term and  $c_i$  is the path leading to it from the document root (the context). We allow “approximate matching” so that a term  $(t_i, c_i)$  in the query can match several actual terms of the form  $(t_i, c_k)$  in the documents. For example, a query term *(John, /author)* can match *(John, /fm/author/fnm)* and *(John, /bm/author/fnm)*. For each query term  $(t_i, c_i)$ , we denote its weight in the query as  $w_Q(t_i, C_i)$ , the weight of each resembling context in the documents as  $w_D(t_i, C_k)$ , and the resemblance measure between the contexts as  $cr(c_i, c_k)$  (see an example function in Section 5.2.2).

Thus, in order to measure the similarity between XML fragments and XML documents we extend (Formula 1) to (Formula 2) below:

$$\rho(Q, D) = \frac{\sum_{(t_i, c_i) \in Q} \sum_{(t_i, c_k) \in D} w_Q(t_i, C_i) * w_D(t_i, C_k) * cr(C_i, C_k)}{\|Q\| * \|D\|}$$

Formula (2)

We impose that  $cr()$  values range between 0 and 1, where 1 is achieved only for a pair of perfectly identical contexts. Thus, we see that (2) is identical to (1), in the special case of free-text where there is one unique default context.

## 4.2 Merging contexts

Recall that for each query term  $(t_i, c_i)$ , we can find a set of document terms  $(t_i, c_k)$  such that each  $c_k$  resembles the given context  $c_i$ . As an alternative approach, instead of weighting the resemblance between  $c_i$  and all its  $c_k$ 's, we consider merging all such  $c_k$ 's and treating them as equally good from the user's perspective. The merged context is assigned a weight as a function of the details the user gave in her query, which is independent of the distance between the query context and the document contexts. Denoting  $w(C_i)$  as the weight of the context  $c_i$  (see an example function in 6.2), our ranking formula becomes:

$$\rho(Q, D) = \frac{\sum_{(t_i, c_i) \in Q} w_Q(t_i) * w_D(t_i) * w(C_i)}{\|Q\| * \|D\|}$$

Formula (3)

## 5. IMPLEMENTATION – THE JuruXML SYSTEM

We have extended a full-text information retrieval system Juru [4], developed at the IBM Research Lab in Haifa so as to support the XML fragment query format and the above ranking mechanisms. We describe now the modifications we applied, for this purpose, to the indexing and to the retrieval processes.

### 5.1 Indexing stage

At indexing time, XML documents are parsed using an XML parser. A vector of  $(t, c)$  pairs is extracted to create the document profile where  $t$  is the textual part or term and  $c$  is the path leading to it from the document root (i.e., the context). In addition we store for each XML tag  $\langle t \rangle$  a pair  $(\_s\_t, c)$  for the tag start and  $(\_e\_t, c)$  for the tag end. By storing terms with their contexts, the posting-list of term  $t$  that encapsulates all occurrences of  $t$  in all documents, is split into separate posting lists, one posting list for each of the contexts in which  $t$  occurs. This splitting allows the system to efficiently handle retrieval of occurrences of a term  $t$  under a specific context  $c$ . For efficiency we map each context to a contextId, which can be stored as an integer.

We use a scheme first introduced in [6], for navigating XML collections and implemented in the XMLFS system that allows to store such pairs  $(t, c)$  in the lexicon of a regular full-text information retrieval system via only minor modifications: each pair  $(t, c)$  is presented to the indexer as a unique key  $t\#c$ . At retrieval time, the system can identify the precise occurrences of the term  $t$  under a given context  $c$  in the collection, by fetching the posting list of the key  $t\#c$ . Juru [4] stores all index terms (that form the lexicon of the system) in a *Trie* data structure (see for example [2]) and therefore all contexts under which the term  $t$  has been stored can easily be retrieved by suffix matching of “ $t\#$ ”

#### 5.1.1 Component statistics

As described in the previous section, the terms we store in the index are of the form  $t\#c$  where  $t$  is a word and  $c$  is the context

leading to the term from the document root. This allows us to query for content under a specific context and to return a specific component as a result. However, Juru[4] tracks statistics (e.g., term frequency) at the document level, therefore relevance can be evaluated only at the document level. This means that all components in a retrieved document will be assigned the same relevance score and thus the same ranking (namely the document’s ranking).

In order to allow ranking at a granularity level other than the full document level, it is possible to define at indexing time a list of elements whose associated fragments will be indexed as separate entities. This allows for statistics to be tracked at the indicated level of granularity, and to score results at the same granularity. While this approach works well for CO like queries, it does not perform as well for queries that specify a combination of contexts since these contexts may reside in different indexing entities.

We will investigate in future work how to support various levels of granularity in one index. In the meantime, for the INEX collection, we used a fixed granularity of <sec> for CO topics.

## 5.2 Retrieval stage

As described above, the query is expressed as a combination of XML fragments and possibly free text. In order for queries to be expressed as valid XML, we encapsulate the query within a pair of <root></root> tags, which have no semantic meaning and are removed at a later stage. We parse queries with a standard XML parser in order to obtain a set of terms in context of the form  $t\#c$ , in the same way as we parsed the original XML documents. The query algorithm is described below:

1. Parse the query and create list of terms of the form  $t\#c_i$  and a ‘ResultTree’ (Section 5.2.3) for post query filtering
2. Expand each term ( $t\#c_i$ ) to relevant terms ( $t_i\#c_k$ ) that resemble it from the index (see Section 5.2.1) and assign a weight to each term (Section 5.2.2)
3. Issue a regular Juru query formed by the expanded terms
4. Rank results according to one of the methods described in Section 4.
5. Filter results based on the ‘resultTree’ (see section 5.2.3)

**Figure 4: Query algorithm**

We detail each of the key steps of the algorithm in the following sections.

### 5.2.1 Query expansion

Let us illustrate the expansion with the example below. Consider the query:

```
<article>
  <bib>QBIC</bib>
</article>
```

It is parsed into “qbic#/article/bibl”. We execute suffix matching (thanks to the trie structure) on “qbic#” and get all the contexts under which the word qbic was indexed. An example of such a context is “/article/bm/bib/bibl/bb”. We now have to check which of them is relevant to the query. In our current implementation, we consider only the contexts for which the query context is a subsequence. Therefore, “/article/bm/bib/bibl/bb” is a relevant context since it includes “/article/bibl” as a subsequence. Note that we allow for gaps in the inclusion. At the end of this step we

have a set of terms of the form  $t\#c$ , which are now sent to Juru as a free text query.

### 5.2.2 Context resemblance function

We assign weights to each of the expanded terms. Let  $t\#c$  be a query term and let  $t\#ec$  be an expanded term. Since in our current implementation we consider only contexts that contain the query context as a subsequence, we define

$$cr(c, ec) = \frac{(1 + |c|)}{(1 + |ec|)}$$

where  $|c|$  is the number of tags in the given query context and  $|ec|$  is number of tags in the expanded context. Thus, in the above example,

$$cr(“/article/bibl”, /article/bm/bib/bibl/bb”) = 3/6 = 0.5$$

It is easy to see that  $0 < cr \leq 1$  and it is equal to 1 if and only if the query context is identical to the expanded context. We are currently investigating an alternate approach that can also handle expanded contexts that do not contain the query context by looking at LCS( $c, ec$ ) (Longest Common subsequence) instead of full containment.

The above weighting methods assume no knowledge of the DTD of the documents. For the INEX collection, we added a bonus factor for what we consider important tags. For example, we increase the computed weight of expanded contexts that contain titles (e.g. /fm/atl, sec/st).

### 5.2.3 The ResultTree

The last step in the retrieval process is to filter results based on the query tree structure and based on the +/- modifiers. We keep a *ResultTree*, which represents the query tree structure. For each of the potential matched results, (which are already sorted by decreasing relevance), we inject their matched terms instances into the *ResultTree*<sup>3</sup>. Through the *ResultTree* we perform the following verifications:

- ‘+’ nodes appear and ‘-’ nodes do not appear
- Check for common context instances. For example, in a query of the form <+au>John Doe</au>, verify that both John and Doe appear under the same <au> instance<sup>4</sup>.
- Check for phrase nodes and “at least one” nodes
- Ensure that the result contains at least one path of the query tree from the root to a leaf.

## 6. INEX RUNS

We conducted three INEX runs. For the first two runs, we applied the automatic query translation rules specified before, while in the 3<sup>rd</sup> run we performed some manual editing of the query attempting to better fit the topic’s <Description>.

<sup>3</sup> For each term, we store in the posting list the offset of the particular instance of the term in the document.

<sup>4</sup> We verify common instances through terms’ offsets saved in the posting lists.



## 6.1 First run – assigning weights to individual contexts

In the first run we employed the ranking method of formula (2) where each expanded term was assigned a weight based on its similarity to the query context using the formula described in Section 5.2.2

## 6.2 Second run – merging contexts

In the second run, we employed the ranking method of formula (3) where the weight function for context  $c$  was

$$w(c) = (|c| + 1)$$

For example, the weight of the context in the query term “qbic#/article/bibl” is 3. We boost weights of contexts that contain tags we consider important similarly to the *Context Resemblance function* (section 5.2.2)

## 6.3 Third run – manual editing

In this run we tried to exploit our query format capabilities by manual editing some of the queries based on their description. Let us consider for instance topic 18 as given in Figure 5.

```
<Title>
  <te>article</te>
  <cw>Hypertext Information Retrieval</cw>
  <ce>article</ce>
  <cw>Hypertext Information Retrieval</cw>
  <ce>bib/bibl/bb/at</ce>
</Title>
<Description>
Retrieve articles on hypertext information
retrieval where the bibliography contains works
with the words "hypertext", "information" and
"retrieval" in at least one of the citations.
</Description>
```

Figure 5: INEX topic 18

This topic was translated for the first two runs into:

```
<article>
  Hypertext Information Retrieval
  <+bib><+bibl><+bb><+at>
  Hypertext Information Retrieval
  </at></bb></bibl></bib>
</article>
```

While it was expressed, in the third manual run as

```
<article>
  Hypertext Information Retrieval
  <+bib><+bibl><+bb><+at>
  Hypertext Information Retrieval
  </at></bb></bibl></bib>
</article>
```

The only difference between these expressions is that in the latter form, a <+bib> is added in order to force all three words *Hypertext Information Retrieval* to appear under some same instance of a <+bb> tag. The manual run returned only 5 such results, while the first 2 runs returned 100 results most of them containing only some of the required words under the same <+bb> item.

## 6.4 Generating the submission format

An INEX submission consists of a number of topics, each identified by a topic ID. A topic's result consist of a number of result elements as in the example below (we omit full format due to space limitation. It can be obtained from [3])

```
<result>
  <file>tc/2001/t0111</file>
  <path>/article[1]/bm[1]/ack[1]</path>
  <rsv>0.67</rsv>
</result>
```

In JuruXML a match is identified by its offset in the document. To generate the above format we parse again the XML document that contains the match and while counting offsets until the match's offset we build the requested <path> info.

## 7. CONCLUSION

The INEX framework allowed us to experiment with the expressiveness of the XML fragments query format. We showed that using, this rather simplistic query format, we could express 58 out of the 60 INEX topics. We tested two ranking methods one that assigns different weights to term occurrences under different contexts and one that merges occurrences of document terms that match a query term. Once the assessments are returned, we will evaluate the usefulness of the proposed methods for combining IR ranking for free text with XML structure ranking.

## 8. ACKNOWLEDGMENTS

We would like to thank David Carmel for his support of the core Juru engine and insight on ranking mechanisms.

## 9. REFERENCES

- [1] XQuery – The XML Query language, <http://www.w3.org/TR/2002/WD-xquery-20020430>
- [2] Donald E Knuth, The art of computer programming: sorting and searching (vol 3), Addison Wesley, 1973.
- [3] Initiative for the evaluation of XML retrieval <http://qmir.dcs.qmul.ac.uk/INEX/>
- [4] D. Carmel, E. Amitay, M. Herscovici, Y. Maarek, Y. Petruschka and A. Soffer, "Juru at TREC 10 - Experiments with Index Pruning", In Proceedings of NIST TREC 10, Nov 2001.
- [5] 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.
- [6] A. Azagury, M. Factor, Y. Maarek, B. Mandler "A Novel Navigation Paradigm for XML Repositories", pp 515-525 in ACM SIGIR'2000 workshop on XML and IR, SIGIR Forum, 2000.
- [7] Y. Maarek, F. Smadja "Full text indexing based on Lexical relations: An application: Software libraries" Proceedings of the 12<sup>th</sup> international ACM SIGIR conference on Research and Development in Information Retrieval, pp 198-206, 1989

# Naive clustering of a large XML document collection\*

Antoine Doucet  
Department of Computer Science  
P.O. Box 26 (Teollisuuskatu 23)  
00014 University of Helsinki  
Finland  
antoine.doucet@cs.helsinki.fi

Helena Ahonen-Myka  
Department of Computer Science  
P.O. Box 26 (Teollisuuskatu 23)  
00014 University of Helsinki  
Finland  
helena.ahonen-myka@cs.helsinki.fi

## ABSTRACT

In this paper, we address the problem of clustering a homogenous collection of text-based XML documents. We present some experiments we have led on clustering the INEX<sup>1</sup> structured document collection. Our claim is that element tags provide additional information that must help improve the quality of clustering. We have implemented and experimented various ways to account for document structure, and used the well-known k-means algorithm to validate these principles.

## Keywords

Document Clustering, XML, Information Retrieval

## 1. INTRODUCTION

Document clustering has been applied to information retrieval following the **cluster hypothesis**, which states that relevant documents tend to be highly similar to each other, and subsequently they tend to belong to the same clusters[3]. The theory behind this is that document clustering should permit to improve the effectiveness of an IRS by permitting to recall more of the relevant documents; Notably, in a best-match approach, some very relevant documents might receive a low rank simply because they miss one of the keywords of the query. Based on the cluster hypothesis however, these documents are to be clustered together with the best-ranked documents and can be found this way [1]. Document clustering can be performed prior to the query, in which case it is used to form a document taxonomy similar to that of the well-known “Yahoo” search engine. An alternative application of document clustering to IR is **post-retrieval clustering** [11], which is not performed on the whole document collection, but solely on the candidate subcollection to

\*This work is supported by the Academy of Finland (project 50959; DoReMi - Document Management, Information Retrieval and Text Mining)

<sup>1</sup>Initiative for the evaluation of XML Retrieval (<http://qmir.dcs.qmw.ac.uk/inex/>)

be answered to a query. In this case, the clustering is used to ameliorate the quality of the final answer.

Nowadays, Internet is a repository for huge amounts of data. The quantity of XML data shared over the World Wide Web is increasing drastically. A large majority of this XML data has been database-centered, but text-centered XML document collections are now getting more and more frequent. As a consequence, it became necessary to provide means to manage these collections. This can be done by automatically organizing very large collections into smaller subcollections, using document clustering techniques. Unfortunately, most of the research on structured document processing is still focused on database-centered XML (see for example [2] and [13]).

In this paper, based on the conjecture that “As structure is supplementary information to raw text, there must exist a way to use it, that improves the clustering quality”, we present various naive approaches to represent text-centered XML documents (section 2) and experiment them using a well-known partitioning clustering algorithm. The results presented in section 3 are emphasizing the difficulty of this task and calling for discussion of the results and a description of the eventual directions of our future work (section 4).

## 2. PROCEDURE OF THE EXPERIMENTS

### 2.1 Document representation

As a representation of the documents, we have used the vector space model. In this representation, each document is represented by an  $N$ -dimensional vector, with  $N$  being the number of **document features** in the collection. In most approaches, the features have been the most significant words of the collection. All the words are not selected as features, as the number of dimensions of the vector would easily place the computational efficiency at stake. For this reason, in the case of very large document collections, **feature selection** techniques are applied. We have used three different feature sets along our experiments: text features (i.e., words), tag features, and finally a combination of both.

- “Text features only”: For the text feature set, as the size of the document collection is very large, we have used a few feature selection techniques. First, we have ignored words of less than three characters, and used a **stoplist** to delete longer words with a weak dis-

criminative power (such as articles, pronouns, conjunctions and auxiliary verbs). We also pruned all words containing a numerical character. This simple heuristic diminished the feature set of about 50,000 word terms! The last step has been to **stem** the words, that is, to reduce them to a canonical form (for example, 'brought' 'bring' and 'brings' can be reduced to 'bring'), using the Porter algorithm[8]. The resulting set contained 188,417 features.

- “Tag features only”: The clustering method we are willing to develop for clustering structured documents aims to be general. Therefore, we have made the choice to not manually group any tag labels. In practice, this means that all tag labels are distinct (e.g., 'ssl' and 'ss2' for sub-section of level 1 and 2 are distinct). The only preprocessing we made was to pruning the closing tags, as we decided to account as much for 'complete' tags (with both a starting and an ending tag) as for the non-closed ones (e.g., 'art', 'entity', 'colspec'). Finally, we found 183 different tag features.
- “Text+tags”: This last method combines both feature sets, by simply merging them. The total number of features is then 188,600.

The document vectors were then filled in with normalized tf-idf measures. Tf-idf combines term frequency (tf) [6] and inverted document frequency (idf) [4]. Term frequency is simply the number of occurrences of the feature words in a document. Its weakness is that it does not take the specificity of the terms into account. A term which is common in many documents is less useful than a term common to only a few documents. This is the motive for combining this measure with the inverse document frequency of a term, which divides the total number of documents in the collection by the total number of documents where the term occurs. In short, term frequency is a measure of the importance of a term in a document and inverted document frequency is a measure of its specificity within the collection.

## 2.2 Similarity measure

Clustering techniques group items based on their pairwise similarity. Thus, the first task is to find the right similarity measure. Following the vector space model, two measures are commonly used. The first one is the Euclidean distance, which has the advantage of being easily understandable. The other frequent measure is the cosine similarity. Its strength is very efficient computation for normalized vectors, since in that case  $\cosine(\vec{d}_1, \vec{d}_2)$  simplifies to  $(d_1 \cdot d_2)$ . Because their results are very similar in nature, cosine similarity can be preferred to Euclidean distance (see for example [14]).

## 2.3 Clustering technique

There are two main families of clustering algorithms. Given  $n$  documents, hierarchical clustering produces a nested sequence of partitions, with a single cluster containing all documents at the top, and  $n$  singleton clusters at the bottom. This result can be displayed as a dendrogram (a subclass of the tree family). In partitional clustering, where **k-means** is the most common technique, the number  $k$  of desired clusters is either given as input, or determined as part of the

### 1. Initialisation:

- $k$  points are chosen as initial centroids
- Assign each point to the closest centroid

### 2. Iterate:

- Compute the centroid of each cluster
- Assign each point to the closest centroid

### 3. Stop condition:

- As soon as the centroids are stable

**Figure 1: Base k-means algorithm**

process. The collection is initially partitioned into clusters whose quality is repeatedly optimized, until a stable solution is found.

In general, hierarchical clustering has been considered as the best quality clustering approach, and its quadratic complexity seen as its main weakness. For large documents, the linear time complexity (w.r.t. the number of documents) of partitional techniques has made them more popular. This is especially true for IR systems where the clustering is often aimed to improve the system's efficiency. Furthermore, Steinbach et al. [10] have made large scale experiments with numerous datasets and evaluation metrics which finally pointed out as a result that the cluster-quality of the bisecting k-means technique was at least as good as that of the hierarchical approaches they tested. In these experiments, we have decided to use the k-means algorithm, both for its linear time complexity and the simplicity of its algorithm.

Given a number  $k$  of desired clusters, k-means techniques provide a one-level partitioning of the dataset in linear time ( $O(n)$  or  $O(n \log n)$ ) where  $n$  stands for the number of documents[12]). The *base* algorithm presented in figure 1 assumes the number of desired clusters be given and relies on the idea that documents are seen as data points.

## 2.4 Discussion on evaluation

### 2.4.1 Internal and External Quality.

There are two main families of quality measures. The **external** quality measures use an (external) manual classification of the document classification, whereas the **internal** quality measures do not. The principle of an external quality measure is to compare the clustering to existing testified classes. The better the clustering and the classification “match”, the better the external quality measure evaluates the clustering.

In this work, we have used entropy and purity, two frequent external quality measures.

- The **entropy** is an information theoretic measure presented by Shannon [9]. It measures how the classes (manually tagged) are distributed within each cluster. This provides a quality evaluation for un-nested clusters (for hierarchical clustering, this means an entropy value can be computed only per level of the dendro-

gram). Note from the nature of entropy that the optimal score is obtained with singleton clusters and thus it can hardly be used to compare clustering solutions of different sizes.

The technique consists of first calculating the class distribution of the document collection, that is the number of documents in each class. The entropy of each cluster  $C$  is based on the probability that a document of  $C$  belongs to each class. The overall entropy is the average per cluster entropy weighted by the size of each cluster.

- The **purity** of a cluster measures how much that cluster is “specialised” in a class. It is simply its largest class divided by its size. The overall purity of a clustering solution is then a weighted average of the purity of each of its individual clusters.
- There exist many more measures. For example, the well-known IR F-measure has been adapted to clustering [5]. We did not use it, however, as it is by definition adapted for the case where the evaluation classes are query answers (this evaluation method was used with various TREC collections and their assessment results).

Internal quality measures are used when no manual classification is provided. They are computed by calculating average inter- and intra-cluster similarities. An example of an internal quality measure is **cohesiveness** (a.k.a. “overall similarity”), which is defined for each cluster as the average similarity between each two documents of that cluster.

#### 2.4.2 The INEX case

Our experiments compare the use of different feature sets. As such, they result in different pairwise document similarity values. Thus, it is clear that estimating the feature sets based on inherent internal quality measures would not make any sense. Therefore, we must use external quality measures. Nevertheless, any external quality measure relies on an existing manual classification, and at the time being, the only classification existing for the INEX collection is the journal volume in which an article was published. For more consistency, we have actually used the journals as our classes. We have also made another class of the 125 volume descriptions, which contain a listing of the articles published in that volume.

Unfortunately, this classification has a number of problems. The main issue is that these classes form a partition of the document collection, that is, the classes are disjoint. This property is rather unappropriate for document collections, as there exist no such strict border between two articles as there may be with other data types. The fact that an article was published in a given journal rarely means that it could not have been published in another one. Hence, the journal title classification is probably too strict.

In fact, a good classification for evaluating document clustering is typically a manual assessment of the answers to a set of queries. By using the topics of an IR evaluation initiative (e.g., TREC or INEX) as classes, and the corresponding documents as the elements of the class, researchers have often

found a satisfying way to evaluate the quality of clustering methods. These classes offer a more trustable human-expert classification, that furthermore allows a document to belong to many classes or none.

Finally, the clusterings have been evaluated according to the 18 journals where the documents were published, plus an additional volume class. Finally, the 12,232 documents of the INEX collection have been mapped to 19 classes.

Of course, in order to keep the experiments fair, all document parts containing the name of the journal in which it was published was pruned. In practice, this means the elements <doi>, <fno> and <hdr> and their content were ignored.

### 3. RESULTS

We have implemented and experimented the techniques described above on the INEX collection, using the publicly available clustering tool implemented by George Karypis, University of Minnesota<sup>2</sup>.

We have run the k-means method with  $k \in \{5; 10; 15; 20; 25; 35\}$  for text-only, tags-only, tags and text. We have then computed entropy and purity using the journal titles as classes.

The results of our experiments for 5, 15, 20 and 35 clusters are shown respectively in tables 1,2,3, and 4. The runs were computed on a 1333 Mhz desktop with 1 gigabyte of memory.

**Table 1: Results of k-way clustering for k=5**

Features	Text	Tags	Text + Tags
Entropy	0.711	0.836	0.812
Purity	0.301	0.211	0.216
Clustering Time	150s.	4s.	160s.

**Table 2: Results of k-way clustering for k=15**

Features	Text	Tags	Text + Tags
Entropy	0.633	0.798	0.678
Purity	0.379	0.228	0.372
Clustering Time	754s.	11s.	837s.

**Table 3: Results of k-way clustering for k=20**

Features	Text	Tags	Text + Tags
Entropy	0.598	0.775	0.677
Purity	0.413	0.237	0.332
Clustering Time	1101s.	15s.	1191s.

#### 3.1 Including all tags decreases quality!

A clear observation is that, for any desired number of clusters  $k$ , the best quality is obtained with the text features. Tag features as a stand-alone perform much worse, and when they are combined to the text features, the worsening is just averaged.

<sup>2</sup>CLUTO, <http://www-users.cs.umn.edu/~karypis/cluto/>

**Table 4: Results of k-way clustering for k=35**

Features	Text	Tags	Text + Tags
Entropy	0.568	0.758	0.612
Purity	0.454	0.254	0.385
Clustering Time	2016s.	25s.	2215s.

However, one may expect that adding an extra piece of information about documents would improve their description, and subsequently their pairwise similarity measures, and should finally result in a better clustering quality.

There are various reasons for this quality worsening following the inclusion of tag features. First, the quality evaluation issue must be recalled; For example, using the tag features, a few clusters have terms like ‘`tmath`’ or ‘`math`’ as their most descriptive features. They mainly gather articles from the journals “Transactions on Computers” and “Transactions on Parallel & Distributed Systems”. This predominance of two different classes implies low external quality measures. It is however impossible to claim as a consequence that those clusters are uninteresting. On the other hand, some clusters are doubtlessly negatively affected by the addition of the tag features. Document clusters dominated by style features (e.g., ‘`b`’ or ‘`tt`’) are rather grouping documents based on their authors’ writing style. As an illustration, those clusters are almost equally distributed amid the classes.

### 3.2 “Tags only” permits very fast clustering

The clustering based solely on tag features is computed much faster. This is no surprise as the number of items is then 183, when it is 188,417 for text only. What is surprising is how good the tags-only results are, considering that the whole process runs in seconds on a standard desktop.

In applications involving a huge number of documents, and requiring fast clustering (e.g., “prior to query” document clustering for IR), the trade-off between quality and efficiency may advantage the tags-only option.

It is however difficult to tell, besides the raw quality scores, how well the tag features clustering are matching the “text only” clustering. A good question to ask is how close are these clustering solutions? We know that the tags-only clustering performs reasonably well with respect to its computational efficiency, but how close is this good answer to the better answer issued from the “text-only” clustering? Unfortunately, this is still in the list of future work!

### 3.3 Exception: the ‘volume’ class

For each clustering, most of the 125 volume.xml files, compiling entity references to the articles of a given volume of a journal, are found within the same cluster. In contradiction with the general observation that text features give higher quality clustering, we have found that for this specific cluster, “text+tags” and “tags only” give the best performance. In table 5, we have computed values for recall and precision for this ‘volume’ cluster. Precision is the number of ‘volume’ documents found in the volume cluster (true positives), divided by the size of that cluster. Recall is the number of ‘volume’ documents found in the volume cluster, divided by the total number of volume documents (i.e., 125).

**Table 5: Results of k-way clustering for the ‘volume’ cluster with k=15**

Features	Text	Tags	Text + Tags
Precision	28%	99%	100%
Recall	100%	100%	100%
Entropy	0.722	0.016	1
Purity	0.295	0.992	0
Internal Similarity	0.094	0.900	0.912
Clustering Time	754s.	11s.	837s.

**Table 6: 3 most descriptive features within the ‘volume’ cluster for k=15**

Text only	january (19%)	society (13%)	publish (6%)
Tags only	<entity>(63%)	<title>(20%)	<sec1>(14%)
Text+tags	<entity>(63%)	<title>(20%)	<sec1>(15%)

This result is due to the very specific structure of the volume files. They contain the list of the titles of all articles published in the corresponding journal volume. This type of documents totally misleads the text features approach, as in this case the most specific features are not article titles, but various details about the publishing (month of publication for example). We have computed the most descriptive terms for the volume cluster, for each of the three feature sets in table 6. The descriptivity measure for a feature within a cluster is the percentage of internal similarity that is due to this particular feature. When the feature set contains element labels (i.e., tags), they tend to dominate the text features, as a consequence of a more discriminant distribution.

### 3.4 Best clustering method, with respect to journal title classes

Following these results, we foresaw a better clustering method, based on the principle to use the tag features clustering as a preprocessing. The idea is to pre-detect those documents which are structurally different. This is harmless from a computational point of view, as the tag features are so few, and since their extraction is done in linear time.

Eventhough the “text+tags” performs slightly better, the efficiency/quality trade-off obviously plaid for preferring the tag-feature clustering as the preprocessing for the simple clustering method we present right below.

- Step 1: k-means clustering of the full document collection based on the “tags-only” representation. The  $n$  clusters with an average internal similarity above a threshold (say 0.9) are kept.
- Step 2: A  $(k-n)$ -means clustering is then led, based on the remaining documents (all those that do not belong to one preselected cluster).

Only the volume cluster is preselected. The results are shown in table 7 and confirm the clustering quality improvement. However, we are aware that such a method can not be claimed to be superior, before further experiments are made (particularly using different collections).

**Table 7: Text features based clustering with and without tag features pre-clustering, for k=15**

	Text features	Same, but with pre-clustering
Entropy	0.633	0.630
Purity	0.379	0.394
Time	754s.	11+742s.

Anyhow, we believe that the general idea to use structure-based clustering as a preprocessing of standard clustering must permit to improve the clustering quality. But to extend the application of this principle to the general case, we are willing to consider more general and sophisticated structural similarity measures. A recent work has notably provided an edit tree distance between the structure trees of XML documents [7].

#### 4. DISCUSSION AND CONCLUSION

We addressed the problem of clustering homogenous structured document collections. We experimented a common partitioning clustering algorithm with various sets of features. As the current evaluation system is not yet reliable, the results we found can not be considered as definitive, but should rather be seen as hints.

Our results have then hinted that simply adding tag labels to the feature set does not improve the clustering quality. However, our conjecture that a way to exploit the structure exists is still standing. What the first results emphasize is that the solution is not straightforward, and that combining structural similarities to content similarity indeed permits to improve the clustering quality.

It seems like computing tf-idf measures of tag labels is insufficient, and we are now considering more sophisticated measures for structural similarity between documents. Instead of using the frequency of the elements, options are to weight the documents with the total size of the elements (or with their average size). It would still remain to be decided whether the size of an element should be defined locally or as the total size of its sub-elements, in which case normalization issues would emerge.

So far, this work has been difficult due to the lack of a very large text-centered structured document collection. The INEX initiative has already provided such a collection, but meaningful classes are not yet available at the time being. As soon as the manual assessments of the retrieval systems of the INEX collection will be released, we will be able to further evaluate the various structured document clustering approaches. In this regard, we will also need more document collections, so as to make sure that the results we get are not statistical accidents, due to some specificity of the INEX data.

There are various possible research directions. One is to develop feature selection methods for tag labels. Some simple ways might be to replace words by their full path expressions, or by their local path expressions. It would also make sense to develop ways to detect different classes of tag labels. The distinct nature of some of these classes would then call for different processing techniques. It is clear, for example,

that the tags 'tfmath', 'sgmlmath', and 'math' have much in common and that they may probably be merged to a single 'meta-math' class. For the least, we must try to account for the fact that 'tfmath' and 'sgmlmath' are more similar than 'sgmlmath' and 'ss1' (subsection of level 1).

Another interesting problem emerges, following the work in the INEX initiative: multi-level clustering. The idea is to compute representations of document sub-elements together with the documents, and give as a result clusters containing items of different granularities. This idea is clearly derived from the IR problem posed by INEX, of retrieving the best matching elements, rather than full documents exclusively.

#### 5. REFERENCES

- [1] B. Croft. *Organizing and searching large files of document descriptions*. PhD thesis, University of Cambridge, 1978.
- [2] D. Guillaume and F. Murtaugh. Clustering of XML Documents. *Computer Physics Communications*, 127:215–227, 2000.
- [3] N. Jardine and C. van Rijsbergen. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval*, 7:217–240, 1971.
- [4] K. S. Jones. A Statistical Interpretation of Term Specificity and Its Application in Retrieval. *Journal of Documentation*, 28(1):11–21, 1972.
- [5] B. Larsen and C. Aone. Fast and Effective Text Mining Using Linear-time Document Clustering. In *Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, California*, pages 16–22, 1999.
- [6] H. P. Luhn. A statistical approach to mechanical encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317, 1957.
- [7] A. Nierman and H. Jagadish. Evaluating Structural Similarity in XML. In *Fifth International Workshop on the Web and Databases (WebDB 2002), Madison, Wisconsin*, 2002.
- [8] M. Porter. An algorithm for suffix stripping. *Program*, 14:130–137, 1980.
- [9] C. E. Shannon. A mathematical theory of communication. *Bell System Tech*, 27:379–423, 623–656, 1948.
- [10] M. Steinbach, G. Karypis, and V. Kumar. A Comparison of Document Clustering Techniques. In *Proceedings of KDD 2000, Workshop on Text Mining*, 2000.
- [11] A. Tombros. *The effectiveness of hierarchic query-based clustering of documents for information retrieval*. PhD thesis, University of Glasgow, 2002.
- [12] P. Willett. Recent trends in hierarchic document clustering: a critical review. In *Information Processing and Management*, 24(5):577–597, 1988.

- [13] J. Yi and N. Sundaresan. A classifier for semi-structured documents. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, Massachusetts*, pages 340–344, 2000.
- [14] Y. Zhao and G. Karypis. Criterion functions for document clustering. Technical report, Department of Computer Science and Engineering, University of Minnesota Twin Cities, 2001.

# XML Retrieval by Extreme File Inversion

## Shlomo Geva

School of Software Engineering and Data Communication  
Faculty of Information Technology  
Queensland University of Technology  
GPO Box 2434 Brisbane Q 4001 Australia  
[s.geva@qut.edu.au](mailto:s.geva@qut.edu.au)

### Abstract

In this paper we describe the implementation of an extreme variation to the inverted file scheme. The scheme supports a comprehensive set of Boolean search operators, down to the single character level. When combined with a heuristic document ranking algorithm it supports retrieval of raw XML data, using the embedded tags as search arguments. We tested the system against a set of XML queries and the entire set of IEEE Computer Society publications 1995-2002, in XML format.

### Keywords

XML Retrieval, Text Retrieval, Pattern Matching, Partial Match Search, Proximity Search.

### 1. Introduction

Associative Memory, a memory that is accessed by content rather than by address, is an idea that has been a subject of research by the computer industry for many years. Access methods for text retrieval and for partial match search have also been the subject of intensive research. Such systems usually provide adequate performance in keyword based searches. However, in recent years there has been an increased effort to extend the support to Information Retrieval in a broader sense and to support higher level search operations. For example, when searching for partially matching documents, when ranking documents according to user information needs, or when processing natural language queries.

Most existing database systems are designed to handle commercial applications, where the types of queries are anticipated and the data itself is well structured and very carefully controlled. The emphasis is almost always on database Integrity. Physical data organization techniques are designed to handle queries with suitable speed. With the advent of the Internet and the World Wide Web much less control over data organization and integrity can be exercised. Furthermore, there is an ever-increasing requirement for systems to handle queries or produce reports that were not anticipated in detail. Text retrieval systems were the immediate and natural technology to address the problem. However, despite great advances in

the past decade in the technology of search engines and text retrieval, a truly satisfactory solution is still unavailable. The annual TREC conference proceedings provide ample evidence of the difficulty involved when text retrieval systems are extended to support *Information Retrieval*.

The XML scheme provides a compromise between the fully structured predetermined database schema, and the unstructured and unpredictable nature of heterogeneous documents and collections. While the physical structure of the XML document remains only loosely defined, the XML document is not undisciplined – it contains self-identifying data elements (in the form of XML tags). Neither conventional database systems nor text retrieval systems were designed to handle such data organization. Therefore, considerable effort is currently undertaken to come up with information retrieval systems for XML collections that are able to take advantage of the XML tags.

Most database systems support multiple access paths to records or relations by the use of indexes or other more sophisticated text retrieval techniques. A query language such as SQL supports powerful search capabilities. The difficulty with such systems in the context of distributed data repositories is the rigid requirement with respect to a database schema. The recent trend, to move towards XML representation, does not altogether lend itself to treatment with conventional database technology, nor is it fully supported by text retrieval systems. Almost invariably there will be some ad hoc queries which will not be supported to a satisfactory level by the data structure or hardware with regard to functionality or response time. This paper describes an attempt to combine the functionality of an inverted file system, pushed to the extreme (as will be explained in detail in the following section), with a higher level heuristic search algorithm, to support complex queries on a large XML database.

The remainder of this paper is organized as follows: Section 2 describes the extreme file inversion method and how it differs from the conventional approach. Section 3 describes the basic principles of extreme file inversion. Section 4 describes how the file store requirements are minimized. Section 5 describes how EFI was used to evaluate the INEX



XML topics. Section 6 presents results of evaluation during INEX 2002.

## 2. Extreme File Inversion

Inverting a file is an old and proven technique to facilitate fast access to records using inverted lists. A fully inverted file is a file for which inverted lists exist for each field (or each word). Such a file structure facilitates access to records based on any (attribute, value) pair and complex queries using Boolean operators can be efficiently implemented. File inversion is common in Text Retrieval applications, where word locations within documents are also maintained to facilitate proximity text searching operations on free-text fields, such as phrase searching or context searching.

Our Extreme File Inversion (EFI) data structures and algorithms were developed in 1985 as a response to a specific pattern matching need of a user with large text collections. Conventional text retrieval systems do not support sub-word search arguments (at least not efficiently). EFI is a conceptual variation of the inverted file designed to overcome this problem. EFI is based on two major modifications. The first deviation from conventional methods is the total separation of the semantics of content and internal record structure from the structure of the index (inverted lists). Each record is simply regarded as an array of characters. For the purpose of file inversion a record of  $k$  characters is regarded as consisting of  $k$  one-character long attributes. For each character an inverted list is created such that all the pointers to records having a given character value in a given character position, may be found by obtaining the corresponding list. With a character set of  $n$  characters the total number of inverted lists for the file is  $k * n$ .

To summarize, rather than invert the file by the values of the attributes, the file is inverted by the values contained in character positions. This representation is almost devoid of any knowledge about the records (documents) contents and structure.

The second deviation from the conventional method (at least back in 1985 when it was devised) is the implementation of the inverted lists. Rather than maintain a pointer array, (a list of record keys in each list), an array of bits, or a bitmap, where one bit is associated with each record in the file, is maintained. Although the use of bitmaps was hardly new even then, the application of bitmaps together with file inversion by character provided a very powerful tool for data searching.

## 3. Search Operators

In this section we describe retrieval algorithms. In passing we mention a full set of efficiently implemented search operators; however, for the sake of brevity we restrict ourselves to a more detailed description of only a few operators that are relevant to XML retrieval.

For each character position in the data record one bit map is maintained for every character in the character set. To refer to a specific bitmap the notation  $X.n$  is used throughout this paper, where  $X$  is a character value in the implementation character set, and  $n$  is a character position within the data record. For example,  $A.12$  identifies the bitmap for the character 'A' in position 12 within the data record.

For a given bitmap,  $X.n$ , Those bits which are set to 1 are associated with records which contain the character value  $X$  in the character position  $n$ . Bits that are set to 0 are associated with those records that do not contain that value in that position. The ordinal position of any bit in a bitmap is the ordinal position of the record it is associated with, in the file. In order to access a given bitmap we generally require one direct disk access.

Clearly, users express queries in terms of field names rather than in terms of character positions. Therefore, the internal structure of data records is defined in a dictionary. For each record a number of fields are defined. Each field is characterized by the following parameters:

*{ name, position, length, word-size }*

The meaning and usage of each of the field definition parameters is explained in detail later on in this section. This simple definition of fields, giving sections of the data record - identified by start position and length – a name by which they can be referenced in queries, allows the user to select on elementary fields, group fields, arrays or the entire record.

The implementation of the following list of selection operator types is facilitated by the data structure:

- Equal, Starts With, Ends With
- Greater Than, Greater Than or Equal
- Less Than, Less Than or Equal
- Within Range
- Contains
- Min, Max, Total

The selection specification rules also allow the use of some 'special' characters: ? - the wild card character, and \* - the *elastic* wild card. In addition to these, the logical operators AND, OR, and NOT are easily implemented in the query

syntax to allow complex Boolean conditions to be specified.

### 3.1 The *Startswith* operator

The simplest selection criterion to evaluate is the selection on a single character. The field name and value are entered in the query, e.g.

***GENDER SW M***

The field name is used to look up the dictionary record description to determine the field position in the data record. The field position parameter specifies the position of the first character of the field within the record. If the character position of the gender field within the data record is 324, then the bitmap is identified as ***M.324***

To select all the records for the query above, the bit map identified as ***M.324*** is obtained. Those bits in the bitmap that are set to 1 correspond to data records that contain the character ***M*** in character position ***324***.

The process of evaluating the ***SW*** condition is only a little more complex where selection is applied to fields which are more than one character long, e.g.

***COLOUR SW RED***

If the ***COLOUR*** field starts in position 732 within the data record, and is 3 characters long, then ***R.732***, ***E.733***, and ***D.734*** identify the bitmaps for the literal ***RED***. The next step is a bit-wise ***AND*** operation, performed in a serial fashion on the bitmaps, to produce a result bit map, expressed as :

***R = R.732 & E.733 & D.734***

where the ***&*** represents the bit-wise ***AND*** operator. Any bit in ***R*** that is set to 1 points to a data record in which the ***COLOUR*** field starts with the value ***RED***.

When the character ***?***, the wild card character, appears in a literal, it masks out a single character, in the corresponding character position, e.g.

***NAME SW SM?T***

This leads to selection, for example, of records where ***NAME*** starts with ***SMITH***, ***SMYTH***, ***SMET*** etc. The implementation of this feature is straight forward: the character position masked by the wild card is ignored.

Multiple key queries are also easily implemented as in the query:

***COLOUR SW WHITE OR GREEN***

The implementation of the ***OR*** and ***AND*** operators requires the application of the corresponding bit-wise operator to the bitmaps resulting from each of the individual queries. In this example only 10 I/O operations are required to satisfy the query (one I/O per bitmap).

To make the query language even more powerful, the 'elastic' character, ***\****, is easily implemented. When the elastic character appears in a literal, it is interpreted as zero or more occurrences of a wild card, for example, the query:

***NAME SW R\*D***

is interpreted, by expanding up to a predefined field width, as:

***NAME SW RD OR R?D OR R??D OR R???D ...***

### 3.2 The *Equal* operator

The ***EQUAL*** operator is similar to the ***SW*** operator but also checks for trailing spaces in a field. When the ***EQUAL*** operator is applied to an alphanumeric field, the literal specified in the query is padded with trailing spaces before evaluation begins. For example, the query

***NAME EQ SMITH***

where ***NAME*** is a 12 characters field , is evaluated by adding trailing spaces :

***NAME EQ "SMITH "***

and the bitmaps corresponding to the trailing spaces are used during evaluation to ensure that records where names like ***"SMITHY"*** or ***"SMITH JOHNS"*** appear are not selected.

### 3.3 Text Searching

Fields of type text, are fields which may contain more than a single word. The idea behind the implementation process described here is that a text field can be treated as if it were an array of words.

#### 3.3.1 Word Alignment

In an array, all elements start on an element boundary. Text fields can be transformed to exhibit a similar property, by ensuring that words in the text start on a 'word-boundary'. The transformation is aligning words in text fields, on a word boundary, such that every word in a text field starts in a particular character position, which is an integer multiple

of a predefined *word-size*, and by doing so, generating a 'word aligned' field.

The *word-size* is a small integer, related to the average size of a word in the language used in the text. It is the equivalent of the size of an array element except that words in the text are only required to start on a predefined alignment, but may extend into the next 'element' and cross a word-boundary.

### 3.3.2 The STARTSWITH Operator and Text

Applying word-alignment to text fields allows a more efficient search for records where the text field contains a word which STARTSWITH the specified search string.

For example, if NAME occupies character positions 1-300, aligned on a 5-character boundary, then the condition

*NAME STRATSWITH MAC*

is expressed as

$$R = ( M.1 \ \& \ A.2 \ \& \ C.3 ) | \\ ( M.6 \ \& \ A.7 \ \& \ C.8 ) | \\ \dots \\ ( M.296 \ \& \ A.297 \ \& \ C.298 )$$

## 4. File Structures

Ideally the data is stored in a relative or direct file, where each record is identified by its ordinal position in the file. The data may however reside on any other type of direct access file.

The bitmaps file requires a direct access mechanism and several options are available. Because of the intensive I/O operations on bitmaps during query evaluation it is essential to minimize access time. Any record access mechanism that requires considerably more than one physical I/O to retrieve a record is not attractive.

A memory resident index, which is loaded into main memory at system start-up, allows for direct access to bitmaps without incurring any additional I/O at run time. This provides for *exactly* 1 physical to 1 logical I/O. However, one may question if this is a feasible solution, as the main store requirements may be prohibitive. To answer that question we can calculate the size of the index

$$I = l * m * 4$$

where *l* is the (fixed) record length in the data file, *m* is the size of the character set employed, and we assume that 4 bytes are sufficient to hold a bitmap address.

For a file of 1,000,000 records, having record size of 512 characters, and the ASCII character set of 64 displayable characters, the file size will be about 500 Mbyte, and the index table size will be 320 Kbyte. For an application running on a PC this is a feasible figure, and the assumption of 1 physical to 1 logical I/O is realistic.

### 4.1 Bitmaps file store requirements

During the bitmaps load process, *n* bit maps are created for every character position in the data record, where *n* is the number of characters in the character set. With a character set of 64, each character in a data record is reflected in 64 bitmaps ( as a 1-bit in one bit map, and as a 0-bit in all the other.) Each character in the data file requires only 8 bits storage (assuming no compression.) Therefore the overhead in file store is 8 times the size of the data file. This seems rather expensive, but after compression, discussed in the next section, the overhead is reduced to an acceptable level.

### 4.2 Bitmap compression

The Zero-run-length technique is used to compress a bit map by creating an array of bytes, where a run-length of 0-bits separating 1-bits is encoded by a single byte. The value of 255 is reserved to indicate a zero only run of 255 bits not followed by a 1-bit. This allows for zero-runs of more than 255 bits to be encoded on several bytes.

Note that compression of bitmaps with a ratio of less than 1:8 of 1-bits to 0-bits, will result in having a compressed version which is in fact larger in size than the original. In such cases, of course, compression is not applied. We have addressed the possibility of using more or less than one byte to encode a run-length, but it turns out to provide only marginal compression gains, and increases the CPU load.

This compression scheme reduces the size of the bitmaps considerably. In fact, for a random character distribution in the data records, the size of the compressed bit maps file is approximately equal to the size of the data file. Consider a character set of cardinality 64 and a random character distribution. On average, only one bit in 64, in each of the bit maps, will be set to 1. Since encoding of a zero-run of length 63 requires only one byte, the compression will produce a reduction in size by a factor of  $64 / 8 = 8$ . This is exactly the overhead figure we obtained earlier. This result is not surprising since after all the bitmaps store the same information as the data file, except in a different representation.

How does the zero-run-length scheme perform in practice? Our INEX2002 data file in word-aligned uncompressed ASCII representation occupies 750Mbyte while the

Bitmaps file occupies 650Mbyte. The overhead is about 87%.

While fixed length records are required for file inversion, there is no reason to actually store the data file itself in a fixed length record format. It is only during file inversion that a temporary file with fixed length records is needed, so that this overhead cannot be put onto the account of EFI. It allows one to de-normalize a database file structure to generate an extract file for the purpose of efficient searching by EFI, without the need for costly join operations. This technique is obviously more suitable to static databases.

## 4.2 INEX XML File Structure

Since the INEX data set contains Journal articles of various formats, record lengths, and sizes, we had to convert it to a suitable format for EFI. For lack of time we applied brute force – each of the articles was scanned and transformed into a flat file of 500 characters wide records. Lines were split pretty much arbitrarily, except that we did take care not to split atomic units - where possible. So, an <author> XML unit, for instance, was kept on the same line, and words were not split. However, some paragraphs exceeded 500 characters, and were split into several lines. Text was also word-aligned during this process.

This arbitrary split is not ideal, but it still allowed the search engine to search effectively, as our results demonstrate. We hope to improve on this with more time on our hands.

In addition to the above, each line was also prefixed with document details corresponding to the text line. Specifically, we kept the full document path, thereby preserving journal, year, and article information.

It is important to note that we inverted the entire XML collection, tags and all. With this we were able to issue queries which take into account embedded XML tags. For instance, to find instances of the surname **Geva** we issue the query:

```
Text equal "<snm> geva"
```

## 5. Document Retrieval and Ranking

The INEX 2002 XML retrieval task consists of 60 XML Topics. An XML Topic could not be evaluated as such by our search engine. Each topic had to be transformed into a set of EFI search engine queries. Furthermore, the results of the corresponding set of queries had to be consolidated to provide a ranked list of documents, as described in the following sections.

### 5.1 Transforming Topics into EFI Queries

Each of the INEX XML Topics consists of four elements: <title>, <description>, <narrative>, and <keywords>. We have only used the <title> and <keywords> in our system.

The basic strategy was to extract keywords and word-phrases from the <title> and <keywords> elements, and apply a separate search for each word-phrase and keyword. Our transformation preserved context information by explicitly including XML tags as search arguments. Note that all the transformations were done by a single computer program in a pre-processing step, with no manual intervention. All topics were pre-processed by the same program.

Consider the following topic <title> element

```
<Title>
  <cw>description logics</cw>
  <ce>abs, kwd</ce>
</Title>
```

This topic was transformed to produce the following queries:

- 1) text = "description logics" and text = "<abs"|"<kwd"
- 2) text = "description" and text = "<abs"|"<kwd"
- 3) text = "logics" and text = "<abs"|"<kwd"

The reason that we obtained 3 separate queries is that the INEX topic specification does not support the specification of a word-phrase as distinct from a set of keywords. In this instance we had to try all possibilities. Where the topic specified word phrases explicitly, we did not expand the search to single keywords. For example, the element <cw>software engineering survey, programming survey, programming tutorial, software engineering tutorial</cw> produced only 4 word-phrase queries because commas were used to separate phrases (our parser looks for commas, quotes, and other cues for phrases).

During query evaluation however, if a word-phrase is found to occur more than once, the component keyword queries for the phrase are not executed. This is an automated runtime decision. The assumption that we made is that if a word-phrase is frequent then chances are that the user meant the phrase rather than a list of keywords.

The <keywords> element is treated in a similar manner to <title>, except that there is no explicit XML context element.

### 5.2 Ranking Documents

The results of EFI queries correspond to raw XML text lines in the articles. It is necessary to combine query results in order to rank a given document. We apply a simple heuristic weighting to query results to produce a weighted sum rank for each document. The documents are then

sorted by descending rank. We have used the following heuristic approach:

- A weight is associated with each query. It is the inverse of the number of lines that match that query. This is justified by the observation that often queries involve common words and are more likely to be matched, but are less indicative.
- Query weights are totaled for each *document* that they match. A single line in a document could be matched by more than one query and a document may have many matching lines. The sum total of the weights is the document rank.
- Documents are ranked in descending order. The highest ranked document is that matching more of the (weighted) queries generated by a topic than any other document.

## 6. Experimental Results

The system appears to identify relevant documents for most queries. One component of the retrieval system that we have not yet completed is the identification of target elements. This however can be done *after* matching documents were retrieved. Our system always returned the front matter (<fm>) of matching documents assuming this is sufficient to identify the article, and usually includes an abstract (It could have returned the entire article of course). The relative performance of this system is yet to be determined, following the INEX conference. This will be discussed in detail in the final version of the paper.

## REFERENCES

- [1] Geva, S., "Implementing a Software Associative Memory", Thesis (1987), QUT

# INEX 2002 Experiments

PRELIMINARY NOTES ON THE USE OF A COMMERCIAL SEARCH ENGINE

**Richard M. Tong, Ph.D.**

*rtong@tgncorp.com*

TARRAGON CONSULTING CORPORATION

1563 Solano Avenue, #350, Berkeley, CA 94707, USA

## 1. Introduction

Tarragon Consulting Corporation (Tarragon) used a commercial search engine from Verity, Inc.<sup>1</sup> to perform all the indexing and search functions for the INEX 2002 experiments.

Our goal for INEX 2002 was to use the "out of the box" Verity engine to provide a baseline against which we will be able to test a range of techniques for search and retrieval of XML documents that we have been investigating. Our ultimate objective is to provide these capabilities as add-ons to the standard Verity product offering.

In developing queries for each of the 30 content only and content plus structure topics, we attempted to emphasize precision at the expense of recall. However, since the standard Verity engine does not provide a mechanism for returning pointers to the specific elements that match the search criteria, we had to use a path reporting strategy that selects either the first, or the smallest unique, element that contains the matched element(s). In general, of course, this has the effect of depressing both the recall and precision scores.

The remainder of this workshop paper includes a short description of the indexing strategy we used for the INEX documents, brief details of the techniques we used for constructing queries for each of the INEX Topics, and our initial comments on the overall experiment.

## 2. Document Indexing

We used Verity's built-in "zone indexing" mechanism to create a complete inverted keyword index, together with a set of auxiliary indexes that allow us to restrict searches within zones. The zones in this case correspond exactly to the set of XML tags defined by the IEEE DTD.

This enables us to issue queries of the form:

```
"QBIC" <IN> bb1
```

which the engine interprets as a search request to look for the keyword QBIC in the element defined by the pair of <bb1/> tags.

The Verity Query Language (VQL) supports nested zone queries so that expressions of the form:

```
"ibm" <IN> aff <IN> fm
```

can be used to capture a <cw/><ce/> constraint like:

```
<cw>ibm</cw><ce>fm//aff</ce>
```

in a direct way.

## 3. Content Only Queries

We used a semi-automatic technique for constructing queries from the Content Only (CO) topics.

The first step was to run a simple Perl script to extract a list of terms and phrases from the <Title/>, <Description/> and <Keywords/> elements in each query. We then manually post-processed this list to remove "noise" terms and phrases. Then finally, using the edited list and a simple template, we automatically generated a VQL content expression corresponding to the original topic.

So, for example, CO Topic 31 looks, in part, like:

```
co_topic_31 <Accrue>
* 0.50 "computational biology"
* 0.50 "bioinformatics"
* 0.50 "genome"
* 0.50 "genomics"
* 0.50 "proteomics"
* 0.50 "sequencing"
* 0.50 "protein folding"
```

where <Accrue> is the VQL operator that implements a basic evidence summation function, and the weights 0.50 define the relative

<sup>1</sup> See, <http://www.verity.com/>

contribution of each term or phrase. For the simple template used in the INEX baseline experiments, we assigned all terms and phrases the same weight.

Not shown above are the zone restrictions we used in the baseline query template. In all CO queries we restricted the search to the `<bdy/>` zone, so that the actual VQL expressions used are of the form:

```
"computational biology" <IN> bdy
"bioinformatics" <IN> bdy
etc.
```

Each CO query was executed against the indexed collection and the list of matching document IDs returned. We used another Perl script to format the results for submission. So, for example, the first part of the results file for Topic 31 has the form:

```
<topic topic-id="31">
  <result>
    <file>ex/2001/x6014</file>
    <path>/article[1]</path>
    <rank>1</rank>
    <rsv>0.94</rsv>
  </result>
  <result>
    <file>ex/2001/x6008</file>
    <path>/article[1]</path>
    <rank>2</rank>
    <rsv>0.91</rsv>
  </result>
  <result>
    <file>ex/2000/x2020</file>
    <path>/article[1]</path>
    <rank>3</rank>
    <rsv>0.90</rsv>
  </result>
  ...
</topic>
```

Note that here, and for all the other CO queries, we chose to report the result path as `/article[1]`.

## 4. Content and Structure Queries

We used a similar semi-automatic strategy for constructing queries from the Content and Structure (CAS) topics. The basic difference being that we mapped all the `<cw/><ce/>` constraints into VQL zone expressions and then conjoined them with the content based VQL expressions.

Each CAS query thus has the form:

```
cas_topic_xx <And>
* cas_xx_constraints
* cas_xx_contents
```

and so, for example, the constraints for CAS Topic 08 looks like:

```
cas_08_constraints <And>
* "ibm" <IN> aff <IN> fm
* "certificates" <IN> sec <IN> bdy
```

Each CAS query was executed against the indexed collection and the list of matching document IDs returned. As for the CO topics, we used a Perl script to format the results for submission, but in this case included a topic specific path. As noted above, the standard Verity engine does not return a pointer to the element(s) that match the query expressions, so we finessed this point by hand-coding a path for each topic.

Of the 30 CAS topics, seven of them have `<te/>` elements that are unique, so in these cases we used the `<te/>` element specified in the topic. In 14 other topics, we were able to assume a unique element. So, for example, in Topic 01 we simply reported the first author (i.e., we used the path `/article[1]/fm[1]/au[1]`, since there is always at least one author). And for the remaining nine topics, we selected the smallest unique element guaranteed to contain the retrieved element. In many cases, of course, this was just the path `/article[1]`.

## 5. Overall Comments

We found the overall INEX 2002 experiment cycle to be an extremely worthwhile exercise. It certainly helped us achieve the first part of our goal, that is to understand the limitations of the standard Verity engine. Unfortunately, time and resource constraints prevented us from testing the planned extensions, but we hope to run at least a selection of additional experiments after the INEX Workshop and before the final papers are due.

As part of the "lessons learned" during the effort, we feel strongly that the assessment and results scoring procedures need further investigation. For example, it is not clear to us that it really is possible to treat relevance and coverage as independent concepts. And we also believe that the very different nature of the information needs expressed by the CO and CAS topics, argues for the use of different scoring algorithms for the two sets of results.

We look forward to reading the discussion on these and other issues at the INEX Workshop.

# An IR Approach to XML Retrieval based on the Extended Vector Model

Carolyn J. Crouch  
Department of Computer Science  
University of Minnesota Duluth  
Duluth, MN 55812  
(218) 726-7607  
ccrouch@d.umn.edu

S. Apte  
Department of Computer Science  
University of Minnesota Duluth  
Duluth, MN 55812  
(218) 726-7607  
apte0002@d.umn.edu

H. Bapat  
Department of Computer Science  
University of Minnesota Duluth  
Duluth, MN 55812  
(218) 726-7607  
bapa0005@d.umn.edu

## ABSTRACT

The authors describe their approach to XML retrieval based on the extended vector space model of Fox [3]. Complete implementation of the system, using the Smart experimental retrieval system, is currently underway.

## Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Retrieval Models.

## General Terms

Design, Experimentation

## Keywords

XML retrieval, extended vector model

## 1. INTRODUCTION

When Vannevar Bush [1] first envisioned an ability to retrieve relevant information while sitting at his desk—that is, to have the data he sought immediately available at his fingertips—one could argue that he was in fact foreseeing the capabilities available to today's researchers through the development of facilities based on hypertext, multimedia, networking, and theoretical and applied research in information retrieval, among others. As XML becomes more dominant in the representation of web documents, it is a natural extension for information retrieval research.

When we first became familiar with the XML task posed by INEX, we were struck by the similarity of portions of the task to earlier work we had done [2] based on the extended vector model proposed by Fox [3]. Since our interests lie in information retrieval, we chose this approach for our initial investigations in XML retrieval.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

It seems safe to assume that everyone in the retrieval community is familiar with the vector space model [6], developed by Salton and used so successfully over the years by both his researchers at Cornell and others. This model is the basis of the Smart retrieval system, which evolved over a 30 year period under the direction of Salton, Buckley, and others. In the vector space model, each document (and query) is viewed as a set of unique words or phrases and is represented as a weighted term vector. The weight assigned to each term is indicative of the contribution of that term to the meaning of the document. The similarity between vectors (e.g., document and query) is represented by the mathematical similarity of their corresponding term vectors.

Fox [3], recognizing that the vector space model could be modified to include concepts other than the normal content terms, extended it as follows. He developed a method for representing in a single, extended vector different classes of information about a document, such as author name, terms, bibliographic citations, etc. Here a document vector consists of a set of subvectors, where each subvector represents a different concept class or c-type. Similarity between extended vectors is calculated as a linear combination of the similarities of corresponding subvectors. Subsequent work by both Fox and others [4, 2] focused on the problem of automatic generation of extended queries in this domain. (Of course, if we utilize the extended vector model for XML retrieval, this particular problem is no longer an issue because the query that is given can easily be translated into an extended vector query.)

The XML experiments are designed to handle two types of queries: the content-only (CO) query (the traditional query in information retrieval) and the content-and-structure (CAS) query. For CO queries, the retrieval system is expected to return a ranked list of the most relevant elements (article, paragraph, section, etc.). No target element is specified. For the CAS queries, the retrieval system should return a ranked list of elements specified in the target element (<te>) field, rather than a ranked list of documents. Search words themselves are specified in the <cw> element, and the context of the search words is specified in the context element (<ce>). In a relevant document, the search words in the <cw> element should occur in the element specified in the <ce>. Otherwise (if no <ce> is specified), the search words can occur anywhere in the document.



## 2. OUR APPROACH

In our approach, based on Fox's extended vector model, documents and queries are represented in extended vector form. The extended vector itself is a combination of subvectors, some containing normal text and others containing objective identifiers associated with the document. (Our current representation of an XML document/query consists of 18 subvectors.) For CO queries, we chose at this point to return a ranked list of documents. Keywords are not confined to a specific context, and we search for them throughout the document. The challenge for those using vector-based systems like ours is to deal with CAS queries, which consist of pairs of `<cw>`, `<ce>` elements. Consider, for example, the title section of CAS query 8:

```
<title>
    <te>article</te>
    <cw>ibm</cw><ce>fm/aff</ce>
    <cw>certificates</cw><ce>bdy/sec</ce>
</title>
```

In this case, the query is to return a ranked list of articles as specified by the target element `<te>`. The narrative specifies that the body or sections of relevant documents should contain information about the use of certificates for authenticating users on the Internet. And since the context of the content word *ibm* is *fm/aff*, the author(s) of those documents must be affiliated with IBM. Thus the query should retrieve only those articles on the use of certificates whose author(s) are affiliated with IBM.

The vector space model is not designed to handle this essentially Boolean query. Direct use of the extended vector model does not guarantee that each keyword will occur in the specified context. We deal with this issue by splitting the query into two parallel queries as follows:

Query 1: `<cw>ibm</cw><ce>fm/aff</ce>`

Query 2: `<cw>certificates</cw><ce>bdy/sec</ce>`

Affiliation and section are two different c-types. So query 1 searches for documents containing the objective identifier *ibm* in the affiliation subvector. Query 2 seeks documents whose section(s) contain the subjective identifier *certificate*. Our retrieval system (Smart) returns a ranked list of documents for both queries. The intersection of these lists is the final, ranked list of documents returned for query 8. Our retrieval is based on untuned *Lnu.ltu* [7] weighting of the collection.

## 3. CONCLUSIONS

Our efforts to date have been limited by the small size of our team, the substantial commitment required to produce the team's contribution to the XML project, the deferral of deadlines to meet the needs of the participants and the restrictions of the academic schedule. We will defer our discussion of results until the INEX tool for evaluation is available.

Our current efforts center on the implementation, within the extended vector model, of providing what is quite accurately referred to as "flexible retrieval"—i.e., "the retrieval over arbitrary combinations and nestings of element types"—by Grabs and Schek [5]. An excellent description of this task within the vector space model may be found in this reference. A related issue involves weighting within the local environment, weighting within the larger XML collection, and weighting among subvectors in the extended vector model.

## 4. REFERENCES

- [1] Bush, V. As we may think. The Atlantic online. <http://www.theatlantic.com/unbound/flashbks/computer/bush.html>.
- [2] Crouch, C. J., Crouch, D. B., and Nareddy, K. R. The automatic generation of extended queries. In Proc. of the 13<sup>th</sup> Annual International ACM SIGIR Conference, (Brussels, 1990), 369-383.
- [3] 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).
- [4] Fox, E. A., Nunn, G. L., and Lee, W. C. Coefficients for combining concept classes in a collection. In Proc. of the 11th Annual International ACM SIGIR Conference, (Grenoble, 1988), 291-307.
- [5] Grabs, T. and Schek, H. Generating vector spaces on-the-fly for flexible XML retrieval. <http://www-dbs.inf.ethz.ch/~grabs>.
- [6] Salton, G., Wong, A., and Yang, C. S. A vector space model for automatic indexing. *Comm. ACM* 18, 11 (1975), 613-620.
- [7] Singhal, A., Salton, G., Mitra, M., and Buckley, C. Document length normalization. In *IP&M* 23, 5 (1996), 619-633.

# Compression and an IR Approach to XML Retrieval (Draft)

Vo Ngoc Anh     Alistair Moffat

Department of Computer Science and Software Engineering  
The University of Melbourne

Victoria 3010, Australia

[www.cs.mu.oz.au/~{vo,alistair}](http://www.cs.mu.oz.au/~{vo,alistair})

**Abstract** A two-phase evaluation scheme is proposed for XML retrieval. In the first phase, a modified vector space model is employed to obtain similarity scores for the textual nodes of XML trees. In the second stage, the scores are propagated upward in the XML trees, with scores of the textual nodes being modified and scores of other nodes being generated. As a result, while a vector space ranking is used, the final scores computed do not truly reflect the vector space scores. In addition to the two-phase evaluation, an integrated compressed file system is proposed for both storing and retrieving XML documents. This leads to an efficient representation of XML repositories.

## 1 Introduction

Applying IR techniques to XML retrieval is undoubtedly an interesting and promising approach. Conventional IR techniques, however, cannot be employed directly because of the need to handle content-and-structure queries. To accept this kind of query, retrieval systems must capture the structure of the documents and queries, and carry out some computation over these structures. In this paper we focus on two of the various aspects of the task. The first focus is on an alternative method to extend the vector space model to XML retrieval. The second is a unified compression scheme that supports both the retrieval model and efficient decompression of any part of an XML document. While the first goal is core to the *INEX* project, the second goal should as well be regarded as important. XML document collections can be large. Moreover, retrieval of XML elements involves not only the document content but also its structure, potentially consuming more disk space than retrieval of flat documents would.

A number of techniques to extend the vector space model to XML retrieval have been presented. Three main approaches are worth commenting on. Fuhr et al. [1998], Fuhr and Großjohann [2001] explicitly indicate *indexing nodes*, each of which is a group of XML nodes. Indexing is then done for these nodes. This static index is used directly for query processing. Grabs and Schek [2002] proposed to generate vector space statistics on-the-fly during query processing. In this approach, a static index is built only for *basic indexing nodes*, which can be defined manually or automatically. At query time, the basic index is used to derive appropriated vector space statistics depending on the query scope. Carmel et al. [2002] chose to index the pairs (*path*, *word*) (as opposed to

the conventional indexing of *words* only), where *path* is the XML path of the node that contains *word*.

A common property of these techniques is that they are tightly bound to the vector space model. During the evaluation of a query, the statistics are retrieved or generated for all nodes that are in the query's scope. These statistics are then used to compute similarity scores and rank the nodes. The common property likely guarantees the correctness of applying a vector space ranking, since otherwise there would be serious problems with ranking inconsistency. On the other hand, semi-structured XML documents are quite different from flat documents for which vector space ranking is good, and an alternative formulation of the similarity score might be preferable. Moreover, it is still not clear how to fairly combine different kinds of XML node according to a common statistical scale.

We use a vector space ranking technique because of its efficiency and effectiveness for flat text retrieval. But we do not rely exactly on the vector space score. Instead we adjust the scores, possibly more than once. The "right" statistics for an appearance of a word are counted once for the node that contains the word directly. Only these nodes are then processed through the conventional vector space ranking, regardless of whether they are compatible with the structural conditions of the query. Even at this stage, the scores computed are not exactly vector space scores – they are augmented according to the structural conditions. After the IR stage has been done, a second stage is conducted where the scores are propagated upward in the XML tree, and then the top nodes are selected as answers.

For our second goal – providing a compression framework for XML retrieval, we mainly rely on the existing work. Our contribution here is extending the current compression framework for flat text retrieval to XML retrieval. We introduce additional files to keep the XML collection in the compressed form, allowing effective decompression of any XML node.

The remainder of the paper is organized as follows. Section 2 introduces some concepts of XML documents and presents our opinion on query format and interpretation of queries. Then, section 3 describes the data structures employed for compressing XML collections. Section 3 also introduces a general scheme for query evaluation with these structures. Sections 4 and 5 describe the main techniques employed for the two phases of evaluation. Section 6 shows the experiments and their results. Section 7 is intended to draw some conclusions and directions for future work. The last two sections will be completed after the workshop.

## 2 Documents, queries and query interpretation

**Documents** A simplified example of an XML document is provided in figure 1 and is used throughout this text to illustrate the concepts introduced.

It is convenient to list some of the standard definitions here. Thus, an XML document is a set of *nodes* or *elements* such as

---

```

<article>
  <atl> XML Retrieval </atl>
  <au sequence="first">
    <fnm> First N. </fnm>
    <ref> Surname </ref>
  </au>
  <sec> <st> Everything </st>
  <p>
    Everything about <it> XML </it>
    and <it> XML retrieval </it>.
  </p>
</sec>
</article>

```

---

Figure 1: Example of XML document.

`<article>` and `<p>`. Each node is associated with a *path*, for example, `/article` and `/article/sec/p`. The exact location and content of a node is defined by its *positional path*. For example, if the above XML document is the first one in a collection, then `/article[1]/sec[1]/p[1]/it[1]` and `/article[1]/sec[1]/p[1]/it[2]`, respectively, is used to indicate `<it> XML </it>` and `<it> XML retrieval </it>`.

The following concepts are introduced for this paper. A node is called *textual* if and only if it has some proper free text which does not belong to any of its children or descendants. Otherwise, the node is called *skeleton* and it contains no proper text. In the above document, for example, textual nodes are

```

/article[1]/atl[1],
/article[1]/au[1]/fnm[1],
/article[1]/au[1]/ref[1],
/article[1]/sec[1]/st[1],
/article[1]/sec[1]/p[1],
/article[1]/sec[1]/p[1]/it[1],
/article[1]/sec[1]/p[1]/it[2];

```

and the skeletal nodes are

```

/article[1],
/article[1]/au[1],
/article[1]/sec[1].

```

Note that normally in an XML tree, leaf nodes are textual, and internal nodes are skeletal, but this cannot be assumed. A counterexample is the `/article[1]/sec[1]/p[1]`, which is an internal node, but containing some proper text. This type of node is popular in the *INEX* collection.

The textual part of a textual node, including any accompanying punctuation, is called a *textual item* of the node *wrt* the XML collection. Thus, the textual item of `/article[1]/au[1]/ref[1]` is *Surname*, while that of `/article[1]/sec[1]/p[1]` is *Everything about and*.

**Queries** We appreciated the straightforward query format supplied by the *INEX* organizer and described by Fuhr et al. [2002]. In our opinion, the format (of course, after removing *Description* and *Narrative* fields) is simple and powerful enough, at least for the purpose of IR approaches.

To make the queries more consistent, we introduced a couple of small changes to the initial format. Firstly, words appearing in a `ce` field are included inside the field itself. Secondly, a formal element `<nw> ... </nw>` is added to surround negative words in queries. For example, topic 09 is now reformatted as shown in figure 2.

We believe that the *Keywords* of the original *INEX* queries is unnecessary and it would better be removed totally from the query

---

```

<query>
  <te> article </te>
  <ce>
    <cp> bdy/sec </cp>
    <cw> nonmonotonic reasoning </cw>
  </ce>
  <ce>
    <cp> hdr/yr </cp>
    <cw> 1999 2000 </cw>
  </ce>
  <ce>
    <cp> tig/atl </cp>
    <cw> <nw> calendar </nw> </cw>
  </ce>
  <cw> belief revision </cw>
  <kw>
    nonmonotonic reasoning belief revision
  </kw>
</query>

```

---

Figure 2: Example of query: the reformatted version of topic 09.

format, making queries simpler and shorter. However, to be consistent with the settlement of this round of *INEX*, this element is left in this format with the new name of `<kw>`.

There is a number of points that should be made clear. Firstly, the *Title* field in this format is removed since we consider that field the main part of queries. As the field is in fact a structured node, it is simply removed. Secondly, the format is used for both content-only and content-and-structure queries, and we also recommend the use of queries which have no `te` field but contain `ce` fields. Thirdly, it is easy to build a script to transfer all *INEX* queries to the new format automatically. And last, except for the `te` field, all other information should be considered by a retrieval system as inexact constraints as is also the case in conventional IR ranking. For example, the first `ce` element in 2 should be interpreted as the desire of having the sections discussing *about* “nonmonotonic reasoning”, and it does not necessarily mean that the sections must contain these word. In the same manner, a retrieved article for the query, for example, might not be published in 1999-2000 as required by the topic’s author.

### 3 System Architecture

**Backbone** Our system is developed from the *MG* system (see <http://www.cs.mu.oz.au/mg/>). The main feature of *MG* for text retrieval is that it uses compression for document collections as well as for their indexes. This feature is especially suitable for our task of building a compact repository for XML retrieval. We report here only the changes made specifically for this task.

**File system** *Textual and related files:* All textual items of the XML documents are gathered together in a data structure, called *textual file*. That is, each item in this file corresponds to one textual node of a certain XML document. This file is compressed and is accompanied with some auxiliary files supporting direct access to, and decompression of, each of of the textual items. An illustration of textual files is given at the bottom left of figure 3. Information about text compression methods employed, as well as about the auxiliary data structures, can be found in [Witten et al., 1999].

*Structural files:* Each node (either textual or structural) of any XML document has an entry in a structural file. In this data structure, entries are stored in the order of their appearance (or, more correctly, of the appearance of their opening markups) in the XML

collection. An entry of the structural file describes a node's structure and its position in the parent's node. The entry includes

- the opening markup of the node (including the accompanying parameters, if any);
- distance to the parent node (ie. number of nodes being between the node and its parent, which is 0 if the current node is a root node);
- byte-offset position of the beginning of the node relative to the (end of) its immediately preceding markup;
- pointer to the textual item of the node, ie. to the corresponding item in the textual file (the value is 0 if the node is a skeleton).

The bottom right block in figure 3 illustrates the content of a structural file. Note that for each node, the closing markup is not stored.

To the structural items, random access is needed. Since all the numerical values of the file is generally small, and the texts (ie. the markups) are generally repeated, the file can be compressed effectively even with the random access requirement. Our ad-hoc solution is to use a dictionary for all the text parts, then to replace each text with the pointer to the corresponding element in the dictionary, hence transforming each structural item to a quadruple of numbers prior to the compression. Conventional compression techniques for small integers are then applied.

Note that with support of the textual files, which allow direct access and decompression of any textual item, the structural file can be used to build back any node of the original XML document collection. An example of this process is given in figure 3. Truly, the compression is lossy: when there is no text between two consecutive markups, the punctuation between them (if any) is not stored anywhere. However, as the primary purpose of the XML documents is to have the structure of documents along with their texts, not to render them, the compression scheme can be considered as lossless.

*Text-structure mapping files:* A text-structure mapping file is illustrated at the top of figure 3. The file maps any item in textual file to its corresponding entry in the structural file. During query processing it is better to have the mapping resided in the memory, so the random access to the file is not required. Hence, the numbers indicating absolute position of a structural node (in the structural file) are replaced by the gaps between it and its preceding . That is, run-length coding is applied. In our current implementation, Elias's Gamma code Elias [1975] is used for this purpose.

*Index files:* Changes have been made to *MG* to suit our needs, in both the indexing and the querying modules. While the changes are already reported in Anh and Moffat [2002], it is worth reiterating that the weighting scheme for terms of the textual items is integrated into the index, and that during query processing, the calculation of cosine measure for these items is not required.

*Remark:* It might be arguable about the need to divide the XML collection into textual, structural and the mapping files since keeping them in one file might be better for compression. The point is that during query evaluation the structural parts are needed anyway, when the whole textual parts are needed only when the documents need to be rebuilt to present as the answer. Another argument might be that it would probably better to insert empty items to the textual file to represent the structural nodes, and hence exclude the mapping file from consideration. However, number of such empty items is relatively high, making the compression of inverted files ineffective.

**Query evaluation** After an query has been parsed, information about each of its distinct terms is stored in a general list data structure. This information includes representation of the term itself,

list of the query's  $\langle ce \rangle$  paths that contains the term (with a special value to represent "any path"), and the frequency for each of these paths. The evaluation then involves the following main steps:

1. *Scoring:* Conventional vector space technique, with an adjustment to take into account structural conditions of queries, is employed to calculate similarity score for each textual item. The weighting scheme for this step is reported in section 4.
2. *Propagation:* The scores obtained are propagated upward in the XML tree, hence awarding the internal (not necessarily being structural) nodes with some scores. The techniques for doing this step is shown in section 5.
3. *Selection:* After the previous step we come up with a list of nodes with non-zero scores. The task of the selection step is a) to delete some anomalies, and b) to select the nodes with the top scores. There are three situations where a node is considered abnormal. The first case is when the node or any of its descendants has negative score. The second case happens when the parent of the node scores higher than it as well as any of its siblings. The motivation behind this case is to avoid retrieving the descendants of retrieved nodes. The third case is applied only to content-and-structure queries. It involves the clearing of scores of the nodes that do not belong to the  $\langle te \rangle$  list.
4. *Presentation:* The list of the nodes with the top scores is now used to retrieve the actual nodes. In this step, we use information from the structural file to rebuild the full node. Figure 3 serves as an example for this process.

For simplicity, the first and second steps, and only them, are referred to as the first and second, respectively, phase of the query evaluation process.

## 4 Weighting Textual Items

The weighting scheme employed for the textual items is based on our impact transformation technique [Anh and Moffat, 2002]. The weight is an integer number and computed as

$$S_{d,q} = \sum_{t \in q \cap d} p_{d,q,t} \cdot \omega_{d,t} \cdot \omega_{q,t}$$

where  $p_{d,q,t}$  is *cross-structural importance* of  $t$  relative to  $d$  and  $q$ ,  $\omega_{d,t}$  and  $\omega_{q,t}$  are *quantized impact* of term  $t$  in textual item  $d$  and query  $q$ , respectively.

The cross-structural importance is defined by

$$p_{d,q,t} = c_w \cdot p_{d,q,t}^w + c_{\bar{w}} \cdot p_{d,q,t}^{\bar{w}} + c_e \cdot p_{d,q,t}^e + c_{\bar{e}} \cdot p_{d,q,t}^{\bar{e}}$$

Here  $c_w$ ,  $c_{\bar{w}}$ ,  $c_e$  and  $c_{\bar{e}}$  are constants and, in this series of experiments, are set to 1, 10, -10 and -20, respectively. Other values are generally 0 except for the following special cases:

- $p_{d,q,t}^w$  is set to 1 if  $t$  appears in either  $/query/cw$  or  $/query/kw$ , and  $d$  is any textual item,
- $p_{d,q,t}^{\bar{w}}$  = 1 if  $t$  appears in  $/query/cw/nw$ , and  $d$  is any textual item,
- $p_{d,q,t}^e$  = 1 if  $t$  appears in an  $/query/ce/cw$  field and the parent of this field contains at least one item that is the same as, or the ancestor of, the path name of the textual element  $d$ ,
- $p_{d,q,t}^{\bar{e}}$  = 1 if  $t$  appears in an  $/query/ce/cw/nw$  field, and the ancestor  $/query/ce$  of this field contains at least one item that is the same as, or the ancestor of, the path name of the textual element  $d$ .

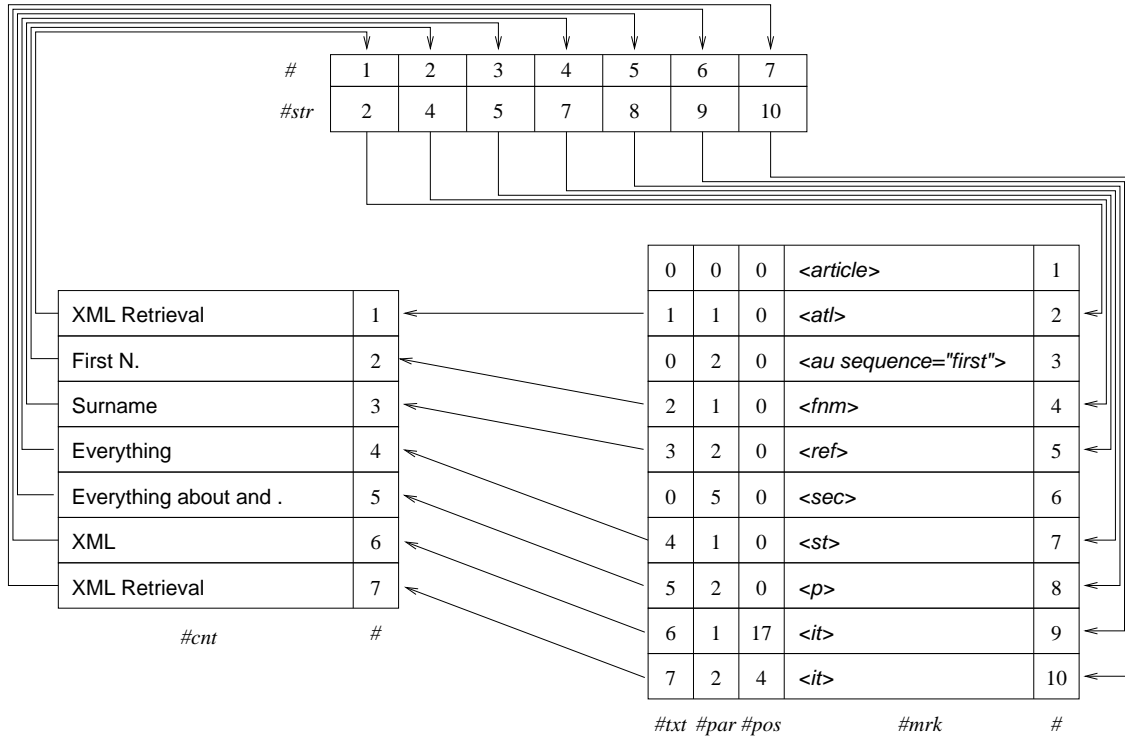


Figure 3: Example of textual, structural and their mapping files. The picture conceptually describes a database which has only one XML document, namely, the document presented in figure 1. The arrows represent explicit or implicit links from a file to another. To each file, the field # is added to show the item number. The contextual file is shown at the bottom left. It contains 7 items, each for one textual node of the document. The ranking is first done in the IR manner for these items. In this process, the system can use the mapping file (top) to map the items with their path in the structural file (bottom right). In the structural file, the #par link a node to its parent. For example, the value 2 of #par for the last item (item number 10) means that its parent is at 2 positions ahead, ie. is the item number 10-2=8. The columns #txt and #pos are used to rebuild nodes. For example, rebuilding of item number 8 begins from building its initial string value <p> Everything about and . <p>, then the next structural items are taken to insert into this string since they are the item's children. Value #par = 17 of item 9 shows that the corresponding node should be inserted to position 17 after <p> (its preceding markup), making the above string become <p> Everything about <it> XML </it> and . <p>. Similarly, item number 10 should be inserted to this string at position 4 after </it>.

Each of the quantized impacts  $\omega_{d,t}$  and  $\omega_{q,t}$  is in the range 1 to  $2^b$ , with (in these experiments)  $b = 5$ . Each of them is calculated in two steps. First, a normal cosine similarity is used to compute  $w_{d,t}^*$  and  $w_{q,t}^*$ :

$$w_{d,t} = (1 + \log_e f_{d,t})$$

$$W_d = \sqrt{\sum_{t \in d} w_{d,t}^2}$$

$$W_d^* = 1 / ((1 - s) + s \cdot W_d / W^\alpha)$$

$$w_{d,t}^* = w_{d,t} / W_d^*$$

$$w_{q,t}^* = \log_e \left( 1 + \frac{f_t^m}{f_t} \right) \cdot (1 + \log_e f_{q,t})$$

where  $f_{d,t}$  is the term frequency in the textual item,  $f_{q,t}$  is frequency of  $t$  in the textual part of the query  $q$  (that is,  $f_{q,t}$  is calculated without considering the markups);  $f_t$  is the number of textual items that contain term  $t$ ;  $f_t^m$  is the greatest value of  $f_t$  in the textual file;  $W_d$  is length of the textual item  $d$ ;  $W^\alpha$  is the average value of  $W_d$  over all items of the textual file; and  $W_d^*$  represents the normalized item length using pivoted normalization [Singhal et al., 1996] with a slope of  $s = 0.7$ .

Then, a small enough positive value  $L$  and a large enough positive value  $U$  are chosen such that all of the  $w_{d,t}^*$  lie between  $L$  and  $U$ , thereby allowing the following transformation to be calculated:

$$\omega_{d,t} = \left\lceil 2^b \cdot \frac{\log w_{d,t}^* - \log U}{\log U - \log L + \epsilon} \right\rceil + 1$$

$$\omega_{q,t} = \left\lceil 2^b \cdot \frac{\log w_{q,t}^* - \log U}{\log U - \log L + \epsilon} \right\rceil + 1$$

in which  $B = (U/L)^{L/(U-L)}$ , and  $\epsilon$  is a small positive quantity, and the impact values are recorded and used as integers.

Our experiments made use of two different types of transformation, characterized by the choice of  $L$  and  $U$ . In the first, referred to as *global*, the values of  $L$  and  $U$  respectively are the minimum and maximum values of  $w_{d,t}^*$  over the whole textual file. In the second, referred to as *local*, each textual item or query  $x$  is associated with its own  $L$  and  $U$ , which are the minimum and maximum among all of the values  $w_{x,t}^*$  generated from  $x$ . That is, in the local transformation, a value  $w_{x,t}^*$  is transformed with respect to the values of  $L$  and  $U$  of  $x$  – the textual item or query it belongs to.

## 5 Propagating Scores

After having the scores of the textual nodes, the next step is to propagate the scores upward in the XML trees (or tree). Two methods are investigated in our experiments. In the description of the methods (below) it is supposed that the propagation is being done for a node  $b$  whose parent is  $a$ , and that  $a$  has totally  $n$  children, of them  $m$  have non-zero (possibly negative) score.

The first method is called *maximum-by-category*. Here, each distinct term is called a *category*. For this method, whenever a score is computed, regardless of whether the computation belongs to the first or the second phase of the evaluation process, it is calculated separately and kept separately for each category. A real score of an item is then the sum of its scores over the categories. Hence the categorical score of  $b$  can be represented as  $(s_1(b), s_2(b), \dots, s_{|q|}(b))$ , and the real score for  $b$  is

$$s(b) = \sum_{i=1}^{|q|} s_i(b)$$

where  $|q|$  is number of distinct terms of query  $q$ . The score  $s(a)$  of  $a$  is computed based on

$$s_i(a) = s_i(b) + \text{sign } s_i(b) \cdot \alpha \cdot \max_b |s_i(b)|,$$

where  $|s_i(b)|$  is the absolute value of  $s_i(b)$ ,  $\alpha$  is a constant and is set to 0.8 in these experiments.

The second method of propagation is called *summation*. It involves not only the calculation of  $s(a)$  but also the re-scoring of  $s(b)$ .  $s(a)$  is computed as

$$s(a) = s(a) + \sum_b (\beta \cdot s(b)/n + \gamma \cdot s(b)/m)$$

and  $s(b)$  is redefined as

$$s(b) = s(b) - (\beta \cdot s(b)/n + \gamma \cdot s(b)/m).$$

where  $\beta$  and  $\gamma$  are constants. Both of them are set to 0.5 in the experiments reported below.

## 6 Experiments and performance

**Hardware** The experiments were carried out on a 933 MHz Intel Pentium III with 1 GB RAM, a 9 GB SCSI disk for system needs, and four 36 GB SCSI disks in a RAID-5 configuration for data. The indicative times reported below are for experiments in which there was no other activity on the hardware.

Label	Characteristics
um_mgx21_short	<i>Queries:</i> not having $\langle kw \rangle$ elements <i>Type of transformation:</i> global <i>Propagation method:</i> summation
um_mgx2_long	<i>Queries:</i> having $\langle kw \rangle$ elements <i>Type of transformation:</i> global <i>Propagation method:</i> maximum-by-category
um_mgx26_long	<i>Queries:</i> having $\langle kw \rangle$ elements <i>Type of transformation:</i> local <i>Propagation method:</i> maximum-by-category

Table 1: Settlement of the experiments

**Experiment parameters** Three experiments have been conducted. Their label and setting is listed in table 6.

## Outcomes

## 7 Conclusion

## References

- V. N. Anh and A. Moffat. Impact transformation: effective and efficient web retrieval. In M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–10, Tampere, Finland, Aug. 2002. ACM Press, New York.
- D. Carmel, N. Efraty, G. M. Landau, Y. S. Maarek, and Y. Mass. An extension of the vector space model for querying xml documents via xml fragments. In *Proc. SIGIR'2002 Workshop on XML and Information Retrieval*, pages 14–25, Tampere, Finland, Aug 2002.
- W. Croft, D. Harper, D. Kraft, and J. Zobel, editors. *Proc. 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, Louisiana, USA, Sept. 2001. ACM Press, New York.
- P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, Mar. 1975.
- N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas. INEX: Initiative for the Evaluation of XML Retrieval. In *Proc. SIGIR'2002 Workshop on XML and Information Retrieval*, pages 62–70, Tampere, Finland, Aug 2002.
- N. Fuhr, N. Gövert, and T. Rölleke. Dolores: A system for logic-based retrieval of multimedia objects. In W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proc. 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 257–265, Melbourne, Australia, Aug. 1998. ACM Press, New York.
- N. Fuhr and K. Großjohann. XIRQL: a query language for information retrieval in xml. In Croft et al. [2001], pages 172–180.
- T. Grabs and H. Schek. Generating vector spaces on-the-fly for flexible xml retrieval. In *Proc. SIGIR'2002 Workshop on XML and Information Retrieval*, pages 4–13, Tampere, Finland, Aug 2002.
- A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, editors, *Proc. 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–29, Aug. 1996.
- I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, second edition, 1999.

# Appendix

Guidelines for topic development	106
Guidelines for retrieval result submission	110
Guidelines for relevance assessments	113
Assessments and evaluation measures	117

# INEX Guidelines for topic development

May 2002

The aim of the INEX initiative is to provide means, in the form of a test collection and appropriate scoring methods, for the evaluation of 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 a large test collection for the evaluation of XML retrieval. 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, or queries, and a set of relevance assessments that lists 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.

With the use of XML query languages, users of an XML retrieval system are able to restrict their search to specific structural elements within an XML collection. A test collection for XML retrieval should therefore include two types of query:

- content-and-structure, and
- content-only.

Content-and-structure queries are topic statements, which contain references to the XML structure, either by restricting the context of interest or the context of search terms. Content-only queries ignore the document structure and are the traditional topics used in IR test collections. The need for this type of query for the evaluation of XML is well published and stems from the fact that users may not know the XML structure, or may not want to restrict their search to specific target elements. Examples of both types of query are given in Section 2.2.

Finally the relevance assessments for an XML collection must also consider the structural nature of the documents. Currently, there are several issues as to the exact particulars of the relevance assessment procedures. Participating organisations will be given the opportunity to contribute their opinions and ideas on this matter prior to the release of the relevance assessment guidelines.

The next section provides detailed guidelines for the creation of topics for the XML test collection.

## 2. Topic creation

### 2.1. 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 relevant documents or only relevant documents.

When creating topics, a number of factors should be taken into account.

- **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!)**
- Topics should reflect what real users of operational systems might ask.
- Topics should be diverse.
- Topics should be representative of the type of service that operational systems might provide.
- Topics may also differ in their coverage, e.g. broad or narrow topic queries.



## 2.2. Topic format

A topic is made up of four parts: title, description, narrative and keywords. Title is a short, 2-3 word version of the topic statement, made up of words that best describe what the user is looking for. In the case of content-and-structure queries, it also specifies the target element(s) - <te> - of the search and the context(s) - <cx> - of the search word(s) - <cw>. A topic description is a one-sentence definition of an information need. The narrative is the explanation of the topic statement in more detail and the description of what makes a document relevant or not. Keywords are good scan words that are used in the collection exploration phase of the topic development process (see Section 2.3.2.). Scan word may include synonyms or broader, narrower terms from that listed in the topic description or title. Below is an example of a content-only and a content-and-structure topic. Note that there are no <te> and <cx> elements for the content-only query, meaning that there is no restriction on what element should be returned by the engine and the content words may also occur in any arbitrary element.

```
<topic>
  <title>
    <cw>Combating alien smuggling</cw>
  </title>
  <description>
    What steps are being taken by governmental or even private
    entities world-wide to combat the smuggling of aliens.
  </description>
  <narrative>
    To be relevant, a document must describe an effort being made
    (including border patrols) in any country of the world to prevent
    the illegal penetration of aliens across borders.
  </narrative>
  <keywords>
    smuggling illegal trafficking alien customs border country world
    prevent combat stop government
  </keywords>
</topic>

<topic>
  <title>
    <te>chapter, article_title</te>
    <cw>nuclear energy</cw><ce>article_title</ce>
    <cw>technical report</cw><ce>article_type</ce>
    <cw>safety nuclear power plant</cw>
  </title>
  <description>
    Retrieve the title and relevant chapters of technical reports
    about the safety procedures and safety issues of nuclear power
    plants where the title of the report contains reference to nuclear
    energy.
  </description>
  <narrative>
    Relevant documents would be preferably, but not exclusively,
    chapters of technical reports which discuss the day-to-day
    operational safety guidelines and procedures of nuclear power
    stations world wide. References to safety issues and possible
    shortfalls of the safety procedures are also of interest. Reports
    about nuclear disasters or incidents may also be relevant provided
    they hint at the cause of the problem.
  </narrative>
  <keywords>
    nuclear energy power plant station safety regulations upkeep
    servicing checks incident accident leak radiation health hazard
  </keywords>
</topic>
```

The example of a content-and-structure topic shows that the target elements (that is, what the user wants to retrieve) are chapters and article titles. Furthermore, it specifies that the context element (or container element) of the search words “nuclear” and “energy” should be the article\_title element, and

that the element `article_type` should contain the words “technical” and “report”. The search words “safety”, “nuclear”, “power” and “plant” may occur anywhere. Note that both the target element and the context element may be given as paths (e.g. `article/header/article_title`) or as element types (e.g. `article_title`). Content-and-structure queries may specify both target and context elements, or either target or context only elements.

The structure of the topics is given in the DTD below.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT topic (title, description, narrative, keywords)>
<!ELEMENT title (te?, (cw, ce?)+)>
<!ELEMENT te (#PCDATA)>
<!ELEMENT cw (#PCDATA)>
<!ELEMENT ce (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT narrative (#PCDATA)>
<!ELEMENT keywords (#PCDATA)>
```

## 2.3. Procedure for topic development

**Each participating group will have to submit 3 content-only and 3 content-and-structure queries by the 10<sup>th</sup> of June by filling in the form (one per topic) at**

**<http://qmir.dcs.qmw.ac.uk/inex/TopicSubmission.html>**.

This section outlines the procedures involved in the development of candidate topics. There are four steps in the process of creating topics for a test collection: creating initial topic statements, exploring the collection, selecting final set of topics, and refining the topic statements.

### 2.3.1. Initial topic statements

In this step, you create a one-sentence description of the information you are seeking. This should be a simple description of the needed information without regard to retrieval system capabilities or document collection peculiarities. This will become the topic description field.

### 2.3.2. 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/document components 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.

Use the Candidate Topic Form to record information during your exploration (this form will be used to submit your candidate topics). For each query record the initial query statement (the result of task 2.3.1), the set of keywords that you use for retrieval. You should try and make this query 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/document components of your retrieval result and record the number of relevant components and their element types. If you have found at least 1 relevant component and no more than 20, perform a feedback search and 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/document components in the table. Finally record your thoughts on what makes a document/document component relevant.

To assess the relevance of a retrieved document or document component use the following working definition: mark a document/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/document component should be judged on its own merits. That is, a document/document component is still relevant even if it is the thirtieth document/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.

### 2.3.3. Refining topic statements

Refining the topic statement means finalising the topic title, description, keywords and narrative. Note that each of the four parts of a topic (title, description, narrative and keywords) should be able to be used in a stand-alone fashion (e.g. title for retrieval using short queries, narrative for filtering etc.). The expectation is that by judging 100 documents/document components you will have determined how you will judge the topic in the assessment phase. The narrative of the topic should reflect this. Note that there will be a three-month gap before you will do the relevance assessments, so it is vital that you record as much as you can in order to maintain judgement consistency.

### 2.3.4. Topic selection

The data obtained from the collection exploration phase will be used as input to the topic selection process. Make sure you submit all **6** candidate topics by filling in the form at <http://qmir.dcs.qmw.ac.uk/inex/TopicSubmission.html> no later than the 10<sup>th</sup> of June. We (the clearinghouse) will then decide which topics to use such that a wide range of likely number of relevant documents is included, and will distribute these back to you as the final set of topics to be used for the retrieval and evaluation.

# INEX retrieval result submission format

An INEX submission is a record of the search results you obtained with respect to the INEX topics. For the relevance assessment and evaluation of your results we require your submissions to be in the format described in this document.

The overall submission format is defined by the following DTD:

```
<!ELEMENT inex-submission      (topic+)>
<!ATTLIST inex-submission
  participant-id  CDATA      #REQUIRED
  run-id         CDATA      #REQUIRED
  run-descr      CDATA
>

<!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)>
```

A submission should contain the top 100 retrieval results for each of the INEX topics. A submission must contain the participant ID of the submitting institute (available at <http://qmir.dcs.qmw.ac.uk/inex/Participants.html>) and a run ID. You may submit up to 3 retrieval runs (one per submission file), each identified by a unique run ID. You may also include a short description of your retrieval run in the run-descr attribute. A submission consists of a number of topics, each identified by a topic ID (which will be provided in the topic descriptions). A topic consists of a number of result elements, the retrieval results of your search on that topic, described by a file and a path. A result description can have a rank and/or a retrieval status value (rsv). Before we describe the various elements and attributes of the above DTD, this is how an example submission could look like:

```
<inex-submission participant-id="12" run-id="MyApproach">

  <topic topic-id="01">

    <result>
      <file>tc/2001/t0111</file>
      <path>/article[0]/bm[0]/ack[0]</path>
      <rsv>0.67</rsv>
    </result>

    <result>
      <file>an/1995/a1004</file>
      <path>/article[0]/bdy[0]/sec[0]/p[2]</path>
      <rsv>0.1</rsv>
    </result>

    [ ... ]

  </topic>

  <topic topic-id="02">

    [ ... ]
```

```
</topic>

[ ... ]

</inex-submission>
```

## Ranks and RSV

Ranking of results can be either described in terms of rank values (consecutive natural numbers, starting with 1; there can be more than one element per rank) or retrieval status values (RSVs, real numbers; result elements might have the same RSV). Choose either one to describe the ranking within your submissions. If both, rank and rsv are given we will consider the rank for evaluation. If your retrieval approach does not produce ranked output, omit these elements in your submission.

## File and path

Since XML retrieval approaches may return arbitrary XML nodes from the documents in 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 plus a path specification in XPath syntax.

File names are relative to the INEX collections xml directory. They use '/' for separating directories. Article files as well as the volume.xml files can be referenced here. The extension .xml must be left out. Examples:

```
an/1995/a1004
an/1995/volume
```

Paths are given in (extended) XPath syntax. To be more precise, only fully specified paths are allowed, as described by the following grammar:

```
Path          ::= '/' ElementNode Path | '/' ElementNode '/' PathEnd

PathEnd       ::= ElementNode | AttributeNode

ElementNode   ::= ELeMentName Index

AttributeNode ::= '@' AttributeName

Index         ::= '[' integer ']'
```

An example path

```
/article[0]/bdy[0]/sec[0]/p[2]
```

would describe the element which can be found if we start at the document root, select the first “article” element, then within that element, select the first “bdy” element, within that element select the first “sec” element, within that element select the third “p” element. As can be seen, XPath counts elements starting with zero.

As said before, elements are unambiguously identified by a (file name, path) pair. On the other hand, there are two ways to specify an element within the INEX collection. The first way is via the article file, the second one is via the respective volume.xml file. In the example below the two specifications refer to the same element:

```
<result>
  <file>an/1995/a1004</file>
  <path>/article[0]/bdy[0]/sec[0]/p[2]</path>
```

```
</result>
```

```
<result>
```

```
  <file>an/1995/volume</file>
```

```
  <path>/books[0]/journal[0]/article[1]/bdy[0]/sec[0]/p[2]</path>
```

```
</result>
```

Both of these methods are valid and will be accepted as correct submissions.

An application, which helps you to check the correctness of your path specification will be available at <http://ls6-www.cs.uni-dortmund.de/ir/projects/inex/download/#explore>.



# INEX Relevance Assessment Guide

## 1. Introduction

During the retrieval runs, participating organisations evaluated the 60 INEX queries against the IEEE Computer Society document collection and produced a list (or set) of document components (XML elements) as the retrieval result for each query. The top (or first) 100 components in a query's retrieval result 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. However, 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 whenever possible).

The aim of this guide is to outline the process of providing assessments for the INEX test collection. This requires first a definition of the metrics against which document components will be assessed (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 and Coverage

For an XML test collection it is necessary to obtain assessments for the following two dimensions.

- *Topical relevance, which describes the extent to which the information contained in a document component is relevant to the topic of the request.*
- *Document coverage, which describes how much of the document component is relevant to the topic of request.*

To assess the topical relevance dimension, we adopt the following 4-point relevance degree scale.

- 0: Irrelevant**, the document component does not contain any information about the topic of the request.
- 1: Marginally relevant**, the document component mentions the topic of the request, but only in passing.
- 2: Fairly relevant**, the document component contains more information than the topic description, but this information is not exhaustive. In the case of multi-faceted topics, only some of the sub-themes or viewpoints are discussed.
- 3: Highly relevant**, the document component discusses the topic of the request exhaustively. In the case of multi-faceted topics, all or most sub-themes or viewpoints are discussed.

To assess the document coverage, we define the following 4 categories.

- N: No coverage**, the topic or an aspect of the topic is not a theme of the document component.
- L: Too large**, the topic or an aspect of the topic is only a minor theme of the document component.
- S: Too small**, the topic or an aspect of the topic is the main or only theme of the document component, but the component is too small to act as a meaningful unit of information when retrieved by itself (e.g. without any context).
- E: Exact coverage**, the topic or an aspect of the topic is the main or only theme of the document component, and the component acts as a meaningful unit of information when retrieved by itself.

Note that the two dimensions are orthogonal to each other. Relevance measures the exhaustiveness aspect of a topic, whereas coverage measures the specificity of a document component with regards to the topic. This means that a document component can be assessed as having exact coverage even if it only mentions the topic of the request (marginally relevant) or discusses only some of the topic's sub-

themes (fairly relevant) as long as the relevant information is the main or only theme of the component. According to the above definitions, however, an irrelevant document component should have no coverage and vice versa.

### 3. What to judge

Depending on the topic, a pooled result set may contain between 1000 and 2000 document components of 300-1000 articles, where a component may be a title, paragraph, section, or article etc. The document components in each pooled result set have been sorted alphabetically according to the article's file name and the component's path. Furthermore, all references to retrieval scores or ranking have been removed. This is so that your judgement is not influenced by the order in which document components are presented to you.

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. tables, figures, paragraphs, sections, subsections, articles etc. Therefore, in order 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 relevance and coverage - for the found relevant component, for its ascendant elements until you find an irrelevant component or a component with coverage L (too large), and for its descendant elements until you find an irrelevant component or a component with coverage N or S (no coverage or too small). For example, given the XML structure in Figure 1, if you judged Sub-section A fairly relevant with exact coverage (2E), Section C highly relevant with exact coverage (3E), but Body D highly relevant and too large (3L), then it can be assumed that Article E and Journal F are also highly relevant and too large (3L). On the other hand, if Sub-sub-section 1 was irrelevant with no coverage (0N) or marginally relevant and too small (1S), then it can be assumed that its descendant elements, e.g. Paragraph 3 and Paragraph 4, are also irrelevant with no coverage (0N) or marginally relevant and too small (1S).

Note that by the definition of "relevance" the relevance level of a parent element is equal to or greater than the relevance level of its children elements. The only exception to this rule is when a topic has a target element specification. In this case all elements (including the ascendant and descendant elements of a target element) except the target element are irrelevant, as they do not satisfy the structural condition of the topic.

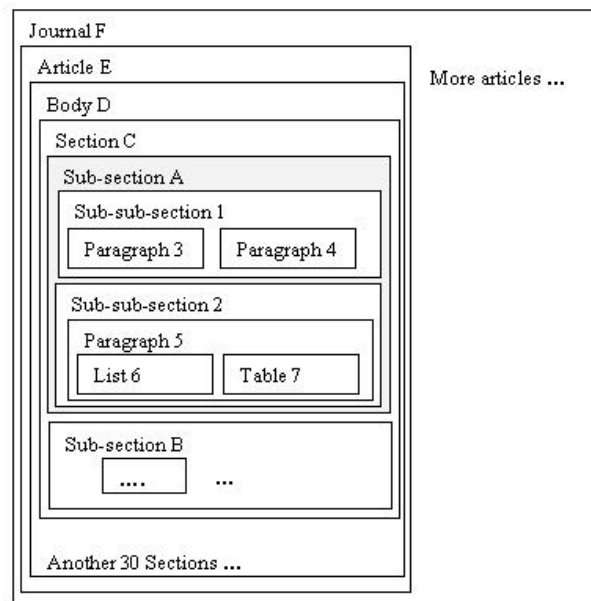


Figure 1. Example XML structure and result element

Furthermore, you will also need to judge the sibling elements of those relevant XML elements whose parent elements you judged more relevant than the element itself. For example, in the example above,



Section C was judged highly relevant, whereas Sub-section A was only marginally relevant. This means that Sub-section B must have contained some relevant information (either marginally or highly relevant), which must be explicitly specified during the assessment.

## **4. How to judge**

To assess the relevance and coverage of document components, we recommend a two-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 potentially relevant cue or search words within the article (see Section 5).
- In the second pass you should assess the relevance and coverage of the found relevant components, and of their ascendant and descendant XML elements. Remember you will only need to judge ascendant elements until you reach a component with too large coverage or an irrelevant component (when assessing a CAS topic with target element specification), and descendant elements until you reach an irrelevant component or a component with too small coverage (see Section 3).

During the relevance assessment of a given query, all parts, with the exception of the keywords, of the query specification should be consulted in the following order of priority: narrative, topic description and topic title. 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 the case there is conflicting information between the narrative and other parts of a topic, the information contained in the narrative is decisive. A document component, in general, should be judged relevant if it satisfies, to some degree (marginally, fairly, or highly, see Section 2), the query's information need as expressed within the narrative, the topic description and the topic title. 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, however, 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 even 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.

In the case of structure-and-content (CAS) queries, the topic titles contain structural constraints: pairs of concepts-context elements (cw, ce) and target element (te) specifications. These structural conditions should also be satisfied by relevant document components.

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 search 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://ls6-www.cs.uni-dortmund.de/ir/projects/inex/download/#assess>,



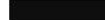
which allows you to view the pooled result set of a given query assigned to you for assessment, browse the IEEE-CS document collection and to record your assessments. Use your username and password to access this system.

After logging in, you will be presented with the topic ID numbers of the topics assigned to you for relevance assessment. Clicking on the topic ID will display the topic text. You should print this so that

you may refer to the topic description at any time during your assessment. A “pool” hyperlink is shown next to each topic ID. Click on this link to see the result elements in the query’s pooled result set.

Result elements in the pooled result set are shown in alphabetical order of the article's file name (that the result element is a part of) and the result element's path. At the top of this page you will see an “Edit your wordlist” button. This feature allows you to specify a list of words to be highlighted when viewing the contents of an article during assessment. The default list of words that appears in the wordlist is the words listed in the keywords section of the selected topic. You may edit, add to or delete from this default list of words. You may also specify the preferred highlighting colour for each and every word. Note that phrases have to be entered as individual words in separate lines.

When you finished setting up your wordlist, return to the pooled results page. On this page, the current assessment status of each article will be shown by one of the following three flags.

	article has no assessments at all,
	article has some assessments,
	article is finished.

To view the article that a result element is a part of you can choose from two available views: document and XML. Assessments must be done within the XML view, where the XML structure of the articles is shown explicitly. The document view is more readable for humans and might especially help you in the first pass of the assessment procedure (e.g. when skim reading the article to locate relevant information).

Within the article (in both views), the content of the result element will be highlighted in red and terms matching words in the wordlist will be highlighted in a shade of yellow (or your preferred colour). At the top of the page the path of the result element is printed (as a sequence of hyperlinks).

In the XML view, next to each XML start tag in the article you will see an input text box, where you should record the element's degree of relevance (0,1,2 or 3) and the category of coverage (N, L, S or E). Note that the order of the two dimensions is not strict and the coverage category is not case sensitive. Furthermore, there are two additional assessment input text boxes at the top of the page; one next to the “Journal” hyperlink referring to the journal that the article is a part of, and another next to the “Book” hyperlink referring to the book element that the journal is a part of. Assessments already provided for the XML elements in the article, journal and book will be displayed in any future assessment sessions.

As described in Section 4, first you will need to skim-read the text of the article (even if the result element itself is not relevant!) in order to identify any relevant information within the article. The highlighted words and the highlighted result elements are there to help you in finding possibly relevant information quickly. Mark any found relevant information by recording a degree of relevance and category of coverage to it in the appropriate assessment input text box. During your second pass you should return to the found pieces of relevant information and assess the relevance and coverage of their ascendant and descendant elements (until you find an irrelevant component or a component that is too large or too small, see Section 3).

At the bottom of the page (in XML view) you will see two buttons:

- “Submit assessment”: will save all assessments done so far and will set the assessment status of the article on the pooled results page to “article has some assessments”.
- “Finish article”: will save all assessments done so far and set the assessment status of the article to “article is finished”. Note that all non-assessed XML elements within the article will be automatically assigned either default or inferred relevance and coverage values, where the default is ON, and inferred is for ascendants: max(child relevance level) and min(child coverage level), for descendants: parent's relevance level and parent's coverage level, where consistency will be checked.

Note, to minimise the time it takes to keep displaying the pooled results page after returning from a document or XML view, you could keep the result pool in a separate browser window (or tab if your browser supports that) and reload this page time to time to update the flags.

# Assessments and a preliminary evaluation metric for XML document retrieval

Norbert Gövert  
University of Dortmund  
Germany  
goevert@ls6.cs.uni-dortmund.de

Gabriella Kazai  
Queen Mary University of London  
Great Britain  
gabs@dcs.qmul.ac.uk

## 1 Introduction

INEX<sup>1</sup> is the *Initiative for the Evaluation of XML Retrieval*. The aim of this initiative is to provide means, in the form of a large testbed (test collection) and appropriate scoring methods, for the evaluation of retrieval of XML documents. The test collection consists of XML documents, retrieval tasks / topics, and quality assessments.

Due to the nature of XML retrieval, there is the need to invent new evaluation procedures. Metrics from traditional evaluation initiatives like TREC and CLEF cannot be applied. Here we give an outline about what has been done within a preliminary evaluation of the INEX 2002 submissions.

In Section 2 the quality assessments done by the INEX 2002 participants are characterized. In Section 3 we describe how implicit assessments can be derived from the explicit assessments done by the assessors. The evaluation metric proposed here bases on standard recall/precision metrics. In order to apply them, the INEX quality assessments are to be mapped onto a binary relevance scale. The mapping function applied for this is given in Section 4. In Section ?? we describe the recall / precision metric applied. Section 6 gives few details about the implementation, while an overview on the resulting recall / precision curves is given in Section 7. Section 8 enumerates the drawbacks of the solution presented here.

## 2 Relevance and coverage assessments

The assessment guide<sup>2</sup> defines the two dimensions for which quality assessments were obtained:

**Topical relevance** , which describes the extent to which the information contained in a document component is relevant to the topic of the request.

**Document coverage** , which describes how much of the document component is relevant to the topic of request.

To assess the topical relevance dimension, we adopt the following four-point relevance degree scale:

- 0 Irrelevant, the document component does not contain any information about the topic of the request.
- 1 Marginally relevant, the document component mentions the topic of the request, but only in passing.
- 2 Fairly relevant, the document component contains more information than the topic description, but this information is not exhaustive. In the case of multi-faceted topics, only some of the sub-themes or viewpoints are discussed.
- 3 Highly relevant, the document component discusses the topic of the request exhaustively. In the case of multi-faceted topics, all or most sub-themes or viewpoints are discussed.

To assess the document coverage, the following four categories have been defined:

- N No coverage, the topic or an aspect of the topic is not a theme of the document component.
- L Too large, the topic or an aspect of the topic is only a minor theme of the document component.
- S Too small, the topic or an aspect of the topic is the main or only theme of the document component, but the component is too small to act as a meaningful unit of information when retrieved by itself (e.g. without any context).
- E Exact coverage, the topic or an aspect of the topic is the main or only theme of the document component, and the component acts as a meaningful unit of information when retrieved by itself.

## 3 Implicit assessments

Due to the nature of the two assessed dimensions *relevance* and *coverage* and from the INEX quality assessment guide one can deduce assessments for nodes which have not been assessed explicitly. In the following we use Datalog [Ullman 88] [Fuhr 00] for illustrating the rules for deducing such assessments. Deduction of implicit assessments is done based on the tree structure of the documents and the explicit assessments given. An example tree structure is depicted in

<sup>1</sup><http://qmir.dcs.qmul.ac.uk/INEX/>

<sup>2</sup><http://ls6-www.cs.uni-dortmund.de/ir/projects/inex/download/assessments.pdf>

Figure 3.

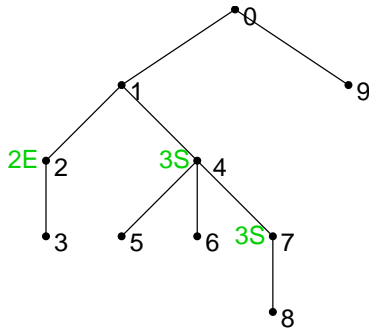


Figure 1: Example document with three explicit assessments for relevance and coverage.

The tree structure is described by means of a `parent(Parent, Child)` predicate by the following facts:

```
parent(n0, n1).
parent(n0, n9).
parent(n1, n2).
parent(n1, n4).
parent(n2, n3).
parent(n4, n5).
parent(n4, n6).
parent(n4, n7).
parent(n7, n8).
```

The explicit relevance assessments are represented by the two predicates `rel_assess(Node, Rel)` and `cov_assess(Node, Cov)` for the relevance and coverage dimension, respectively:

```
rel_assess(n2, 2).
cov_assess(n2, exact).
rel_assess(n4, 3).
cov_assess(n4, small).
rel_assess(n7, 3).
cov_assess(n7, small).
```

Now the two predicates `rel(Node, Rel)` and `cov(Node, Cov)` represent explicit and implicit assessments. For the two dimensions we now give a list of rules.

### 3.1 Relevance dimension

The most simple rule just takes over the explicit relevance assessments. In case there is no explicit relevance assessment for a given node, there are two possibilities to derive one. Either the assessment is derived from ancestors or from descendents in the document tree hierarchy. Propagation from the document's leaves towards the root is preferred here {Why do we do that? I don't know a real reason.}. If a relevance assessment could not be determined in one of these ways it is assigned relevance value of zero. The `rel(Node, Rel)` predicate thus can be described by the following four rules:

- (1) `rel(Node, Rel) :- rel_assess(Node, Rel).`
- (2) `rel(Node, Rel) :- ! rel_assess(Node, Rel1) & rel_up(Node, Rel).`
- (3) `rel(Node, Rel) :- ! rel_up(Node, Rel1) & rel_down(Node, Rel).`
- (4) `rel(Node, 0) :- ! rel_down(Node, Rel).`

Here, the explicit relevance assessments are propagated from the node assessed up to the root. The propagation is stopped if there is an explicit relevance assessment on the path towards the root node:

```
rel_up(Node, Rel) :- rel_assess(Node, Rel).
rel_up(Node, Rel) :- ! rel_assess(Node, Rel1)
    & parent(Node, Child)
    & rel_up(Child, Rel).
```

In the case that there is more than one child from which a given node could get an assessment, a maximum function on the relevance values can be used to decide which one to take over. {If assessors take the assessment guide serious, this should not happen...}

For nodes where no explicit assessment is found and no upward propagation could be done, relevance assessments are propagated down towards the document tree's leaves. However, there is one exception where the assessment is not copied verbatim. Nodes which have at least one sibling with a relevance assessment greater than zero, receive a relevance status of zero:

```
(* find out about the siblings of a given node *)
sibling(Node, Sibling) :- parent(Parent, Node)
    & parent(Parent, Sibling)
    & !=(Node, Sibling).
```

```
(* is there a sibling which has a
   relevance assessment already? *)
rel_sibling(Node) :- sibling(Node, Sibling)
    & rel_assess(Sibling, Rel).
```

```
rel_down(Node, Rel) :- rel_assess(Node, Rel).
rel_down(Node, 0) :- ! rel_assess(Node, Rel)
    & rel_sibling(Node).
rel_down(Node, Rel) :- ! rel_assess(Node, Rel1)
    & ! rel_sibling(Node)
    & parent(Parent, Node)
    & rel_down(Parent, Rel).
```

Figure ?? depicts the example tree with the implicit relevance assessments as computed by the rules described above.

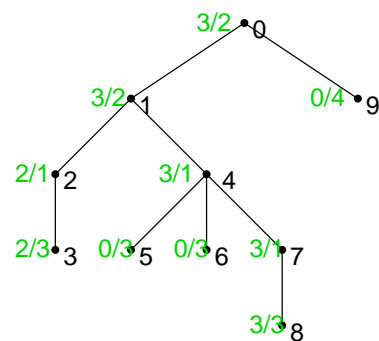


Figure 2: Example document with explicit and implicit assessments for relevance. The number of the rule by which the assessments are deduced is given after the respective slashes.

## 3.2 Coverage dimension

As for the relevance dimension there are three possibilities to arrive at an coverage assessment. Either an existing explicit assessment is taken or an assessment is propagated towards the document's root node or it is propagated towards the document leaves. Nodes which did not receive a relevance assessment by one of the above rules get a relevance status of zero.

- (1) `cov(Node, Cov) :- cov_assess(Node, Cov).`
- (2) `cov(Node, Cov) :- ! cov_assess(Node, Cov) & cov_up(Node, Cov).`
- (3) `cov(Node, Cov) :- ! cov_up(Node, Cov) & cov_down(Node, Cov).`
- (4) `cov(Node, no) :- ! cov_down(Node, Cov).`

In case no explicit coverage assessment is available, propagation towards the root is tried. Note that not for all coverage assessment of a given node a statement can be made about the coverage of the parent node. If the coverage of a node is assessed with *'too small'* nothing can be said about the coverage of the parent node. The same is true for coverage assessments *'no coverage'*. However if a node is assessed with *'exact coverage'* it can be said that the parent node, which is the including context, has coverage *'too large'*. The same is true for coverage assessments of type *'too large'*:

```

cov_up(Node, Cov) :- cov_assess(Node, Cov).
cov_up(Node, large) :- ! cov_assess(Node, Cov)
    & parent(Node, Child)
    & cov_up(Child, exact).
cov_up(Node, large) :- ! cov_assess(Node, Cov)
    & parent(Node, Child)
    & cov_up(Child, large).

```

For propagation of coverage assessments towards the document tree's leaves it can be argued in a similar way. No statement can be made for a node if its parent has a coverage assessment of *'too large'*. In case a node has *'no coverage'* it can be assumed that all children also have *'no coverage'*. In case of an *'exact'* coverage assessment, child nodes must be of *'too small'* coverage. The same is true for nodes with coverage *'too small'*.

```

(* is there a sibling which has a
   coverage assessment already? *)
cov_sibling(Node) :- sibling(Node, Sibling)
    & cov_assess(Sibling, Cov).

cov_down(Node, Cov) :- cov_assess(Node, Cov).
cov_down(Node, small) :- ! cov_assess(Node, C1)
    & ! cov_sibling(Node)
    & parent(Parent, Node)
    & cov_down(Parent, small).
cov_down(Node, small) :- ! cov_assess(Node, C1)
    & ! cov_sibling(Node)
    & parent(Parent, Node)
    & cov_down(Parent, exact).
cov_down(Node, no) :- ! cov_assess(Node, C1)
    & cov_sibling(Node).

```

Figure ?? depicts the example tree with the implicit coverage assessments as computed by the rules described above.

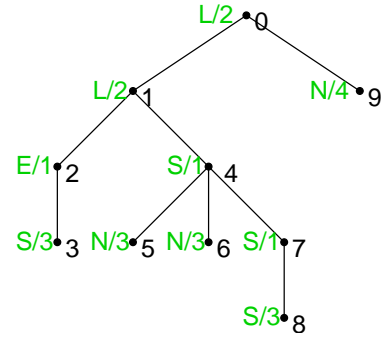


Figure 3: Example document with explicit and implicit assessments for coverage. The number of the rule by which the assessments are deduced is given after the respective slashes.

## 4 Quantization of relevance and coverage

In order to apply traditional recall/precision metrics values for the two dimensions relevance and coverage must be quantized by some function  $f_{quant}$  to a single relevance value:

$$f_{quant} : Relevance \times Coverage \rightarrow [0, 1]$$

$$(rel, cov) \mapsto f_{quant}(rel, cov)$$

Here, the set of relevance assessments is  $Relevance := \{0, 1, 2, 3\}$ , and the set of coverage assessments is  $Coverage := \{no, small, large, exact\}$ .

For example the quantisation function  $f_{strict}$  could be used to evaluate whether a given retrieval method is capable of retrieving highly relevant document components:

$$f_{strict}(rel, cov) := \begin{cases} 1 & \text{if } rel = 3 \text{ and } cov = exact, \\ 0 & \text{else.} \end{cases}$$

The following quantisation function  $f_{prelim}$  has been used in the preliminary evaluation:

$$f_{prelim}(rel, cov) := \begin{cases} 1 & \text{if } rel = 3 \\ & \text{and } cov \in \{small, large, exact\} \\ & \text{or } cov = exact \text{ and } rel > 2 \\ 0 & \text{else.} \end{cases}$$

## 5 Recall / precision computation

Given the complete set of (explicit and implicit) assessments and a quantisation function for mapping the assessments to a single relevance value evaluation metrics for standard document retrieval can be applied.

It is assumed that the results given for a topic come as a ranking of document components. Dependencies between the single elements of the ranking are ignored here. Furthermore it is assumed that users sequentially view through the ranking until they got a certain number  $NR$  of relevant elements. Based on this criterions, precision can be interpreted as the probability  $P(rel|retr)$  that a viewed document is relevant [Fuhr 02]:

$$P(rel|retr)(NR) := \frac{NR}{NR + esl_{NR}} = \frac{NR}{NR + j + s \cdot i / (r + 1)}$$

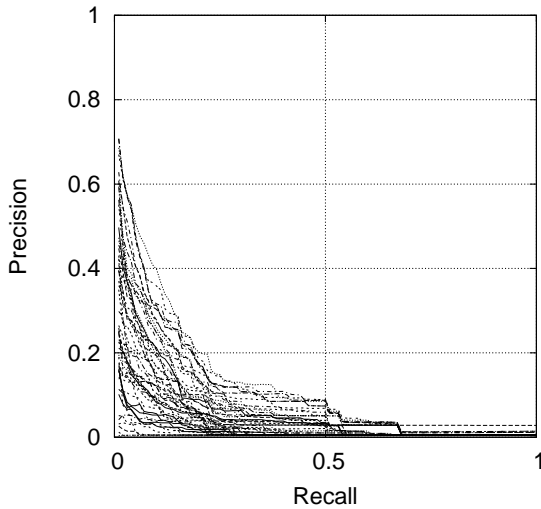


Figure 4: Recall / precision curves for all INEX 2002 submissions.

$esl_{NR}$  denotes the *expected search length*, that is the expected number non-relevant elements seen in the rank  $l$  with the  $NR$ th relevant document plus the number  $j$  of non-relevant documents seen in the ranks before (see [Hartmann 86] for details on the derivation).  $s$  is the number of relevant documents to be taken from rank  $l$ ;  $r$ ,  $i$ , are the numbers of (non-)relevant elements in rank  $l$ , respectively.

[Raghavan et al. 89] give a theoretical justification, that also intermediate real numbers can be used (here,  $n$  is the total number of relevant documents in the collection):

$$P(\text{rel}|\text{retr})(x) := \frac{x \cdot n}{x \cdot n + esl_{x \cdot n}} = \frac{x \cdot n}{x \cdot n + j + s \cdot i / (r + 1)}$$

This leads to an intuitive method for interpolation.

## 6 Implementation

1. Computation of implicit assessments for all paths given in the pools.
2. Mapping of relevance / coverage value pairs onto single relevance values.
3. Precision has been computed for 100 recall points.

## 7 Results

The evaluation method described in the previous sections has been applied to all submissions with all assessments available (at the time of processing assessments for 43 topics were available; they have been published under version 1.2 in the INEX {up,down}load area<sup>3</sup>).

Figure 4 contains the recall / precision curves for all INEX 2002 submissions. For computing the implicit assessments, the rules given in Section 3 have been used. Function  $f_{\text{prelim}}$  (Section 4) has been used for quantisation of the quality assessments. It is evident that the level of the curves is much lower than within TREC<sup>4</sup> and / or CLEF<sup>5</sup> evaluations.

<sup>3</sup><http://ls6-www.cs.uni-dortmund.de/ir/projects/inex/download/>

<sup>4</sup><http://trec.nist.gov/>

<sup>5</sup><http://www.clef-campaign.org/>

## 8 Drawbacks

- Elements in the ranking are regarded as being independent from each other. Given the possibility that several nodes of a given document may appear at arbitrary positions in the ranking, it is obvious that this assumption does not hold in reality.
- The quantisation function selected in the first version of our evaluation software discretises the two-dimensional INEX assessments to binary relevance assessments. A higher resolution would be more suitable.
- The rules for computation of implicit relevance assessments should be further elaborated.
- Results for CAS topics have been treated the same way as CO topic results. Obviously the rules for obtaining implicit assessments for CAS topics must be adjusted.

## References

- Fuhr, N.** (2000). Probabilistic Datalog: Implementing Logical Information Retrieval for Advanced Applications. *Journal of the American Society for Information Science* 51(2), pages 95–110.
- Fuhr, N.** (2002). *Information Retrieval. Lecture notes.* Technical report, Universität Dortmund, Fachbereich Informatik. [http://ls6-www.cs.uni-dortmund.de/ir/teaching/lectures/ir\\_ss02/folien/irskall.pdf](http://ls6-www.cs.uni-dortmund.de/ir/teaching/lectures/ir_ss02/folien/irskall.pdf).
- Hartmann, S.** (1986). *Effektivitätsmaße für die Bewertung von Rankingverfahren.* Studienarbeit, TH Darmstadt, FB Informatik, Datenverwaltungssysteme II.
- Raghavan, V.; Bollmann, P.; Jung, G.** (1989). A Critical Investigation of Recall and Precision as Measures of Retrieval System Performance. *ACM Transactions on Information Systems* 7(3), pages 205–229.
- Ullman, J. D.** (1988). *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville (Md.).